# Structural Neural Learning Machines

vorgelegt von
Diplom-Mathematiker, Diplom-Informatiker
Brijnesh Johannes Jain

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
- Dr.rer.nat. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof.r. K. Obermayer
Berichter: Prof. Dr. F. Wysotzki
Berichter: Prof. Dr. S. Wrobel

Tag der wissenschaftlichen Aussprache: 14. Juli 2005

Berlin 2005
D83

To my grandmother

# Abstract

Learning on structural patterns from examples gives rise to two challenging problems:

- *the combination of structural and statistical pattern recognition;*
- *the integration of the symbolic and sub-symbolic paradigm.*

The present work is devoted to the above issues, focusing on the construction of structural neural learning machines for adaptive processing of attributed graphs.

The first part of this thesis is concerned with the graph matching problem arising in the feed-forward pass of structural neural learning machines. We start with an introduction to structural pattern recognition from the perspective of the graph matching problem. To provide a generic mathematical framework, we transform all common graph matching problems to a generalized form of the maximum clique problem. Next, we derive an equivalent continuous quadratic program of the generalized clique problem. To minimize the quadratic program, we propose a selective attention control system (ACS) that ensures convergence to feasible solutions, requires no tuning of system parameters, and optimally adapts its system parameters during computation. Based on the proposed ACS, the remainder of the first part deals with advanced techniques for special graph matching problems. We present a two-stage neural network solution to the noisy graph isomorphism problem, an anytime ACS algorithm that can be interrupted at any time and return a result of guaranteed quality, and a self-organizing classifier for graphs combining the principle elimination of competition with concepts from anytime computing.

The second part of this thesis focuses on the problem of formulating learning rules to minimize structural error criteria as functions of attributed weight graphs. The starting point is the structural dot product. Though it is not an inner product, we show that the structural dot product of attributed graphs has the same geometrical properties as the standard dot product of vectors. In addition, the structural dot product gives rise to the key result of this contribution: parts of the powerful machinery of differential analysis can be applied to the domain of attributed graphs. In particular, we show that the gradient of a permutation invariant function on attributed graphs is a well-defined graph pointing in the direction of steepest increase. Since the structural dot product is only smooth almost everywhere, we accommodate nonsmooth techniques to minimize structural functions on attributed graphs. Using the mathematical toolkit at hand, we construct structural neural learning machines and analyze their basic principles and mechanisms.

The resulting structural neural learning machines constitute a generic framework that directly operates on attributed graphs. As such, they neither have to cope with uncontrollable loss of structural information when transforming graphs to simpler data types, nor with the reconstruction problem.

# Acknowledgements

I would like to thank the following people:

- my supervisor Prof. Dr. Fritz Wysotzki
- Prof. Dr. S. Wrobel for being the second referee of this work
- P. Geibel, S. Bischoff, and B. Hammer for scientific cooperation
- my nice collegues I. Ehmke, P. Geibel, C. Gips, C. Lippke, U. Schmid
- my project and master students
- J. Brinning from Modilingua Berlin for proofreading
- my parents and my sister
- Nina & Lea Anouk

# Contents

# List of Symbols

| | | |
|---|---|---|
| $(i,j)^{\phi}$ | image $\big(\phi(i),\phi(j)\big)$ of item $(i,j)$ | 16 |
| $E$ | set of edges | 12 |
| $E(X)$ | edge set of graph $X$ | 14 |
| $H$ | hypothesis | 191 |
| $I(X)$ | item set of graph $X$ | 15 |
| $L_W(\cdot)$ | linear map | 227 |
| $N(i)$ | set of neighbors of vertex $i$ | 15 |
| $N_X(i)$ | set of neighbors of vertex $i$ in $X$ | 15 |
| $R_\pi(\cdot)$ | rotation/relabeling operation | 227 |
| $S_W(\cdot)$ | parameterized structural dot product | 227 |
| $V$ | set of vertices | 12 |
| $V(X)$ | vertex set of graph $X$ | 14 |
| $W^i$ | weight graph of structural unit $i$ | 253 |
| $X \diamond Y$ | association graph of $X$ and $Y$ | 50 |
| $X$ | graph | 12 |
| $X[U]$ | induced subgraph of graph $X$ induced by $U$ | 16 |
| $X \bullet Y$ | structural dot product of graphs $X$ and $Y$ | 205 |
| $X \simeq Y$ | graphs $X$ and $Y$ are isomorphic | 17 |
| $X^\pi$ | permutation $\pi$ acting on vertices of graph $X$ | 202 |
| $[\cdot]$ | unlabeling operation | 227 |
| $[n:m]$ | discrete interval $\{n, n+1, \ldots, m\}$ of integers | 5 |
| $[n:s:m]$ | discrete interval from $n$ to $m$ in steps of $s$ | 5 |
| $[u_i]_\beta$ | limiter function | 68 |
| $\mathcal{G}_{\mathbb{B}}$ | set of simple or binary graphs | 17 |
| $\mathcal{G}_{\mathbb{U}}$ | set of normalized graphs | 17 |
| $\mathcal{G}_{\underline{\mathbb{R}}}$ | set of weighted graphs | 17 |
| $\mathbb{B}$ | set of binary values $\{0,1\}$ | 4 |
| $\mathbb{G}_{\mathcal{A}}$ | space of labeled graphs | 201 |

| | | |
|---|---|---|
| $\mathbb{G}_{\mathcal{A}}^{n}$ | set of complex attributed graphs of order $n$ | 200 |
| $\Delta(X)$ | maximum degree of graph $X$ | 15 |
| $\Delta(\mathcal{S})$ | diagonal of $\mathcal{S} \times \mathcal{S}$ | 4 |
| $\Delta_{\mathrm{co}}(X)$ | maximum co-degree of graph $X$ | 15 |
| $\Delta_{\mathrm{w}}(X)$ | maximum weight degree of graph $X$ | 16 |
| $\left[\mathbb{G}_{\mathcal{A}}\right]$ | space of unlabeled graphs | 201 |
| $\mathfrak{I}[E]$ | imaginary edges of a complex graph | 18 |
| $\mathfrak{I}[V]$ | imaginary vertices of a complex graph | 18 |
| $\mathfrak{I}[X]$ | imaginary part of a complex graph $X$ | 18 |
| $\mathbb{N}$ | set of natural numbers | 4 |
| $\Pi^{n}$ | set of $(n \times n)$-permutation matrices | 202 |
| $\mathbb{R}$ | set of real numbers | 4 |
| $\mathfrak{R}[E]$ | real edges of a complex graph | 18 |
| $\mathfrak{R}[V]$ | real vertices of a complex graph | 18 |
| $\mathfrak{R}[X]$ | real part of complex graph $X$ | 18 |
| $\mathbb{R}_{+}$ | set of positive real numbers | 4 |
| $\mathbb{R}_{+}^{0}$ | set of nonnegative real numbers | 4 |
| $\mathcal{A}$ | set of attributes | 13 |
| $\mathcal{B}_{F}(Z, \rho)$ | open ball with center $Z \in \mathbb{G}_{\mathcal{A}}$ and radius $\rho$ | 223 |
| $\mathcal{B}_{\delta}([Z], \rho)$ | open ball with center $[Z] \in \left[\mathbb{G}_{\mathcal{A}}\right]$ and radius $\rho$ | 223 |
| $\mathcal{C}_{X}$ | set of all cliques of graph $X$ | 46 |
| $\mathcal{C}_{X}^{*}$ | set of all maximum weight cliques of graph $X$ | 46 |
| $\mathcal{C}_{X}^{\times}$ | set of all maximal weight cliques of graph $X$ | 46 |
| $\mathcal{D}(\phi)$ | domain of morphism $\phi$ | 16 |
| $\mathcal{H}$ | hypothesis space | 191 |
| $\mathcal{I}_{X,Y}$ | set of isomorphisms from $X$ to $Y$ | 17 |
| $\mathcal{M}_{X,Y}$ | set of partial morphisms from $X$ to $Y$ | 25 |
| $\mathcal{M}_{X,Y}^{\mathfrak{p}}$ | set of $\mathfrak{p}$-morphisms from $X$ to $Y$ | 47 |
| $\mathcal{R}(X, Y)$ | set of optimal rotations of $X$ towards $Y$ | 207 |
| $\mathcal{R}(\phi)$ | range of morphism $\phi$ | 17 |
| $\mathcal{R}_{X,Y}$ | set of optimal relabelings of $X$ and $Y$ | 207 |
| $\mathcal{S}$ | arbitrary set | 4 |
| $\mathcal{S}^{[k]}$ | set of $k$-element subsets of $\mathcal{S}$ | 4 |
| $\mathcal{S}^{n \times m}$ | set of $(n \times m)$-matrices with entries from $\mathcal{S}$ | 5 |
| $\mathcal{S}_{n}$ | set of all permutations acting on $\mathcal{S}_{XY}$ | 116 |
| $\mathcal{V}_{[X]}$ | vector representation of the unlabeled graph $[X]$ | 202 |

| | | |
|---|---|---|
| $\mathcal{X}$ | input space | 191 |
| $\mathcal{Y}$ | output space | 191 |
| $\mathcal{Z}$ | labeled training sample | 191 |
| $\Theta$ | threshold function | 68 |
| $\Theta_P$ | probabilistic threshold function | 69 |
| $\mathbb{U}$ | unit interval $[0, 1]$ | 5 |
| $\mathrm{vec}(\boldsymbol{X})$ | vector representation of matrix $\boldsymbol{X}$ | 42 |
| $\deg_{\mathrm{co}}(i)$ | co-degree of vertex $i$ | 15 |
| $\deg(i)$ | degree of vertex $i$ | 15 |
| $\delta(X)$ | minimum degree of graph $X$ | 15 |
| $\delta(X, Y)$ | structural Frobenius metric between graphs $X$ and $Y$ | 213 |
| $\delta_{\mathrm{co}}(X)$ | minimum co-degree of graph $X$ | 15 |
| $\delta_{\mathrm{w}}(X)$ | minimum weight degree of graph $X$ | 16 |
| $\epsilon$ | null attribute | 13 |
| $\kappa$ | comptability function | 39 |
| $\kappa_{\boldsymbol{ij}}$ | compatibility value of items $\boldsymbol{i}$ and $\boldsymbol{j}$ | 25 |
| $\omega(C)$ | weight of clique $C$ | 44 |
| $\omega(X)$ | weight of $X$ | 26 |
| $\overline{\mathbb{R}}$ | set of real numbers including $+\infty$ | 4 |
| $\partial f(\boldsymbol{\xi})$ | generalized gradient of $f$ at $\boldsymbol{xi}$ | 241 |
| $\phi$ | morphism between graphs | 16 |
| $\pi$ | permutation | 116 |
| $\mathrm{supp}(f)$ | support of pointwise maximizer $f$ | 245 |
| $\underline{\mathbb{R}}$ | set of real numbers including $-\infty$ | 4 |
| $\mathbf{1}_C$ | charactersitic vector of vertex subset $C$ | 38 |
| $\boldsymbol{D}_{\boldsymbol{X}}$ | diagonal matrix of matrix $\boldsymbol{X}$ | 6 |
| $\boldsymbol{I}_n$ | $(n \times n)$-dimensional identity matrix | 5 |
| $\boldsymbol{J}_n$ | $(n \times n)$-dimensional matrix of all ones | 6 |
| $\boldsymbol{O}_{\boldsymbol{X}}$ | off-diagonal matrix of matrix $\boldsymbol{X}$ | 6 |
| $\boldsymbol{X} \otimes \boldsymbol{Y}$ | Kronecker $\kappa$-product of graphs $X$ and $Y$ | 41 |
| $\boldsymbol{X}$ | matrix | 5 |
| $\boldsymbol{X} \bullet \boldsymbol{Y}$ | matrix dot product of $\boldsymbol{X}$ and $\boldsymbol{Y}$ | 204 |
| $\boldsymbol{X}^\top$ | transpose of matrix $\boldsymbol{X}$ | 5 |
| $\boldsymbol{d}_{\boldsymbol{X}}$ | diagonal of matrix $\boldsymbol{X}$ | 6 |
| $\boldsymbol{e}_n$ | $n$-dimensional vector of all ones | 5 |
| $\boldsymbol{u}_i$ | $i$-th unit vector | 5 |

| | | |
|---|---|---|
| $\boldsymbol{x}$ | vector (n-tuple) | 5 |
| $\boldsymbol{x} \cdot \boldsymbol{y}$ | dot product of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ | 203 |
| $\boldsymbol{x}^\top$ | transpose of $\boldsymbol{x}$ | 5 |
| $\boldsymbol{x}_{:j}$ | $j$-th column of matrix $\boldsymbol{X}$ | 5 |
| $\boldsymbol{x}_{i:}$ | $i$-th row of matrix $\boldsymbol{X}$ | 5 |
| $\deg_{\mathrm{w}}(i)$ | weight degree of vertex $i$ | 16 |
| $\overline{E}$ | set of non-edges | 12 |
| $\overline{E}(X)$ | non-edge set of graph $X$ | 14 |
| $\overline{\mathcal{B}}_F(Z, \rho)$ | closed ball with center $Z \in \mathbb{G}_{\mathcal{A}}$ and radius $\rho$ | 223 |
| $\overline{\mathcal{B}}_\delta([Z], \rho)$ | closed ball with center $[Z] \in \left[\mathbb{G}_{\mathcal{A}}\right]$ and radius $\rho$ | 223 |
| $\widehat{f}(\cdot)$ | structural function on unlabeled space $\left[\mathbb{G}_{\mathcal{A}}\right]$ | 226 |
| $g_\beta$ | logistic function | 68 |
| $g_i(\cdot)$ | arbitrary transfer function of standard/structural unit $i$ | 67 |
| $h^i$ | bias/threshold of unit $i$ | 254 |
| $h_i$ | external input of standard unit $i$ | 67 |
| $i^\phi$ | image $\phi(i)$ of vertex $i$ | 16 |
| $l$ | loss function | 191 |
| $u_i$ | activation of standard/structural unit $i$ | 67 |
| $vecX_P$ | permuted matrix $\boldsymbol{X}$ | 202 |
| $w_{ij}$ | weight connecting units $i$ and $j$ | 67 |
| $x_i$ | output of standard/structural unit $i$ | 67 |

# 1 Introduction

In most areas of pattern recognition it is common practice to represent data in terms of feature vectors residing in a Euclidean space, because the Euclidean space provides powerful analytical techniques for data analysis usually not available for other representations. Feature-based vector representations, however, are too limited for many relevant application areas including domains such as computer vision, chemical graphs, bioinformatics, or text mining.

The poor representational capabilities of feature vectors triggered the field of *structural pattern recognition* in the early 1970s. The premise of structural pattern recognition is that the data to be analyzed are composed of interrelated constituents. In addition, the data may be corrupted by noise and structural variation. It turns out that dynamic data structures like strings, trees, and graphs are a more versatile and expressive tool for repesenting structured data. Typical problems in structural pattern recognition are classification and clustering of structural descriptions of images in computer vision [30, 131, 183, 237, 310] and quantitative structure-activity relationships in chemistry [261, 262, 306, 341, 342].

The expressive power and flexibility of graphs to describe structured patterns is undermined by the following problems, which we identify as the two *fundamental problems* of structural pattern recognition:

1. *Computational intractability* arises when comparing two structural description. This is a common task, because many methods in pattern recognition are based on the fundamental concept of proximity. Measuring the proximity of two graphs in terms of their structural characteristics is generally an NP-complete *graph matching problem*.
2. *Analytical intractability* is caused by the lack of practical algebraic operations such as the sum of graphs. Thus, useful characterizations of an inner product or basic statistical concepts like the mean of a set of graphs are undefined or at least intricate. As a consequence, many traditional statistical pattern recognition paradigms and analytical methods are inapplicable for graphs.

Consequently, the fundamental problems of structural pattern recognition triggered two main directions of research:

1. Development of graph matching algorithms.
2. Combination of structural and statistical pattern recognition.

Research on devising efficient graph matching algorithms is almost as old as the field of structural pattern recognition. Meanwhile, this area has developed into a mature field that has produced a number of powerful matching methods. Nevertheless, devising optimization methods for NP-complete problems, like the graph matching problem, will remain an active research direction – at least unless proven $P = NP$.

Research on the second fundamental problem, analytical intractability, is still a widely unexplored, but intriguing and emergent field. With the advent of efficient optimization methods and powerful computational resources, attention gradually shifted towards bridging the gap between structural and statistical pattern recognition. Most work abandons the unexploited domain of graphs towards more accessible domains that provide methodologically sound and technically mature methods for intelligent data analysis. But structural pattern recognition also demands techniques, which directly operate on graphs [47, 195]. This demand is in line with the goal of the neural network community to integrate the symbolic and sub-symbolic paradigm [97, 133, 340]. Current neural approaches operate on a a rather restricted class of linear and quasi-linear structures like strings, trees, and directed acyclic graphs.

The goal of this thesis is to contribute to both issues, (1) the combination of structural and statistical pattern recognition, and (2) integration of symbolic and sub-symbolic learning. To this end, we construct structural neural learning machines for adaptive processing of arbitrary graphs. The elementary building block is a structural model of a neuron, which we refer to as a *structural unit*, or *s-unit*. Construction of s-units poses two questions:

1. *How should the activation of an s-unit be computed*?
2. *How should the weights associated with an s-unit be adjusted*?

To answer the first question, we propose two modifications of standard units. In the standard case, a conventional unit is associated with a weight vector $\boldsymbol{w}$. For a given input vector $\boldsymbol{x}$, the activation of that unit is then determined by the dot product $\boldsymbol{w}^\top \boldsymbol{x}$. To compute the activation of an s-unit, we replace the weight vector by a weight graph, and the dot product by a similar concept for graphs called the *structural dot product*. In the guise of the structural dot product, we encounter both fundamental problems of structural pattern recognition:

1. *Computational intractability.* Determining the structural dot product is an NP-complete graph matching problem.
2. *Analytical intractability.* As opposed to the standard dot product, the structural dot product is not bilinear.

To avoid uncontrolled loss of structural information, we accept that determining the activation of an s-unit is NP-complete. Thus, all we can do is devise new or improve existing graph matching algorithms for computing the structural dot product.

Since the structural dot product is not bilinear, an answer to the second question is more complicated than it is for standard units. To overcome analytical intractability, we show that the structural dot product is the key to another geometry of structures and to differential analysis of permutation invariant functions on graphs. Entering both realms, we derive the necessary theoretical results to establish a sound mathematical foundation, upon which we formulate structural learning rules. These rules exploit gradient-like information to minimize structural error criteria as functions of the weight graphs.

### Thesis Outline

Both fundamental problems of structural pattern recognition also pervade the construction of structural neural learning machines and therefore give rise to a natural segmentation of this thesis in two parts. Part I is concerned with the graph matching problem to determine the activation of an s-unit. Part II deals with the problem of learning in structural domains. Parts of the results have been already published in [108, 134, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189].

#### *Part I*

The first part of this thesis is devoted to the graph matching problem. Chapter 2 provides an overview of relevant issues in structural pattern recognition. Moreover, it establishes a common basis in terms of definitions and notations, which the subsequent chapters of both parts will refer to.

Chapters 3 and 4 present a generic view on all common graph matching algorithms reported in the literature. For this, we generalize classical clique problems to the maximum weight clique problem, where weights are assigned to both the vertices and edges of the underlying graph. In Chapter 3, we propose sufficient conditions to view graph matching as weighted clique search. Furthermore, we present an equivalent continuous characterization of the maximum weight clique problem. Based on the continuous formulation, Chapter 4 offers a generic solution to a broad range of graph matching problems. The main contribution is a novel selective attention control system that ensures convergence to feasible solutions, requires no tuning of system parameters, and optimally adapts the system parameters during run time. The last feature allows us to interpret the system in cognitive-psychological terms of selective attention.

Chapters 5, 6, and 7 are devoted to advanced techniques for special graph matching problems. In Chapter 5, we propose a noise-tolerant neural algorithm for the graph isomorphism problem. Chapter 6 introduces the idea of real-time neural networks for combinatorial optimization problems that can be interrupted at any point in time and provide meaningful results. In the last chapter of Part I, we propose a self-organizing classifier for graphs by combining the principle *elimination of competition* with concepts from anytime computing.

***Part II***

The second part of this thesis is devoted to the construction of structural neural learning machines. We begin with an overview in Chapter 8 and provide the necessary background relevant for subsequent chapters.

Chapters 9, 10, and 11 constitute the theoretical foundation to structural learning. Chapter 9 introduces the structural dot product and develops another geometry of structures. Based on the structural dot product, Chapter 10 draws relevant concepts from differential analysis to the domain of attributed graphs. Since the structural dot product is not differentiable, we accommodate results from nonsmooth analysis for functions on graphs in Chapter 11.

Chapters 12, 13, and 14 propose structural neural learning machines that adaptively process attributed graphs. In Chapter 12, we are concerned with structural single-layer and in Chapter 13 with structural multi-layer feed-forward networks. Chapter 14 is devoted to structural competitive networks.

Finally, Chapter 16 presents an overall conclusion of this thesis.

***Appendix***

Appendix A describes algorithms not developed but used in this thesis, and Appendix B describes experimental settings.

## 1.1 Basic Notations and Definitions

We conclude the introduction with a compilation of basic notations and definitions that are used throughout the thesis.

By $\mathbb{R}$ we denote the set of real numbers, by $\mathbb{R}_+$ the set of positive real numbers, and by $\mathbb{R}_+^0$ the set of nonnegative real numbers. We write $\overline{\mathbb{R}}$ to denote $\mathbb{R} \cup \{\infty\}$ and $\underline{\mathbb{R}}$ for $\mathbb{R} \cup \{-\infty\}$. The set of natural numbers including zero is denoted by $\mathbb{N}$. By $\mathbb{B} = \{0, 1\}$ we denote the set of binary values.

Let $\mathcal{S}$ be a set. The set $\Delta(\mathcal{S}) = \{(i, i) : i \in \mathcal{S}\}$ is called the *diagonal* of $\mathcal{S} \times \mathcal{S}$. By $\mathcal{S}^{[k]}$ we denote the set of all $k$-element subsets of $\mathcal{S}$.

Let $x < y$ be real numbers. Then

- $[x, y] = \{z \in \mathbb{R} : x \leq z \leq z\}$ is the closed interval of $x$ and $y$
- $]x, y] = \{z \in \mathbb{R} : x < z \leq z\}$ is the left open interval of $x$ and $y$
- $[x, y[ = \{z \in \mathbb{R} : x \leq z < z\}$ is the right open interval of $x$ and $y$
- $]x, y[ = \{z \in \mathbb{R} : x < z < z\}$ is the open interval of $x$ and $y$.

The unit interval is of the form

- $\mathbb{U} = [0, 1]$.

For natural numbers $n < m$ we denote the discrete interval of $n$ and $m$ by

- $[n{:}m] = \{k \in \mathbb{N} \,:\, n \leq k \leq m\}$.

We denote the discrete interval from $n$ to $m$ in steps of $s$ by

- $[n : s : m] = \{n + ks \,:\, n \leq n + ks \leq m,\, k \in \mathbb{N}\}$.

### Matrix Algebra

Let $\mathcal{S}$ be a set. The set of $(n \times m)$-matrices with entries from $\mathcal{S}$ is denoted by $\mathcal{S}^{n\times m}$. Matrices from $\mathcal{S}^{n\times m}$ appear as upper case bold face letters $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$. N-tuples of $\mathcal{S}^n = \mathcal{S}^{n\times 1}$ are denoted by lower case letters with bold face $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$. We always consider n-tuples as columns. By $\boldsymbol{X}^\mathsf{T}$ and $\boldsymbol{x}^\mathsf{T}$ we denote the transpose of matrix $\boldsymbol{X}$ and n-tuple $\boldsymbol{x}$, respectively.

Frequently, $\mathcal{S}$ is a set of n-tuples of the form $\mathcal{S} = \mathcal{T}^d$ for some set $\mathcal{T}$. Then a matrix $X$ of $\mathcal{S}^{n\times m}$ is of the form $\boldsymbol{X} = (\boldsymbol{x}_{ij})$, where the components $\boldsymbol{x}_{ij}$ of $\boldsymbol{X}$ are n-tuples. When $\boldsymbol{X} \in \mathbb{R}^{n\times m}$, the components of $\boldsymbol{X}$ appear in normal face letters $x_{ij}$. By

$$\boldsymbol{x}_{i:} = (\boldsymbol{x}_{i1}, \boldsymbol{x}_{i2}, \ldots, \boldsymbol{x}_{im})^\mathsf{T},$$

we denote the $i$-th row of $\boldsymbol{X}$ and by

$$\boldsymbol{x}_{:j} = \begin{pmatrix} \boldsymbol{x}_{1j} \\ \boldsymbol{x}_{2j} \\ \vdots \\ \boldsymbol{x}_{nj} \end{pmatrix},$$

the $j$-th column of $\boldsymbol{X}$.

The vector $\boldsymbol{e}_n \in \mathbb{R}^n$ of all ones is of the form

$$\boldsymbol{e}_n = (1, \ldots, 1)^\mathsf{T}.$$

By $\boldsymbol{u}_i \in \mathbb{R}^n$, we denote the $i$-th unit vector

$$\boldsymbol{u}_i = (0, \ldots, 0, 1, 0, \ldots 0)^\mathsf{T},$$

where only the $i$-th component has value one and all other components are zero.

Let $\boldsymbol{X} = (x_{ij})$ be an $(n \times n)$-matrix from $\mathbb{R}^{n\times n}$. By $\boldsymbol{I}_n$ we denote the identity-matrix

$$\boldsymbol{I}_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

and by $\boldsymbol{J}_n$ the $(n \times n)$-matrix from $\mathbb{R}^{n \times n}$ of all ones

$$\boldsymbol{J}_n = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}.$$

The *diagonal matrix* $\boldsymbol{D}_{\boldsymbol{X}}$ of $\boldsymbol{X}$ is an $(n \times n)$-matrix of the form

$$\boldsymbol{D}_{\boldsymbol{X}} = \begin{pmatrix} x_{11} & 0 & \cdots & 0 \\ 0 & x_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_{nn} \end{pmatrix}.$$

By $\boldsymbol{O}_{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{D}_{\boldsymbol{X}}$ we denote the *off-diagonal matrix* of $\boldsymbol{X}$. The *diagonal* $\boldsymbol{d}_{\boldsymbol{X}}$ of $\boldsymbol{X}$ is the $n$-dimensional vector $\boldsymbol{d}_{\boldsymbol{X}} = (x_{11}, x_{22}, \ldots, x_{nn})^{\mathsf{T}}$.

# I Graph Matching

# 2 The Graph Matching Problem

The aim of this chapter is to set a basic context to outline the graph matching problem. We systematically dissect different concepts related to the graph matching problem that are frequently summarized to a single functional unit in the literature. Our atomized approach will turn out to be useful in gaining more insight into the common structure of diverse graph matching problems. In addition, this chapter serves to introduce the terminology upon which subsequent chapters will build.

## 2.1 Introduction

One primary goal of pattern recognition is to uncover the hidden structure of a given dataset in order to generate a compact representation of the data and to enable symbolic data processing concepts [145]. In domains where traditional techniques from statistical pattern recognition do not directly apply, the concept of proximity may provide a bridge between structural and statistical pattern recognition [120]. If data is represented by attributed graphs, a proximity-based pattern recognition system can be characterized by the following basic steps:

1. *Graph-based representation*
2. *Definition of a structural proximity measure*
3. *Graph matching formulation*
4. *Optimization*
5. *Proximity-based data analysis*

Contemporary literature often summarizes points (1)-(4) or at least (2)-(3) when referring to the graph matching problem. We do not follow this convention in order to accentuate the semantic and functional differences between the individual stages in structural pattern recognition, and to provide a foundation to pursue a rigorous mathematical treatment in subsequent chapters.

Several applications of pattern recognition based on structural proximities have been reported in the literature. Examples include image analysis [83, 284, 330], document processing [22, 235, 346], chemoinformatics [61, 295, 314], image databases [31, 157, 287], and video analysis [326].

### An Example

To illustrate the problem, let us consider the somewhat imaginary problem of recognizing *mandala diagrams*. Mandala is the Sanskrit word for circle. Two examples of mandala diagrams are shown in Figure 2.1. A mandala contains different geo-

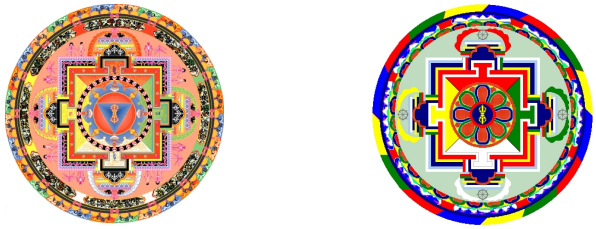


**Figure 2.1** Two Asian mandalas.

metric figures like circles, triangles, and other shapes, as well as symbols such as e.g. diamonds, bells, and wheels. There are many semantic types of mandalas, each of which is characterized by a certain arrangement of geometrical shapes and symbols. Assume that we want to automate the process of classifying a given set of mandala diagrams according to their semantic type.

**Graph-Based Representation.** A central aspect in almost any pattern recognition problem is that of achieving an appropriate representation in order to successfully extract the structure from the given data. We therefore require that an appropriate representation should simply and naturally reveal the structural relationships between the components, and should enable us to express the true but hidden structure of the data. If our data is of a complex nature, graph-based representation might be appropriate to capture relevant structural information.

In our example, we assume that a preprocessor and feature extractor identifies all symbols and geometrical shapes together with their absolute coordinates in the mandala diagram.[1] Together with their spatial relationships, the identified entities are then represented by an attributed graph. In order to classify the given mandala, this representation should be invariant to translation, scaling, and rotation.

We consider graph-based representations in Section 2.2. For a discussion on closely related subproblems like preprocessing, feature extraction, or noise, we refer to [32, 77].

**Structural Proximity Measure.** What we seek is a representation in which attributed graphs that belong to the same *category* or *concept* are somehow *closer* to one another than attributed graphs belonging to different categories. The common technique for assessing closeness among patterns is based on the definition

1. How a preprocessor or feature extractor retrieves the desired information is a problem in computer vision and will not be considered here. A recent explanation of that issue can be found in the dissertation by Bischoff [30].

of a suitable proximity measure. A proximity for graph-based representations is usually a similarity or distance function defined on pairs of attributed graphs. Structural proximity measures are proximities that evaluate structurally consistent or inconsistent subgraphs.

To determine the semantic type of a mandala diagram, we search for the presence or absence of characteristic structural arrangements of symbols and geometrical shapes. Since mandalas are typically handcrafted and therefore hardly identical, an exact correspondence between entities and their spatial relationship is unlikely. Therefore, some noise model is required to cope with structural errors or distortions.

Section 2.3 formally introduces and discusses structural proximity measures.

**Graph Matching Problem.** The key problem of most structural proximity measures is that their computation is an NP-complete combinatorial optimization problem [24, 367]. Optimization problems associated with a structural proximity measure constitute the class of graph matching problems — the central theme not only of this chapter, but also of Part I of this thesis. In Section 2.4, we introduce the graph matching problem and present some important and commonly used problem instances.

**Optimization.** The specific formulation of the graph matching problem determines the optimization methods we can apply to solve that problem. Section 2.5 provides an overview of existing optimization techniques for the graph matching problem.

**Data Analysis.** Once the proximities are evaluated, we can transfer them to a module for data analysis. Examples of such systems are pairwise clustering [145], nearest neighbor methods, the pairwise proximity classifier proposed by Graepel et al. [126], dissimilarity kernel classifiers proposed by Pekalska et al. [278], and Support Vector Machines [106], provided that the structural proximity measure at hand is a kernel. Since we focus on graph matching problems, we do not further discuss methods for data analysis. Instead, we refer to the thesis of Pekalska [277] for a detailed exposition on proximity-based pattern recognition.

To refer to our introductory example, a nearest neighbor classifier for categorizing mandalas according to their semantic type is a simple and a natural choice.

**Final Remark.** In the following, it is now reasonable to consider more manageable synthetic image patterns rather than complex mandala diagrams.

## 2.2 Graph-Based Representations

It is generally accepted that attributed graphs are an appropriate tool to model complex objects from diverse domains such as chemistry [15, 38, 351], computer vision [19, 20, 328], and power-law distributed or scale-free networks like Internet router topologies [84], the World-Wide Web [210], and metabolic networks [190].

A *directed graph* is a pair $X = (V, E)$ of finite sets satisfying $E \subseteq V^2 \setminus \Delta(V)$, where $\Delta(V) = \{(i, i) : i \in V\}$ is the diagonal of $V^2$. The elements of $V$ are the *vertices* of $X$; the elements of $E$ are its *arcs*. An arc $(i, j) \in E$ is *oriented* from

$i$ to $j$. We call $X$ an *undirected* or *simple graph* if $(i,j)$ implies $(j,i)$ for all arcs $(i,j) \in E$. The elements of $E$ in an undirected graph are its *edges*. Occasionally, we also write $\{i,j\}$ to denote an edge connecting vertex $i$ and $j$. Pairs $(i,j)$ that are not in $E \cup \Delta(V)$ are called *non-edges*. By $\overline{E}$ we denote the set of all non-edges of $X$. For convenience of presentation, we will consider only simple graphs. Nevertheless, we stress that the whole theory also applies for directed graphs.

Figure 2.2 shows the usual way to plot an undirected graph. We draw a dot for each vertex and join two of these dots by a line if the corresponding two vertices are connected by an edge. How these dots and lines are plotted is irrelevant. What is relevant is the structural information about vertices and their pairwise relationships.

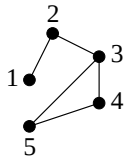



**Figure 2.2** Plot of graph $X = (V,E)$ with vertex set $V = [1:5]$ and edge set $E = \big\{\{1,2\},\{2,3\},\{3,4\},\{3,5\},\{4,5\}\big\}$.

In structural pattern recognition, simple graphs are useful to describe the basic constituents of a complex or structured object. Basic constituents of a complex object are atomic entities, and the relationship between pairs of atomic entities. In a graph-based representation of a complex object, vertices represent atomic entities and edges a certain relationship between entities. Figure 2.3 shows a graphical representation of a simple image pattern. The atomic entities of the pattern are segments of identical color. A pair of entities is related if their corresponding segments are adjacent. Referring to our original problem in recognizing mandala diagrams, segments in our simplistic image pattern may represent geometrical shapes or symbols.

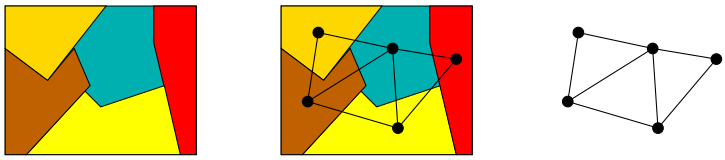


**Figure 2.3** Image pattern and its representation as a simple graph. The left part shows the raw data. The middle part identifies the basic constituents of the image pattern. Finally, the right part shows the structural representation of the image pattern as a simple graph.

Simple graphs do not describe properties of entities and their relationship. Such a representation is often not appropriate for applications in structural pattern recognition. For example, Figure 2.4 shows a different image pattern, which is

usually not considered similar with the first pattern from Figure 2.3. Yet their structural representations as simple graphs are equivalent.

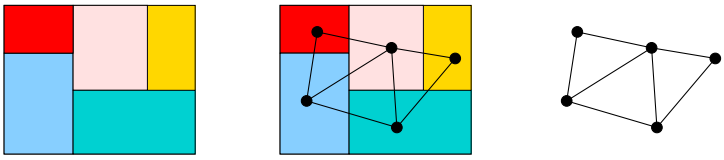


**Figure 2.4** A second image pattern and its representation as a simple graph. The structural representation of this image pattern is equivalent to that of Figure 2.4.

In our search for a more appropriate representation to distinguish between both patterns, we might annotate vertices and edges with attributes that describe relevant features of the entities and their relationships. Examples of vertex features are form, surface area, and color of the corresponding segments. Edge features might be the distance between the centers or the length of the common boundary of corresponding adjacent segments. Figure 2.5 shows representations of both sample patterns as attributed graphs. To unclutter the plots, only color attributes are annotated at the vertices.

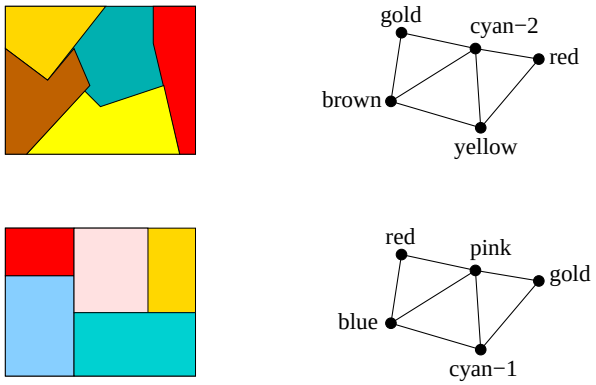



**Figure 2.5** Image patterns and their representations as attributed graphs. For convenience of presentation, only color attributes are attached to vertices.

In order to formalize the concept of attributed graphs, let $\mathcal{A}$ be a set of *attributes* containing a distinguished symbol $\epsilon$ denoting the *void* or *null* attribute. If vertex and edge attributes need to be explicitly distinguished, the set $\mathcal{A}$ can be decomposed into disjoint sets $\mathcal{A}_V$ of vertex attributes and $\mathcal{A}_E$ of edges attributes, for example by introducing a flag as an additional attribute. Since vertex attributes must be distinct from $\epsilon$, we require that the null attribute $\epsilon$ is in $\mathcal{A}_E$.

Unless otherwise stated, we do not impose any restriction on $\mathcal{A}$. That is, an attribute from $\mathcal{A}$ might be a symbol, numerical value, feature vector, function, or anything that is appropriate to describe the basic constituents of a complex object.

We denote attributes from $\mathcal{A}$ by bold face letters $\boldsymbol{a}$ to indicate that their potential domain is broader than the domain of one-dimensional symbolic or numerical feature spaces.

An *attributed graph* is a triple $X = (V, E, \boldsymbol{X})$ consisting of a simple graph $(V, E)$ and an *(attributed) adjacency matrix* $\boldsymbol{X} = (\boldsymbol{x}_{ij}) \in \mathcal{A}^{|V| \times |V|}$ such that

1. $\boldsymbol{X}^\top = \boldsymbol{X}$
2. $\boldsymbol{x}_{ij} = \epsilon \;\Leftrightarrow\; (i,j) \in \overline{E}$

for all $i, j \in V$. The first condition requires that $\boldsymbol{X}$ is symmetric to conform to bidirectional edges of undirected graphs. The second condition colors each vertex and each edge with an attribute from $\mathcal{A} \setminus \{\epsilon\}$. Non-edges are colored with the null attribute $\epsilon$. Although the adjacency matrix captures the whole structure of $X$, it is technically advantageous to carry along the sets $V$ and $E$. For convenience, the term *graph* occasionally refers to an *attributed graph*.

Given the denotation of an attributed graph, we have an expressive tool to appropriately describe the basic constituents of complex objects. In addition, graph-based representations admit descriptions of objects that are invariant under scaling, translation, and rotation. A representation of complex objects by fixed-dimensional feature vectors may result in a loss of structural information. What makes feature vectors too unwieldy to capture structural information is their fixed dimension and strict ordering. Fixed dimensionality bounds the number of atomic entities of a complex object that can be recorded by a feature vector to a maximum limit. Strict ordering of the components of a feature vector may result in completely different representations of the same object and is therefore inappropriate for data analysis.

In the remainder of this section, we introduce important notations and definitions that will be used throughout this thesis. In addition, we introduce two nonstandard concepts, namely *items* and *complex graphs*.

### 2.2.1 Notations and Basic Definitions

A graph with vertex set $V$ is called graph on $V$. The vertex set of a graph $X$ is referred to as $V(X)$, its edge set as $E(X)$, its non-edge set as $\overline{E}(X)$, and its attributed adjacency matrix as $\boldsymbol{X} = (\boldsymbol{x}_{ij})$. By $\mathcal{G}_\mathcal{A}$ we denote the set of attributed graphs with attributes from $\mathcal{A}$.

The number $|X| = |V|$ of vertices of a graph $X$ is its *order* or *size*. Here we only consider graphs of finite order. We use a labeling of the vertex set $V$ in order to deal with it conveniently. A *labeling* is a bijective mapping $\nu : V \to \{1, \ldots, |V|\}$. Using $\nu$, each vertex of $V$ can be identified with a certain number from $[1\!:\!|V|]$. We use these numbers to name the vertices.

### 2.2.2 Items of a Graph

The notion of an *item* is rarely used in graph-based representation although it is useful to unclutter the text from tedious case distinctions.

We call $I(X) = V(X) \times V(X)$ the set of *items* of a graph $X$. An item $\boldsymbol{i} = (i_1, i_2)$ of $X$ gives rise to the following notations:

- $\boldsymbol{i} \in V(X)$ if $i_1 = i_2$,
- $\boldsymbol{i} \in E(X)$ if $(i_1, i_2) \in E(X)$,
- $\boldsymbol{i} \in \overline{E}(X)$ if $(i_1, i_2) \in \overline{E}(X)$,
- $\boldsymbol{x_i} = \boldsymbol{x}_{i_1 i_2}$.

Moreover, we may now identify $\Delta\big(V(X)\big)$ and $V(X)$.

### 2.2.3 Degree of a Vertex

Let $X = (V, E, \boldsymbol{X})$ be a graph. Two vertices $i, j$ of $X$ are *adjacent*, or *neighbors*, if $(i, j) \in E(X)$. The set of neighbors of a vertex $i$ in $X$ is denoted by $N_X(i)$. Here, as elsewhere, we drop the subscript referring to the graph under consideration if the reference is clear, and briefly write $N(i)$. The set of vertices not adjacent to vertex $i$ is given by

$$\overline{N}(i) = V \setminus \big\{N(i) \cup \{i\}\big\}.$$

A vertex $i$ is *incident* with an edge $(k, l) \in E(X)$ if either $i = k$ or $i = l$. An *isolated vertex* is a vertex not incident to an edge. The graph $X$ is *complete* if all vertices of $X$ are pairwise adjacent.

The *degree* $\deg(i)$ of a vertex $i$ in $X$ is the number $\big|N(i)\big|$ of neighbors of $i$. The numbers

$$\begin{aligned}\delta(X) &= \min_{i \in V} \deg(i)\\ \Delta(X) &= \max_{i \in V} \deg(i)\end{aligned}$$

are the *minimum* and *maximum degree* of $X$. The *co-degree* $\deg_{\mathrm{co}}(i)$ of $i$ is the number of vertices not adjacent to $i$. Hence, we have $\deg_{\mathrm{co}}(i) = |X| - \deg(i)$. The *minimum* and *maximum co-degree* are defined by

$$\begin{aligned}\delta_{\mathrm{co}}(X) &= \min_{i \in V} \deg_{\mathrm{co}}(i)\\ \Delta_{\mathrm{co}}(X) &= \max_{i \in V} \deg_{\mathrm{co}}(i).\end{aligned}$$

Suppose that $X = (V, E, \boldsymbol{X})$ is a graph with attributes from $\underline{\mathbb{R}}$, where $\epsilon = -\infty$. The *weight degree* $\deg_{\mathrm{w}}(i)$ of a vertex $i$ in $X$ is the quantity

$$\deg_{\mathrm{w}}(i) = \sum_{j \in N(i)} x_{jj}.$$

Thus, $\deg_{\mathrm{w}}(i)$ is the total weight over all neighbors of $i$. Similarly,

$$\begin{aligned}\delta_{\mathrm{w}}(X) &= \min_{i \in V} \deg_{\mathrm{w}}(i)\\ \Delta_{\mathrm{w}}(X) &= \max_{i \in V} \deg_{\mathrm{w}}(i)\end{aligned}$$

are the *minimum* and *maximum weight degree* of $X$.

### 2.2.4 Subgraphs and Cliques

Let $X = (V, E, \boldsymbol{X})$ and $X' = (V', E', \boldsymbol{X}')$ be graphs. We say $X'$ is a *subgraph* of $X$, written as $X' \subseteq X$ if the following properties are satisfied

1. $V' \subseteq V$,
2. $E' \subseteq E$,
3. $\boldsymbol{x}'_{ij} = \boldsymbol{x}_{ij}$ for all $(i,j) \in E' \cup \Delta(V')$.

If $X' \subseteq X$ and $E' = E \cap V' \times V'$, then $X'$ is an *induced subgraph* of $X$. An induced subgraph $X'$ contains all edges $(i,j) \in E$ with $i,j \in V'$. Thus, the set $V'$ *induces* $X'$ in $X$. More generally, if $U$ is a subset of vertices from $V$, then $X[U]$ denotes the subgraph induced by $U$. In particular, we have $X' = X[V']$.

A *clique* of $X$ is a subset $C \subseteq V$ such that $X[C]$ is complete. A *maximal clique* of $X$ is a clique $C$ with

$$C \subseteq C' \;\Rightarrow\; C = C'.$$

Hence, a maximal clique is not properly contained in any other clique. A *maximum clique* is a clique with maximum cardinality of vertices. That is, $C$ is maximum clique of $X$ if, and only if,

$$|C| \geq |C'|$$

for all cliques $C'$ of $X$.

### 2.2.5 Graph Morphisms

Let $X$ and $Y$ be graphs. A *morphism* from $X$ to $Y$ is a mapping

$$\phi : V(X) \to V(Y), \quad i \mapsto i^{\phi}.$$

A morphism is structureless in the sense that neither relations nor attributes need to be preserved. A morphism $\phi : V(X) \to V(Y)$ induces a mapping on $I(X)$ in the following way: For $(i,j) \in I(X)$ we define

$$\left(i,j\right)^{\phi} = \left(i^{\phi}, j^{\phi}\right).$$

A *partial morphism* from $X$ to $Y$ is a morphism $\phi$ defined on a subset of $V(X)$. The *domain* of $\phi$ is the set

$$\mathcal{D}(\phi) = \left\{ i \in V(X) \,:\, \exists j \in V(Y) \text{ such that } i^{\phi} = j \right\}.$$

For a given item $(i,j)$ of $X$, we write $(i,j) \in \mathcal{D}(\phi)$ if $i \in \mathcal{D}(\phi)$ and $j \in \mathcal{D}(\phi)$. This slight abuse of notation simplifies technicalities considerably. Therefore, we

will make use of both notations $i \in \mathcal{D}(\phi)$ and $(i,j) \in \mathcal{D}(\phi)$. Similarly, the *range* of $\phi$ is the set

$$\mathcal{R}(\phi) = \left\{ j \in V(Y) \,:\, \exists i \in V(X) \text{ such that } j = i^{\phi} \right\}.$$

As for the domain of $\phi$, we say item $(i,j)$ of $Y$ is a member of $\mathcal{R}(\phi)$ if $i \in \mathcal{R}(\phi)$ and $j \in \mathcal{R}(\phi)$.

A *monomorphism* is an injective morphism. A monomorphism preserves vertex and non-vertex items.

A *homomorphism* from $X$ to $Y$ is a morphism $\phi : V(X) \to V(Y)$ such that

1. $\boldsymbol{x_i} = \boldsymbol{y_{i^\phi}}$
2. $\boldsymbol{i}^{\phi} \in E(Y)$ whenever $\boldsymbol{i} \in E(X)$

for all $\boldsymbol{i} \in I(X)$. A homomorphism preserves attributes and demands that edges of $X$ are mapped to edges of $Y$. Two vertices $i$ and $j$ of $X$ may be mapped to a single vertex of $Y$ if attributes are preserved and if $i$ and $j$ are not adjacent in $X$. We call $X$ and $Y$ *homomorphic* if there is a homomorphism from $X$ to $Y$.

An *isomorphism* from $X$ to $Y$ is a bijective morphism $\phi : V(X) \to V(Y)$ such that

1. $\boldsymbol{x_i} = \boldsymbol{y_{i^\phi}}$
2. $\boldsymbol{i} \in E(X)$ if, and only if, $\boldsymbol{i}^{\phi} \in E(Y)$

for all $\boldsymbol{i} \in I(X)$. We call $X$ and $Y$ *isomorphic*, writing $X \simeq Y$ if there is an isomorphism between $X$ and $Y$. By $\mathcal{I}_{X,Y}$, we denote the set of isomorphisms from $X$ to $Y$.

### 2.2.6 Special Classes of Attributed Graphs

Let $X = (V, E, \boldsymbol{X})$ be an arbitrary graph of the particular class of attributed graphs under consideration.

1. *Simple or Binary Graphs*
   Let $\mathcal{A} = \mathbb{B}$ and $\epsilon = 0$. Then any element $X$ of $\mathcal{G}_{\mathbb{B}}$ is a simple or binary graph. In contrast to conventional definitions of an adjacency matrix, the diagonal elements of $\boldsymbol{X}$ are all equal to the value 1.
2. *Normalized Graphs*
   Let $\mathcal{A} = \mathbb{U}$ and $\epsilon = 0$. Then any element of $\mathcal{G}_{\mathbb{U}}$ is a normalized graph. Vertices and edges of a normalized graph are colored with positive scalars from the unit interval $\mathbb{U}$. We assign the weight 0 to non-edges.
3. *Weighted Graphs*
   Let $\mathcal{A} = \underline{\mathbb{R}}$ and $\epsilon = -\infty$. Then any element of $\mathcal{G}_{\underline{\mathbb{R}}}$ is a weighted graph. A weighted graph assigns real valued weights to its vertices and edges. Non-edges are colored by the weight $\epsilon$.

### 2.2.7 Complex Graphs

Next, we introduce a new concept that will be useful to simplify theoretical considerations.

A *complex attributed graph* is a triple $X = (V, E, \boldsymbol{X})$ consisting of a finite set $V \neq \emptyset$ of vertices, a set $E \subseteq V \times V \setminus \Delta(E)$ of edges, and a (complex attributed) adjacency matrix $\boldsymbol{X} = (\boldsymbol{x}_{ij}) \in \mathcal{A}^{|V| \times |V|}$ such that

1. $\boldsymbol{X}^\top = \boldsymbol{X}$
2. $\boldsymbol{x}_{ij} = \epsilon \;\Rightarrow\; (i,j) \in V^2 \setminus E$

for all $i, j \in V$. A complex graph differs from the standard definition of an attributed graph only in the second condition. The null attribute $\epsilon$ is no longer confined to non-edges, but may also be used to color vertices. This implies that a complex graph may have edges incident with vertices colored with $\epsilon$.

We distinguish between imaginary and real parts of a complex graph. By

$$\mathfrak{R}[V] = \{i \in V \,:\, \boldsymbol{x}_{ii} \neq \epsilon\}$$

and

$$\mathfrak{I}[V] = V \setminus \mathfrak{R}[V]$$

we denote the subset of *real* and *imaginary vertices*, resp., of $X$. Similarly,

$$\mathfrak{R}[E] = E \cap \mathfrak{R}[V]^2$$

and

$$\mathfrak{I}[E] = E \setminus \mathfrak{R}[E]$$

are the subsets of *real* and *imaginary edges*, resp., of $X$. The *real part* $\mathfrak{R}[X]$ of a complex graph $X$ is the subgraph $X\big[\mathfrak{R}[V]\big]$ induced by $\mathfrak{R}[V]$. Thus, the real part of $X$ is an attributed graph. The *imaginary part* $\mathfrak{I}[X]$ of $X$ consists of imaginary vertices from $\mathfrak{I}[V]$ and edges incident to those vertices.

## 2.3 Structural Proximity

Proximity is an important concept for various applications in pattern recognition and related areas. This particularly holds for domains without a useful algebraic structure. For example, in the domain of attributed graphs, it is unclear how to model well-defined algebraic operations such as addition and scalar multiplication that are invariant under permutations of the vertices, and at the same time useful for defining statistical concepts like the mean or variance of a sample of graphs. But without an appropriate algebraic model, the rich arsenal of mathematical tools for data analysis in Euclidean spaces remains unavailable in the domain of

attributed graphs. For this reason, exploiting spatial relationships among attributed graphs using pairwise proximities is one key issue of concern in structural pattern recognition.

There are numerous ways to measure proximities between graphs. Here, we focus on *structural proximities* that measure common structural consistent and inconsistent parts. Structural proximity measures are not only widely applied in structural pattern recognition [60, 367]. There is also strong evidence that human cognitive models of comparison and analogy establish relational correspondences between structured objects [110, 122, 158, 192, 204, 236].

Since this chapter is intended to set the stage of the graph matching problem, we only consider proximities that are based on a single structural concept, namely the *maximum common induced subgraph*. As we will see in conjunction with the next section, our approach serves to stress the difference between the terms *structural proximity measure* and *graph matching problem*, which are widely used synonymously in the literature. A systematic and detailed overview of similarity measures for structures is given in Schädler's dissertation [311]. Downs & Willet [76] provide an extended review of similarity measures for chemical structures.

### 2.3.1 Proximity Measures

To meet the requirements of applications in structural pattern recognition, we have to model proximity relations between attributed graphs in a mathematical way. The usual approach measures proximities by a similarity or, alternatively, by a distance function. Care must be taken to adopt standard definitions of proximity measures, because structurally identical (isomorphic) graphs are in general mathematically unequal.

#### *Similarity*

A *(symmetric) similarity measure* is a function

$$s : \mathcal{G}_\mathcal{A} \times \mathcal{G}_\mathcal{A} \to [s_{\min}, s_{\max}]$$

such that

- $X \simeq Y \;\Rightarrow\; s(X,Y) = s_{\max}$
- $s(X,Y) = s(Y,X)$

for all $X, Y \in \mathcal{G}_\mathcal{A}$. The interval limits $s_{\min}$ and $s_{\max}$ denote minimal and maximal similarity. Both limits are real numbers from $\overline{R}$ with $s_{\min} < s_{\max}$. Common choices of $s_{\min}$ and $s_{\max}$ are

$$\begin{aligned} s_{\min} &= 0 \quad \text{and} \quad s_{\max} = 1, \\ s_{\min} &= 0 \quad \text{and} \quad s_{\max} = \infty, \\ s_{\min} &= -m \quad \text{and} \quad s_{\max} = m. \end{aligned}$$

It is noteworthy that some applications use asymmetric measures [270, 288, 298]. Moreover, there are strong arguments that human perception to similarity reveals significant and systematic asymmetries [220, 274, 361].

#### *Dissimilarity*

It is often convenient to measure the dissimilarity of attributed graphs by a distance measure. A *distance measure* is a function

$$d : \mathcal{G}_\mathcal{A} \times \mathcal{G}_\mathcal{A} \to \mathbb{R}_+$$

such that

1. $d(X, Y) \geq 0$
2. $d(X, Y) = d(Y, X)$
3. $X \simeq Y \Rightarrow d(X, Y) = 0$

for all $X, Y \in \mathcal{G}_\mathcal{A}$. A distance measure is a *metric* if the following properties for all $X, Y, Z \in \mathcal{G}_\mathcal{A}$ are also satisfied:

4. $d(X, Y) = 0 \Rightarrow X \simeq Y$
5. $d(X, Z) \leq d(X, Y) + d(Y, Z)$.

Intuitively, smaller distances between graphs mean larger similarities and vice versa.

#### *Transformations Between Similarities and Distances*

Similarity and dissimilarity are dual concepts such that there are various methods to transform similarity to distance measures and vice versa. For example, let $d$ be a distance measure. Then

$$s(X, Y) = \frac{1}{1 + d(X, Y)}$$

is a similarity measure. For further examples of transformations, we refer to [311, 343].

### 2.3.2 Structural Proximities

Simple but popular examples of structural proximity measures are centered around the maximum common induced subgraph (MCIS). In the following, we review the concept of MCIS and provide some examples of proximities based on that concept.

#### *The Maximum Common Induced Subgraph*

A *common induced subgraph* of attributed graphs $X$ and $Y$ is a graph $Z$ isomorphic to induced subgraphs of $X$ and $Y$. A *maximum common induced subgraph* of $X$

and $Y$ is a common induced subgraph of maximum order. Intuitively, the larger the order of $Z$, the more similar $X$ and $Y$ are considered. This similarity concept was apparently first applied in the 1970$s$ by Levi [228], Ambler et al. [10], and Barrow & Burstall [19]. Since then, it has been continuously used in various problems [280, 295, 312, 317, 388].

Initial efforts to ground the concept of MCIS on a firm mathematical footing were made by Zelinka [396] in 1975. For binary graphs $X$ and $Y$ of equal order $|X| = |Y| = n$, Zelinka showed that

$$d_Z(X,Y) = n - \big|Z\big|$$

is a distance metric, where $Z$ is a MCIS of $X$ and $Y$. In 1982, Kaden [197] extended Zelinka's result and showed for graphs $X$ and $Y$ of arbitrary order that

$$d_{K1}(X,Y) = \max\Big(\big|X\big|,\big|Y\big|\Big) - \big|Z\big|$$

and equivalently

$$d_{K2}(X,Y) = 1 - \frac{\big|Z\big|}{\max\Big(\big|X\big|,\big|Y\big|\Big)}$$

are distance metrics, where again $Z$ is a MCIS of $X$ and $Y$. Furthermore, in the same work, Kaden introduced a distance metric based on the *minimum common induced supergraph* $Z'$ of $X$ and $Y$

$$d_{K3}(X,Y) = \big|Z'\big| - \min\Big(\big|X\big|,\big|Y\big|\Big).$$

A *common induced supergraph* of $X$ and $Y$ is a graph $Z'$ such that $X'$ and $Y'$ are isomorphic to induced subgraphs of $Z'$. If $Z'$ has minimum cardinality of vertices, we call $Z'$ a *minimum common induced supergraph* (MCIS′) of $X$ and $Y$. Kaden showed that

$$d_{K1}(X,Y) = d_{K3}(X,Y).$$

Similarly, Bunke et al. [49] showed that a MCIS′ can be expressed in terms of a MCIS.[2]

In order to cope with the requirements of applications, the stringent structural conditions associated with the concept of MCIS have been relaxed by several researchers. In 2001, Fernandez & Valiente [89] relaxed the condition of induction and showed that

$$d_{FV}(X,Y) = \big|Z'\big| - \big|Z\big|$$

2. Obviously unaware of Kaden's work, Bunke & Shearer [50] rediscovered Kaden's second metric $d_{K2}$ in 1998. In addition, Bunke et al. [49] reintroduced the concept of the minimum common induced supergraph as a similarity measure.

is a metric, where $Z'$ is the MCIS′ and $Z$ the MCIS of $X$ and $Y$. In 1999, Schädler & Wysotzki [311, 314] generalized a variant of Kaden's second metric $d_{K2}$ and suggested a family of metrics by incorporating similarities for vertex and edge attributes. A few years later, in 2004, Hidović & Pelillo [142] and Torsello et al. [354] proposed metrics combining the MCIS and similarities of vertex attributes. The core of their metrics is closely related to the family of the Schädler & Wysotzki metrics.

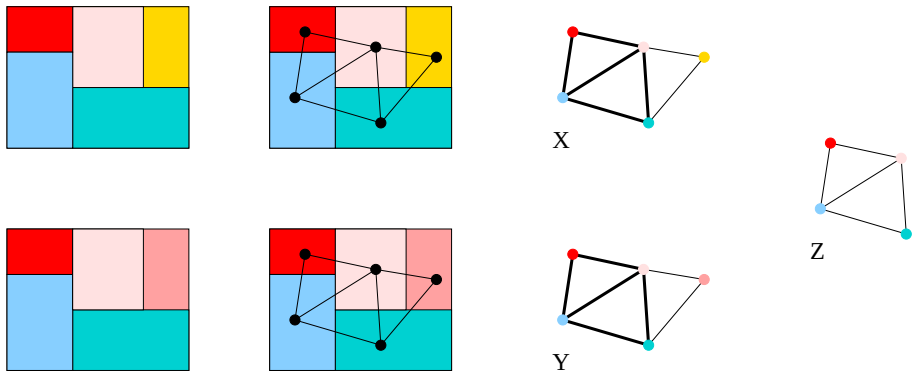



**Figure 2.6** Measuring similarity of image patterns using the concept of maximum common induced subgraph. Column (a) shows the raw data, column (b) identifies the basic constituents of the image pattern, column (c) shows the graph based representations $X$ and $Y$ of both images, and column (d) shows a common induced subgraph $Z$ of $X$ and $Y$. Vertices are colored with the color of their corresponding segments. Two vertices are connected by an edge if the corresponding segments are adjacent. The induced subgraphs of $X$ and $Y$ that are isomorphic to $Z$ are highlighted.

***Example 2.1***
Consider the problem of measuring the similarity of the two images patterns given in Figure 2.6(a). We transform this problem to that of measuring the similarity of two attributed graphs using the concept of the MCIS. Figure 2.6(d) shows that a maximum common induced subgraph $Z$ is of order $|Z| = 4$. Using $|X| = |Y| = 5$, Kaden's metric $d_{K2}$ gives

$$d_{K2} = 1 - \frac{4}{5} = 0.2.$$

### *Structural Similarity Measures Based on the MCIS*

Given a structural concept like the MCIS, different proximity measures can be defined. Several of those measures correspond to similarity coefficients originally defined on binary feature vectors. Table 2.1 summarizes seven graph-based similarity coefficients based on the MCIS.

Given the notation of Table 2.1, let $Z' = \big(X \cup Y\big)/Z$ denote the union of $X$ and

| Graph-based coefficients | | Feature-based coefficients | |
|---|---|---|---|
| Reference | $s(X, Y)$ | Reference | $s(\boldsymbol{x}, \boldsymbol{y})$ |
| Wallis et al. [377] | $\frac{\lvert Z\rvert}{\lvert X\rvert + \lvert Y\rvert - \lvert Z\rvert}$ | Jaccard/Tanimoto | $\frac{z}{x + y + z}$ |
| Kaden [197] | $\frac{\lvert Z\rvert}{\max\left(\lvert X\rvert, \lvert Y\rvert\right)}$ | Braun-Blanquet | $\frac{z}{\max(x, y)}$ |
| Johnson [193, 194] | $\frac{\lvert Z\rvert^2}{\lvert X\rvert\lvert Y\rvert}$ | Cosine | $\frac{z}{\sqrt{x \cdot y}}$ |
| Johnson [193, 194] | $\frac{2\lvert Z\rvert}{\lvert X\rvert + \lvert Y\rvert}$ | Dice | $\frac{2z}{x + y}$ |
| Ellis et al. [80] | $\frac{\lvert Z\rvert}{\min\left(\lvert X\rvert, \lvert Y\rvert\right)}$ | Asymmetric | $\frac{z}{\min(x, y)}$ |
| Ellis et al. [80] | $\frac{\lvert Z\rvert\lvert X\rvert + \lvert Z\rvert\lvert Y\rvert}{2\lvert X\rvert\lvert Y\rvert}$ | Kulczynski | $\frac{zx + zy}{2xy}$ |
| Ellis et al. [80] | $\frac{\lvert Z\rvert\lvert X\rvert + \lvert Z\rvert\lvert Y\rvert}{2\lvert X\rvert\lvert Y\rvert}$ | Sokal and Sneath | $\frac{z}{2x + 2y - z}$ |

**Table 2.1** Graph-based and corresponding feature-based similarity coefficients. The quantity $|Z|$ is the order of the MCIS of graphs $X$ and $Y$. The vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are binary vectors from $\mathbb{B}^n$. The quantity $z$ is the number of bits set in both vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, $x$ is the number of bits set in vector $\boldsymbol{x}$ and not in vector $\boldsymbol{y}$, and $y$ is the number of bits set in vector $\boldsymbol{y}$ and not in vector $\boldsymbol{x}$.

$Y$, where we identify the induced subgraphs $X' \simeq Z$ of $X$ and $Y' \simeq Z$ of $Y$ in $Z'$. We may regard $Z'$ as the graph obtained by gluing the disjoint graphs $X$ and $Y$ along their induced subgraphs $X'$ and $Y'$ via an isomorphism $\phi \in \mathcal{I}_{X',Y'}$. It can be shown that $Z'$ is a MCIS′ of $X$ and $Y$ [49]. Note that $Z'$ is not uniquely determined, even not up to isomorphism, since $Z'$ depends on $X'$, $Y'$, and the isomorphism $\phi \in \mathcal{I}_{X',Y'}$. But what is uniquely determined are the orders of $X$, $Y$, and $Z$ within $Z'$. This justifies our ambiguous but more intuitive notation $\big(X \cup Y\big)/Z$.

The relationship between graph-based and feature-based similarity coefficients becomes evident if we regard $\boldsymbol{x}$ and $\boldsymbol{y}$ as binary vectors of length $|Z'|$. We set a bit in $\boldsymbol{x}$ (and $\boldsymbol{y}$) if the corresponding vertex in $Z'$ belongs to $X$ (and $Y$). Bits that are set in both vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ correspond to vertices of $X'$ and $Y'$ via $\phi$ in $Z'$. Then by construction, $|X| + |Y| = x + y - z$ or equivalently, $|X| + |Y| - |Z| = x + y$. Finally, it is noteworthy that feature-based similarity coefficients can be extended to corresponding structural similarity coefficients based on other concepts of structural relationships including, for example, the family of metrics suggested by Schädler & Wysotzki [311, 314] and the metrics proposed by Torsello et al. [354].

As for feature-based similarity coefficients, optimal selection of graph-based

similarity coefficients as shown in Table 2.1 is problem dependent and may have a significant impact on the effectiveness of a pattern recognition system. A systematic comparison of structural similarity coefficients based on the same structural concept is widely unexplored. First investigations on that issue can be found in [296, 316, 354].

***Complexity***

All structural similarity measures shown in Table 2.1 require that the order of a MCIS of the given graphs be compared. Determining the order of a MCIS is an NP-complete combinatorial optimization problem. This is an immediate consequence of the NP-completeness of the induced subgraph isomorphism problem [104], which is a special case of the MCIS problem.[3] Hence, determining structural similarities based on the concept of MCIS is intractable.

Computational inefficiency holds for almost all structural measures [24] and is the starting point of extensive research on algorithms and heuristics to determine structural similarities. A well-known exception is the graph isomorphism problem.[4] It is still an open question whether the graph isomorphism problem lies in NP-complete or in P. We refer to Chapter 5 for a detailed discussion.

## 2.4 The Graph Matching Problem

The graph matching problem consists of determining a structural proximity. The proximity measure determines the constraints and the objective function to be optimized. As indicated in the previous section, a family of different proximity measures may rely on the same structural concept, for example the MCIS, whose determination is usually intractable. In order to gain more insight into the problem and to apply generic algorithms, we abstract from the trivial terms of a structural proximity measure and solely focus on its intractable part.

This section introduces a formal definition of graph matching problems and provides some basic problem formulations commonly used in structural pattern recognition. Most other graph matching problems are either variations or can be reduced to one of the basic forms.

A *graph matching problem* is a combinatorial optimization problem of the form

$$\begin{aligned} &\text{maximize} && f(\phi, X, Y) \\ &\text{subject to} && \phi \in \mathcal{M}, \end{aligned} \tag{2.1}$$

3. A formal definition of the induced subgraph isomorphism problem is provided in the next section.
4. A formal definition of the graph isomorphism problem is presented in the next section.

| Exact graph matching problems | | |
|---|---|---|
| | Graph matching problem | $\mathcal{M} \subseteq \mathcal{M}_{X,Y}$ |
| 1 | maximum common subgraph | partial monomorphisms |
| 2 | maximum common induced subgraph | partial isomorphisms |
| | (a) induced subgraph isomorphism ($\lvert X\rvert \leq \lvert Y\rvert$) | total isomorphisms |
| | (b) graph isomorphism ($\lvert X\rvert = \lvert Y\rvert$) | total isomorphisms |
| 3 | maximum common homomorphic subgraph | partial homomorphisms |
| | (a) subgraph homomorphism ($\lvert X\rvert \leq \lvert Y\rvert$) | total homomorphisms |

**Table 2.2** Selected examples of exact graph matching problems.

where the *feasible region* $\mathcal{M}$ is a subset of the set $\mathcal{M}_{X,Y}$ of partial morphisms from $X$ to $Y$. The function $f$ to be maximized is the *graph matching objective* or *matching objective*.

As already mentioned, several proximity measures can be derived from an optimal solution $\phi^* \in \mathcal{M}$ or the optimal value $f^* = f(\phi^*)$ of (2.1). In particular, $f^*$ is a similarity of $X$ and $Y$.

Let us consider some examples of the most common graph matching problems. To characterize their different forms, it is sometimes useful to take the perspective of *model-based pattern recognition*. From the point of view of model-based pattern recognition, we regard $X$ as a *data graph* and $Y$ as a *model graph*. In loose terms, the task of graph matching is to recognize to which extent the data graph fits to the model graph. Two kind of corruptions may occur when representing real-world data by attributed graphs:

- *Structural errors* refer to missing or additional vertices and edges caused by missing data, defective preprocessing, or flawed feature extraction.
- *Noise* refers to corrupted measurements of vertex and edge attributes.

In the following, we distinguish between *exact* and *inexact* graph matching problems as summarized in Table 2.2 and 2.3, respectively. The definitions given in both tables emanate from the graph matching problem (2.1) formulated as an optimization problem. Note that we present a more general definition of exact graph matching problems than usually found in the literature.

### 2.4.1 Exact Graph Matching Problems

Exact graph matching problems aim at maximizing exact matches of vertices, edges, and non-edges over the feasible region $\mathcal{M} \subseteq \mathcal{M}_{X,Y}$. Inconsistencies of attributes do not contribute to the matching objective $f$ of (2.1). From the perspective of model-based pattern recognition, exact graph matching problems are appropriate for coping with structural errors caused by missing vertices or edges.

Let $X$ and $Y$ be attributed graphs from $\mathcal{G}_\mathcal{A}$ with adjacency matrices $\boldsymbol{X} = (\boldsymbol{x}_{ij})$

and $\boldsymbol{Y} = (\boldsymbol{y}_{ij})$. Consider the matching objective

$$f\big(\phi, X, Y\big) = \sum_{\boldsymbol{i} \in D(\phi)} \kappa_{\boldsymbol{i}\boldsymbol{i}^\phi},$$

where $\kappa_{\boldsymbol{i}\boldsymbol{j}}$ denotes the compatibility value of item $\boldsymbol{i} \in I(X)$ and item $\boldsymbol{j} \in I(Y)$.[5] We incorporate the condition of exact matches into the compatibility values by defining

$$\kappa_{\boldsymbol{i}\boldsymbol{j}} = \begin{cases} \alpha_V & : \quad \boldsymbol{x}_{\boldsymbol{i}} = \boldsymbol{y}_{\boldsymbol{j}},\ \boldsymbol{i} \in V(X),\ \boldsymbol{j} \in V(Y) \\ \alpha_E & : \quad \boldsymbol{x}_{\boldsymbol{i}} = \boldsymbol{y}_{\boldsymbol{j}},\ \boldsymbol{i} \in E(X),\ \boldsymbol{j} \in E(Y) \\ \alpha_{\overline{E}} & : \quad \boldsymbol{i} \in \overline{E}(X),\ \boldsymbol{j} \in \overline{E}(Y) \\ 0 & : \quad \text{otherwise} \end{cases} \tag{2.2}$$

for all items $\boldsymbol{i}$ of $X$ and $\boldsymbol{j}$ of $Y$. The nonnegative constants $\alpha_V$, $\alpha_E$, and $\alpha_{\overline{E}}$ weight the contribution of exact matches between vertices, edges, and non-edges, respectively, to the matching objective $f$.

An exact graph matching problem is then defined by

$$\begin{aligned} &\text{maximize} && f\big(\phi, X, Y\big) = \sum_{\boldsymbol{i} \in D(\phi)} \kappa_{\boldsymbol{i}\boldsymbol{i}^\phi} \\ &\text{subject to} && \phi \in \mathcal{M} \end{aligned} \quad . \tag{2.3}$$

Depending on the properties of the feasible region $\mathcal{M}$, we obtain different types of exact graph matching problems as shown in Table 2.2. We obtain further types of exact graph matching problems by imposing other properties on $\mathcal{M}$.

In order to provide a graph-theoretic meaning for the examples of exact graph matching problems given in Table 2.2, we define the *weight* $\omega(X)$ of $X$ by

$$\omega(X) = \alpha_V \big|V(X)\big| + \alpha_E \big|E(X)\big| + \alpha_{\overline{E}} \big|\overline{E}(X)\big|.$$

Note that $\omega(X)$ depends on the weight parameters $\alpha_V$, $\alpha_E$, and $\alpha_{\overline{E}}$. For convenience of presentation, we do not make the dependence explicit. Given the particular form of the feasible region $\mathcal{M}$ as provided in Table 2.2, it is now straightforward to show that the following problems are special instances of the exact graph matching problem (2.3).

1. *Maximum Common Subgraph Problem*
   An attributed graph $Z \in \mathcal{G}_{\mathcal{A}}$ is a *common subgraph* of $X$ and $Y$ if there are subgraphs $X' \subseteq X$ and $Y' \subseteq Y$ such that $Z \simeq X'$ and $Z \simeq Y'$. The maximum common subgraph problem of $X$ and $Y$ is that of finding a common subgraph $Z$ with maximum weight $\omega(Z)$.
2. *Maximum Common Induced Subgraph Problem*
   The maximum common induced subgraph problem of $X$ and $Y$ lies in finding a

5. Recall that $(i, j) \in \mathcal{D}(\phi)$ if $i \in \mathcal{D}(\phi)$ and $j \in \mathcal{D}(\phi)$. Hence, the notation $\boldsymbol{i} \in \mathcal{D}(\phi)$ makes sense.

common induced subgraph $Z$ with maximum weight $\omega(Z)$. For $\alpha_V = 1$, $\alpha_E = 0$, and $\alpha_{\overline{E}} = 0$, we obtain the MCIS problem as defined in Section 2.3.2.

(a) *Induced Subgraph Isomorphism Problem*
The induced subgraph isomorphism problem is that of deciding whether one graph is isomorphic to an induced subgraph of the other graph.

(b) *Graph Isomorphism Problem*
The graph isomorphism problem consists of deciding whether two given graphs are isomorphic.

3. *Maximum Common Homomorphic Subgraph Problem*
The maximum common homomorphic subgraph problem of $X$ and $Y$ asks for a common homomorphic subgraph $Z$ of maximum weight $\omega(Z)$.

(a) *Subgraph Homomorphism Problem*
The subgraph homomorphism problem is that of deciding whether one graph is homomorphic to a subgraph of the other graph.

***Remark 2.2***
*For all types of common morphic subgraphs problems, the reference graphs of the feasible region $\mathcal{M}$ are $X$ and $Y$, and not the particular subgraphs between which we try to establish a feasible morphism.*

***Variations***

We obtain variations of the basic forms of exact graph matching problems if we demand additional properties like connectivity, maximization of a common substructure with respect to the cardinality of vertices or edges. As a reference problem we consider the maximum common subgraph problem.

1. *Vertex-Maximum Common Subgraph*
The vertex-maximum common subgraph problem asks for a common subgraph with maximum cardinality of vertices. Choose $\alpha_V = 1$, $\alpha_E = 0$, and $\alpha_{\overline{E}} = 0$.

2. *Edge-Maximum Common Subgraph*
The edge-maximum common subgraph problem asks for a common subgraph with maximum cardinality of edges. Choose, for example, $\alpha_V = 1$, $\alpha_E = 1$, and $\alpha_{\overline{E}} = 0$.

3. *Maximum Common Connected Subgraph*
The maximum common connected subgraph problem asks for a common connected subgraph with maximum weight. Here, we require that $\mathcal{M}$ is the set of monomorphisms between connected subgraphs of $X$ and $Y$.

### 2.4.2 Inexact Graph Matching Problems

The stringent constraints of exact matches are too rigid, in particular, when vertex and edge attributes are noisy. In these cases, it is useful to introduce some degree

| Inexact graph matching problems | | |
|---|---|---|
| | Graph matching problem | $\mathcal{M} \subseteq \mathcal{M}_{X,Y}$ |
| 1 | best $\mathcal{M}$-morphic graph matching | $\mathcal{M}$ |
| 2 | probabilistic graph matching ($_{V(Y)\cup\{\epsilon_V\}}$) | total morphisms |
| 2 | error correcting graph matching | total monomorphisms |

**Table 2.3** Examples of inexact graph matching problems.

of noise tolerance or inexactness into the formulation of graph matching problems. From the perspective of model-based pattern recognition, inexact graph matching problems are appropriate for handling problems in which data graphs are corrupted by both structural errors and noise. We examine three types of inexact graph matching problems summarized in Table 2.3.

Again, we consider attributed graphs $X$ and $Y$ from $\mathcal{G}_\mathcal{A}$ with adjacency matrices $\boldsymbol{X} = (\boldsymbol{x}_{ij})$ and $\boldsymbol{Y} = (\boldsymbol{y}_{ij})$.

### *Best $\mathcal{M}$-morphic Graph Matching Problems*

Assume that the set $\mathcal{A}$ of vertex and edge attributes is decomposed into disjoint sets $\mathcal{A}_V$ of vertex attributes and $\mathcal{A}_E$ of edges attributes. Recall that such a decomposition of $\mathcal{A}$ is possible and that $\epsilon \in \mathcal{A}_E$. Let

$$s_V : \mathcal{A}_V \times \mathcal{A}_V \to [0,1]$$
$$s_E : \mathcal{A}_E \times \mathcal{A}_E \to [0,1]$$

be similarity functions defined on the set of vertex and edges attributes. Consider the following compatibility values between items of $X$ and $Y$

$$\kappa_{\boldsymbol{ij}} = \begin{cases} s_V(\boldsymbol{x_i}, \boldsymbol{y_j}) & : \quad \boldsymbol{i} \in V(X),\ \boldsymbol{j} \in V(Y) \\ s_E(\boldsymbol{x_i}, \boldsymbol{y_j}) & : \quad \text{otherwise} \end{cases} \tag{2.4}$$

for all items $\boldsymbol{i}$ of $X$ and $\boldsymbol{j}$ of $Y$. The *best $\mathcal{M}$-morphic graph matching problem* is of the form

$$\begin{aligned} &\text{maximize} && f\big(\phi, X, Y\big) = \sum_{\boldsymbol{i} \in D(\phi)} \kappa_{\boldsymbol{i}\boldsymbol{i}^\phi} \\ &\text{subject to} && \phi \in \mathcal{M} \end{aligned} \tag{2.5}$$

with compatibility coefficients $\kappa_{\boldsymbol{ij}}$ as defined in (2.4). Typically, the feasible region $\mathcal{M}$ is the set of partial morphisms or partial monomorphisms from $X$ to $Y$.

### *Probabilistic Graph Matching Problem*

In probabilistic graph matching, a probability model is drawn to measure compatibility between items. The aim is to then find a morphism from $X$ to $Y$ that

maximizes a global maximum a posteriori probability.

To describe the probabilistic graph matching problem in formal terms, we first introduce a distinguished *null color* $\epsilon_V$ for vertices not contained in $\mathcal{A}$. Next, we extend the model $Y$ by including an isolated vertex with null color $\epsilon_V$.

The probabilistic graph matching problem is defined by

$$\begin{array}{ll} \text{maximize} & f\big(\phi, X, Y\big) = P\big(\phi | \boldsymbol{X}, \boldsymbol{Y}\big) \\ \text{subject to} & \phi \in \mathcal{M}_{X,Y}, \end{array} \tag{2.6}$$

where the matching objective $P\big(\phi|\boldsymbol{X}, \boldsymbol{Y}\big)$ is the a posteriori probability of $\phi$ given the *measurements* $\boldsymbol{X}$ and $\boldsymbol{Y}$.[6] Applying Bayes Theorem, we obtain

$$P\big(\phi|\boldsymbol{X}, \boldsymbol{Y}\big) = \frac{p(\boldsymbol{X}, \boldsymbol{Y}|\phi)P(\phi)}{p(\boldsymbol{X}, \boldsymbol{Y})}, \tag{2.7}$$

where $P(\phi)$ is the joint prior for $\phi$. The quantities $p(\boldsymbol{X}, \boldsymbol{Y}|\phi)$ and $p(\boldsymbol{X}, \boldsymbol{Y})$ are the conditional measurement density and the probability density functions, respectively, for the sets of measurements.

From a conceptual point of view, the critical constituents of (2.7) are appropriate models for the conditional measurement density $p(\boldsymbol{X}, \boldsymbol{Y}|\phi)$ and the prior $P(\phi)$. The quantity $p(\boldsymbol{X}, \boldsymbol{Y}|\phi)$ models the distribution of the known measurements given a match $\phi$. The prior $P(\phi)$ measures the overall consistency of a morphism. Note that we may ignore $p(\boldsymbol{X}, \boldsymbol{Y})$, since it does not affect maximization of (2.6).

### *Error Correcting Graph Matching Problem*

The error correcting graph matching problem lies in determining the *graph edit-distance* of $X$ and $Y$. The graph edit-distance is defined as the minimum cost over all sequences of basic edit operations that transform $X$ into $Y$. Following common use, the set of basic edit operations are *substitution*, *insertion*, and *deletion* of items. Different cost functions can be assigned to each edit operation.

From the perspective of model-based pattern recognition, edit operations are transformations of items in order to correct structural errors and noise in the attributes. Typically, edit operation for corruptions that are likely to occur have low cost. Similarly, we usually assign high cost to edit operations for unlikely corruptions.

The sequences of edit operations that make $X$ and $Y$ isomorphic can be identified with the partial monomorphisms $\phi : V(X) \to V(Y)$. Each partial monomorphism $\phi$ induces a bijection $\phi : \mathcal{D}(\phi) \mapsto \mathcal{R}(\phi)$ from the domain $\mathcal{D}(\phi)$ to the range $\mathcal{R}(\phi)$ of $\phi$. In terms of $\phi$, the edit operations have the following form

- *Substitution*: An item $\boldsymbol{i}$ from $\mathcal{D}(\phi)$ is *substituted* by item $\boldsymbol{i}^{\phi}$ from $\mathcal{R}(\phi)$.

6. In accordance with the terminology used in [386] we refer to $\boldsymbol{X}$ and $\boldsymbol{Y}$ as the sets of measurements. Note that [386] only considers unary measurements for vertices.

- *Deletion*: Items $\boldsymbol{i}$ of $\overline{\mathcal{D}}(\phi) = I(X) \setminus \mathcal{D}(\phi)$ are *deleted* from $X$.
- *Insertion*: Items $\boldsymbol{j}$ from $\overline{\mathcal{R}}(\phi) = I(Y) \setminus \mathcal{R}(\phi)$ are *inserted* into $Y$.

If $\boldsymbol{x_i} = \boldsymbol{y_{i^\phi}}$, the substitution $\boldsymbol{i}$ by $\boldsymbol{i}^\phi$ is called *identical substitution*. The cost of a partial monomorphism $\phi$ is then defined by

$$f\left(\phi, X, Y\right) = \sum_{\boldsymbol{i} \in \overline{\mathcal{D}}(\phi)} C_{del}(\boldsymbol{i}) + \sum_{\boldsymbol{j} \in \overline{\mathcal{R}}(\phi)} C_{ins}(\boldsymbol{j}) + \sum_{\boldsymbol{i} \in \mathcal{D}(\phi)} C_{sub}\left(\boldsymbol{i}, \boldsymbol{i}^\phi\right), \tag{2.8}$$

where $C_{del}(\boldsymbol{i})$ is the cost of deleting item $\boldsymbol{i}$ of $X$, $C_{ins}(\boldsymbol{i})$ is the cost of inserting an item $\boldsymbol{i}$ into $Y$, and $C_{sub}(\boldsymbol{i}, \boldsymbol{j})$ is the cost of substituting an item $\boldsymbol{i}$ from $X$ by an item $\boldsymbol{j}$ of $Y$. We assume that all costs are nonnegative.

The *minimum graph edit problem* is then of the form

$$\begin{array}{ll} \text{minimize} & f\left(\phi, X, Y\right) \\ \text{subject to} & \phi \in \mathcal{M} \end{array} \tag{2.9}$$

where $\mathcal{M}$ is the subset of all partial monomorphisms from $X$ to $Y$. The global minimum $E_*$ of (2.9) is the *graph edit distance* of $X$ and $Y$.

***Remark 2.3***
*The concept of graph edit distance generalizes the Levenshtein edit distance originally defined for strings [227]. In* 1977*, Selkow [322] extended the Levenshtein edit distance to trees. A few years later, Sanfeliu & Fu [309] and Eshera & Fu [83] proposed edit distances for graphs. The graph edit distance can be considered as an example of a proximity measure that incorporates both structurally consistent and inconsistent correspondences. The graph edit distance and related proximity measures have been also discussed in [48, 325, 337, 358, 359].*

### 2.4.3 Complexity

The maximum common subgraph problem is NP-complete [65]. The graph isomorphism problem is suspected to be a candidate for a problem that is neither in P nor NP-complete, provided that P≠NP [212]. The induced subgraph isomorphism problem is a well-known NP-complete problem [104]. The maximum common induced subgraph problem is at least of the same complexity, because it contains the induced subgraph isomorphism problem as a special case. The graph homomorphism problem generalizes the minimum graph coloring problem, which is well-known to be NP-complete [65]. The best $\mathcal{M}$-morphic and error correcting graph matching problems generalize the maximum common subgraph problem and are therefore NP-complete. The probabilistic graph matching problem is closely related to the satisfiability problem SAT, which is well-known to be NP-complete [104]. As a consequence, the graph matching problem in its most general form is, in the worst case, NP-complete.

## 2.5 Optimization

The particular formulation of the graph matching problem determines which algorithms we can use to solve the problem. Since Barrow & Popplestone [20] introduced graph-based representations in high-level computer vision, many algorithms and heuristics have been devised to solve a variety of graph matching problems.

Following Gold & Rangarajan [116], most graph matching algorithms can be divided into *search-based* and *optimization-based* methods. Search-based methods cast the graph matching problem to a search problem in a state space, which is then solved by a search algorithm. Optimization-based methods formulate the graph matching problem as a constrained nonlinear optimization problem in order to apply techniques from nonlinear mathematical programming.

The goal of this section is to illustrate the diversity of solution methods for particular instances of the graph matching problem. Therefore, this section reviews some common techniques for solving graph matching problems. In particular, we consider

1. Tree search
2. Neural networks
3. Relaxation labeling
4. Other optimization methods

Tree search belongs to the class of search-based methods. The other three methods are optimization-based methods. For an overview of more specialized methods see [60].

### 2.5.1 Tree Search

Tree search methods are based on the idea of iteratively expanding partial morphisms by adding a new pair of feasible vertices. They differ in the order partial morphisms are visited. In general, tree search methods provide optimal solutions, but require exponential time in the worst case.

The standard and apparently most widely used algorithm for graph and (induced) subgraph isomorphism problems is the one proposed by Ullman [362].[7] Ullman's algorithm uses a sequential backtracking tree search enhanced with a look ahead function to prune the search space. In its original version, Ullman's algorithm can be applied to simple graphs only. Tree search techniques based on backtracking, branch & bound algorithms, and the $A^*$ algorithm for more general exact and inexact graph matching problems have been addressed by [48, 61, 82, 254, 324, 358, 359].

7. The term *(induced) subgraphs* abbreviates *subgraphs and induced subgraphs.*

### 2.5.2 Neural Networks and Related Energy Minimizing Techniques

Neural networks have been applied to the best (mono)morphic graph matching problem. First connectionist algorithms for solving graph matching problems are attributed to Cooper [62], Kree & Zippelius [218], and von der Malsburg [374] in 1988, and Li [230], Mjolsness et al. [265] and Wysotzki [387] in 1989. Most pioneering solutions to the best monomorphic graph matching problem are based on the framework of Hopfield and Tank [149].

The general approach of Hopfield and Tank to solve combinatorial optimization problems maps the objective function of the optimization problem to an energy function of the Hopfield network. The constraints of the problem are included in the energy function as penalty terms. In the context of the best monomorphic graph matching problem, the constraints are so-called *two-way winner takes all* (WTA) *constraints*. One direction of WTA constraints requires that any vertex $i$ of graph $X$ is mapped to one vertex $j$ of graph $Y$ at most. The image $j$ of vertex $i$ can then be considered as the winner for $i$ among all competitors from $V(Y)$. The other direction of WTA constraints requires that each vertex $j$ of $Y$ is the preimage of one vertex $i$ of $X$ at most. Similarly, the preimage of $i$ is the winner for $j$ among all competitors from $V(X)$. Obviously, the two-way WTA constraints aim to enforce that the solutions of the Hopfield model correspond to *well-defined* and *injective* partial morphisms from $X$ to $Y$. Once the Hopfield model has been initialized, the network dynamic minimizes the energy function until it converges to a local minimum.

Application of early Hopfield networks to graph matching problems fail for the following reason [336]: often the network converges to local minima of the energy function that corresponds either to infeasible solutions or to feasible solutions that are far from being optimal. Since the penalty terms of the energy function do not necessarily enforce the two-way WTA constraints, they are also referred to as *soft constraints*.

To mitigate this problem, Peterson & Soderberg [285] adopted techniques from statistical physics to solve combinatorial optimization problems with neural networks. Simić [332] incorporated the method of deterministic annealing into the elastic net algorithm proposed by Durbin & Willshaw [78] for the best monomorphic graph matching problem. Deterministic annealing convexifies the energy function in order to avoid local minima. Suganthan et al. [345] uses Potts glass approximation for the best morphic graph matching problem. Potts glass enforces one-way WTA constraints via the softmax principle and can therefore be regarded as an implementation of *hard constraints*.

But even if we use Potts glass, the monomorphic graph matching problems require two-way WTA constraints. Therefore, the energy function must still include penalty terms to implement the other set of WTA constraints. A breakthrough to ensure feasibility was made by Aiyer et al. [6], who showed that a sufficiently large penalty term ensures validity of the solution. The problem is that too large penalty terms force the network to converge rapidly to a feasible, but poor solution.

This impaired behavior, however, can be compensated with deterministic annealing techniques. In [313], Schädler & Wysotzki derived a lower bound for the penalty term to ensure feasibility. In addition, they replaced the deterministic annealing scheme by an exterior penalty point method. Pelillo [279] transformed the maximum common induced subgraph problem to a spurious free version of the Motzkin Strauss Theorem, which is then solved using replicator equations from theoretical biology. Other types of connectionist approaches were also proposed for graph matching such as Kohonen networks [392].

### 2.5.3 Relaxation Labeling

According to Gold & Rangarajan [116], the earliest and most successful continuous optimization methods use some form of relaxation labeling. Within a graph matching framework, the basic labeling problem is to find a labeling that assigns vertices of the data graph $X$ to vertices of the model graph $Y$ without violating the constraints. Continuous relaxation algorithms for solving the labeling problem use label weights that measure the extent to which an assignment is accepted. Initially, each vertex $i$ from $X$ is associated with a weight vector — usually consisting of probabilities — that measures the likelihood of labeling $i$ with vertices from $Y$. The goal of relaxation techniques is to iteratively reduce inconsistencies of the initial labeling with respect to a coherence matching objective. After convergence, each vertex of $X$ is labeled with the vertex of $Y$ having the largest label weight. Thus, classical relaxation methods enforce one-way WTA constraints in such that feasible labelings correspond to morphisms from $X$ to $Y$.

One example of pioneering work for this approach was made by Fischler & Elschlager [93] in 1973. A relaxation labeling algorithm that gained enormous popularity because of its simplicity is the method proposed by Rosenfeld et al. [304] in 1976. Faugeras & Price [87] first exploited the Rosenfeld-Hummel-Zucker relaxation scheme for the graph matching domain. From the perspective of Hummel & Zucker [159] in 1983, analysis of relaxation labeling within the probabilistic framework is unsuccessful, because it requires various independence assumptions and, at best, leads to an approximate understanding of only one iteration of the process. As an alternative to probabilistic and heuristic approaches to relaxation labeling, Hummel & Zucker extended the optimization viewpoint of Faugeras & Berthod [86] and treated relaxation labeling from a more basic foundation. Despite the statement by Hummel & Zucker, research on probabilistic relaxation schemes continued. A breakthrough in the theoretical justification of probabilistic relaxation using the Bayesian framework is attributed to Christmas et al. [57] and Kittler & Hancock [209]. Wilson & Hancock [386] derived a statistical model for probabilistic relaxation within a purely structural domain. Since then, further techniques of probabilistic graph matching have been proposed by [90, 272, 353]. A more recent method by Luo & Hancock [243] employs the Expectation-Maximization algorithm on probabilistic graph matching problems.

### 2.5.4 Other Optimization Methods

Besides neural networks and continuous relaxation algorithms, a number of non-linear optimization techniques have been proposed. We distinguish between discrete and continuous optimization methods. Attempts to apply techniques to graph matching problems over discrete spaces include, for example, genetic search [67, 334, 344], simulated annealing [139], and tabu search [384]. Yet, continuous optimization methods are commonly preferred to their discrete counterpart, because their dynamical behavior and convergence properties are usually better understood. Therefore, the main focus is on continuous optimization methods.

The most powerful solutions have been devised for graph matching problems with two-way WTA constraints. The *graduated assignment algorithm*, independently developed by Gold & Rangarajan [116] and Ishii & Sato [163], sets the benchmark with respect to solution quality and speed. From the improved results obtained by using Potts glass, it was widely accepted that optimization schemes using hard constraints via softmax are superior than those using soft constraints via penalty terms [117, 164, 285, 345]. Consequently, Gold & Rangarajan and Ishii & Sato developed a technique termed *graduated assignment* or *soft assign* to enforce hard constraints in both directions. Soft assign operates as an interlocked softmax loop based on a mathematical result of Sinkhorn [335]. One problem of graduated assignment is that the performance of the algorithm declines significantly at mid and high levels of structural corruption [91]. To mitigate this problem, Lozano & Escolano [238] incorporated diffusion kernels into the energy function of the graduated assignment algorithm.

Recently, van Wyk et al. proposed a series of approximate least-squares graph matching algorithms [368, 369, 370, 371]. Their algorithms perform non-Bayesian graph matching without explicit calculation of compatibility values and enforce the two-way WTA constraints using projection methods.

Schellewald et al. [315] used semidefinite programming techniques for the maximum common subgraph problem of simple graphs. Their approach is based on convex relaxation so that no tuning parameters have to be determined.

Spectral methods to monomorphic graph matching problems for weighted graphs of equal order were apparently proposed for the first time by Umeyama [363] in 1988. More sophisticated techniques using eigendecomposition have been suggested by [52, 216, 244, 320, 323, 391]. The advantage of spectral methods is their low computational complexity but their sensitivity to noise is disadvantageous.

## 2.6 Conclusion

This chapter introduced basic concepts of structural pattern recognition primarily from the perspective of the graph matching problem. An appropriate graph-based representation of a given set of complex objects and the choice of proximity measure are of crucial importance for uncovering the hidden structure of the data.

For this reason, structural proximity measures are a preferred choice in many applications of structural pattern recognition. The major difficulty with almost all structural measures is that their computation is an NP-complete combinatorial optimization problem. Therefore, as for many combinatorial optimization problems, an appropriate formulation of the graph matching problem is of crucial importance to efficiently solve the intractable problem of determining a structural proximity measure. As a consequence, considerable effort has been directed at devising a variety of optimization algorithms for graph matching problems. In the next two chapters, we will present a unified view of most common graph matching problems and propose a generic optimization algorithm.

We conclude this chapter by addressing an important open problem in structural pattern recognition. A main focus of research was and still is devoted to the graph matching problem. By now, we have well established and powerful optimization techniques at our disposal. This issue enables research on other, more complex tasks in structural pattern recognition. A good optimization algorithm is important to practically execute data analysis in the domain of graphs. But it is conceptually useless to solve the primary problem in pattern recognition – to uncover the true but unknown structure of the data. To solve this key problem, we need a suitable model of our problem domain, which is an appropriate representation of the data and a carefully chosen proximity measure. Systematic investigations and empirical comparisons in this direction are underrepresented. Exceptional examples of systematic investigations on structural similarity measures can be found in Raymond & Willett [296], Torsello et al. [352], and Schenker et al. [316]. As a consequence, we have paved navigable streets and constructed fast cars, but have a poor map to help us arrive at our goal. This circumstance may give rise to the criticism that the specific graph matching algorithm, and not the problem domain itself, fits our model.

# 3 Graph Matching as Clique Search

This chapter presents a generic graph-theorectical view for different types of graph matching problems. We provide a necessary and sufficient condition so that graph matching problems are equivalent to a clique search in a derived *association graph*. To this end, we generalize classical clique problems to the maximum weight clique problem. To enable application of continuous optimization methods, we present an equivalent continuous characterization of the maximum weight clique problem.

## 3.1 Introduction

As shown in Section 2, the graph matching problem appears in diverse forms, each of which can be solved by a variety of optimization techniques. Specialization to specific matching problems has produced powerful algorithms. An example is the graduated assignment algorithm [116, 163], which is applicable only to the best monomorphic graph matching problem.

A generic view on the graph matching problem is attractive because it

- simplifies analysis of common characteristics,
- enables development of generic algorithms, and
- provides easier access to graph matching problems for the unskilled practioner.

Despite the advantages offered by a generic framework, only little effort has been directed towards this issue. The few existing approaches provide a common view of certain exact graph matching problems as a clique search problem in a derived *association graph* [55, 282].[1]

In this chapter, we generalize the association graph framework and show that graph matching constrained over *closed* feasible regions (domains) is equivalent to a clique search in an extended association graph. The closedness property of the feasible region is a sufficient and necessary condition that holds for most common graph matching problems. This result reveals a common misconception that a clique search is merely a special technique for solving certain types of graph matching problems. The difference between solutions based on association graph techniques

1. A formal definition of an association graph is given in Section 3.5.

and other optimization methods is that the former makes the equivalence between graph matching and clique search explicit, whereas the latter implicitly performs a clique search.

En route to achieving our central objective, we arrive at results that are interesting on their own. In particular, the most relevant are

1. a proof technique to simplify showing whether a new graph matching problem is equivalent to a clique problem in an association graph, and
2. an equivalent continuous formulation of the maximum weight clique problem.

Referring to the first result, former and current association graph techniques provide a complete new proof for each graph matching problem of interest in order to establish equivalence between graph matching and clique search. Using the proposed technique, it all boils down to showing that the feasible region under consideration is closed with respect to some property. The second result is useful, because it allows us to apply a rich arsenal of continuous optimization methods.

This chapter is organized as follows. Section 3.2 provides a generic descriptive characterization of the graph matching problem. In Section 3.3, we transform the descriptive characterization into an equivalent quadratic integer program. The integer program links the graph matching problem with clique search. Section 3.4 extends standard clique problems to the maximum weight clique problems. The key result of this chapter is presented in Section 3.5. We prove that graph matching is equivalent to clique search in an extended association graph, provided some closedness property is satisfied. Section 3.6 presents a continuous formulation of the maximum weight clique problem. Finally, Section 3.7 concludes with a summary of the main results and an outlook for further research.

We conclude this section with a summary of some important notations that will be used in this chapter.

***Notation 3.1***
Let $\boldsymbol{A} = (\boldsymbol{a}_{ij})$ be an $(n \times n)$-matrix with entries from a set $\mathcal{A}$. Then

- $\boldsymbol{D_A} = \operatorname{diag}(\boldsymbol{a}_{11}, \ldots, \boldsymbol{a}_{nn})$ denotes the *diagonal matrix* of $\boldsymbol{A}$,
- $\boldsymbol{O_A} = \boldsymbol{A} - \boldsymbol{D_A}$ denotes the *off-diagonal matrix* of $\boldsymbol{A}$,
- $\boldsymbol{d_A} = (\boldsymbol{a}_{11}, \ldots, \boldsymbol{a}_{nn})^\mathsf{T}$ denotes the *diagonal* of $\boldsymbol{A}$.

Let $X$ be a graph on $V$ and let $C \subseteq V$ be a clique. Then $\mathbf{1}_C = (c_1, \ldots, c_n)^\mathsf{T}$ denotes the *characteristic vector* of $C$ whose elements are of the form

$$c_i = \begin{cases} 1 & : \quad i \in C \\ 0 & : \quad \text{otherwise} \end{cases}$$

for all $i \in [1 : n]$.

## 3.2 The Basic Graph Matching Problem

This section provides a generic declarative formulation of the graph matching problem.

Let

$$\mathcal{G}[I] = \big\{(X, \boldsymbol{i}) \,:\, X \in \mathcal{G}, \boldsymbol{i} \in I(X)\big\}$$

be the set of pairs consisting of items together with their respective graphs. A *compatibility function* is a symmetric, nonnegative function of the form

$$\kappa : \mathcal{G}[I] \times \mathcal{G}[I] \to \mathbb{R}, \quad \big((X, \boldsymbol{i}), (Y, \boldsymbol{j})\big) \mapsto \kappa_{XY}(\boldsymbol{i}, \boldsymbol{j}) = \kappa\big((X, \boldsymbol{i}), (Y, \boldsymbol{j})\big).$$

We drop the subscripts of $\kappa_{XY}(\boldsymbol{i}, \boldsymbol{j})$ referring to the underlying graphs $X$ and $Y$ if the reference is clear, and write $\kappa_{\boldsymbol{ij}}$. A compatibility function measures the degree of compatibility, or consistency, between two items of given graphs. Since $\kappa$ allows us to consider the structural context of an item, a compatibility function extends the concept of similarity functions on vertex and edge attributes.

Consider the *basic graph matching problem* (BGMP)

$$\begin{aligned} &\text{maximize} && f(\phi) = \sum_{\boldsymbol{i} \in \mathcal{D}(\phi)} \kappa_{\boldsymbol{i}\boldsymbol{i}^\phi} \\ &\text{subject to} && \phi \in \mathcal{M}, \end{aligned} \tag{3.1}$$

where the feasible region $\mathcal{M}$ is a subset of the set $\mathcal{M}_{X,Y}$ of partial morphisms from $X$ to $Y$.

***Remark 3.2***
*The matching objective in formulation (3.1) of the BGMP is suitable for both undirected and directed attributed graphs. Consider a non-vertex item $(i, j)$ of $X$. Assume that $\phi \in \mathcal{M}_{X,Y}$ is a feasible morphism with $i^\phi = k$ and $j^\phi = l$. The mapping $\phi$ on $\{i, j\}$ determines a mapping on both directions $(i, j)$ and $(j, i)$ in the obvious manner. Since $\phi$ is feasible, the mappings $(i, j)^\phi = (k, l)$ and $(j, i)^\phi = (l, k)$ are feasible. Therefore, both compatibility values $\kappa_{ij,kl}$ and $\kappa_{ji,lk}$ contribute to the matching objective. If the graphs under consideration are undirected, then the matching objective counts each edge twice.*

Formulation (3.1) is sufficiently general in capturing a broad range of common graph matching problems. To underpin this, we cast the formulations of Section 2.4 to a BGMP. All exact graph matching problems and the best $\mathcal{M}$-morphic graph matching problem have been introduced in Section 2.4 in the form of a BGMP. So it is left to show that probabilisitic and error correcting graph matching are BGMPs.

***Probabilistic Graph Matching Problems***

Probabilisitic graph matching problems are usually solved by a Bayesian inference scheme that does not require explicit calculation of compatibility values in advance. Conceptually, there are compatibility values and a probabilistic graph matching problem turns out to be a BGMP over the set of all total morphisms, where we extend the graph $Y$ by an isolated vertex with null attribute $\epsilon_V$.

***Error Correcting Graph Matching Problems***

To transform the error correcting graph matching problem (2.9) to an equivalent BGMP, we modify the set of attributes $\mathcal{A}$ and the graphs $X$ and $Y$.

We expand $\mathcal{A}$ by including the distinguished symbols $\mathtt{i}$ and $\mathtt{d}$. Let $\mathcal{A}' = \mathcal{A} \cup \{\mathtt{i}, \mathtt{d}\}$ denote the expanded set of attributes. Items with attribute $\mathtt{i}$ and $\mathtt{d}$ are called *insert-items* and *delete-items*, respectively.

Next, we assume that $X$ and $Y$ are of order $|X| = n$ and $|Y| = m$. We expand $X$ by adding $m$ *insert-vertices* with attribute $\mathtt{i}$. Similarly, we enhance $Y$ by adding $n$ *delete-vertices* with attribute $\mathtt{d}$. We denote the expanded versions of $X$ and $Y$ by $X'$ and $Y'$, respectively. Each insert-vertex of $X'$ is connected to all other $n+m-1$ vertices by insert-edges with attribute $\mathtt{i}$. Similarly, each delete-vertex of $Y'$ is connected to all other $n+m-1$ vertices by delete-edges with attribute $\mathtt{d}$.

To define an appropriate compatibility function $\kappa$ for the minimum graph edit problem, we first introduce some auxiliary notations. By $V(X, \boldsymbol{a})$ we denote the subset of all vertices of $X$ that have attribute $\boldsymbol{a}$. Furthermore, let $C'_{sub}$ be a nonnegative real-valued function on $I(X') \times I(Y')$ such that

$$C'_{sub}(\boldsymbol{i}, \boldsymbol{j}) = \begin{cases} C_{sub}(\boldsymbol{i}, \boldsymbol{j}) & : \quad \boldsymbol{i} \in I(X),\, \boldsymbol{j} \in I(Y) \\ 0 & : \quad \text{otherwise} \end{cases} .$$

Now we consider the following compatibility values of $X'$ and $Y'$

$$\kappa_{\boldsymbol{ij}} = \begin{cases} -C_{del}(\boldsymbol{i}) & : \quad \boldsymbol{i} \in V(X),\, \boldsymbol{j} \in V(Y', \mathtt{d}) \\ -C_{ins}(\boldsymbol{j}) & : \quad \boldsymbol{i} \in V(X', \mathtt{i}),\, \boldsymbol{j} \in V(Y) \\ -C'_{sub}(\boldsymbol{i}, \boldsymbol{j}) & : \quad \text{otherwise} \end{cases} \tag{3.2}$$

for all items $\boldsymbol{i}$ of $X'$ and $\boldsymbol{j}$ of $Y'$. Then the BGMP over the set of all total monomorphisms from $X'$ to $Y'$ is equivalent to the error correcting graph matching problem (2.9). Note that we consider negative costs as compatibility values to turn the minimization problem (2.9) into a maximization problem as in (3.1).

The following remark is important for practical issues.

***Remark 3.3***
*Suppose that $C_{del}(\boldsymbol{i}) = c_{del}$ and $C_{ins}(\boldsymbol{j}) = c_{ins}$ are constant functions on $V(X)$ and $V(Y)$, respectively. Then it is sufficient to expand $X$ by only one insert-vertex and $Y$ by only one delete-vertex. This is possible, because*

- $\kappa_{\boldsymbol{ij}} = \kappa_{\boldsymbol{i'j}}$
- $\boldsymbol{i}$ *and* $\boldsymbol{i'}$ *must not be mapped to the same vertex of* $Y'$

*for all distinct insert-vertices* $\boldsymbol{i}$ *and* $\boldsymbol{i'}$ *from* $X'$*. Hence, we may safely contract all insert-vertices to exactly one insert-vertex. A similar statement holds for delete-vertices.*

*Aligning both graphs to the same order* $n+m$ *serves technical simplicity to avoid tedious case distinctions between the monomorphic and non-monomorphic part of a morphism between the expanded graphs.*

## 3.3 A Quadratic Integer Program for Graph Matching

In this section, we cast the BGMP (3.1) to an associated constrained quadratic integer program. The quadratic program can be viewed as a basis for diverse optimization techniques such as soft assign [116], semidefinite programming [315], or eigendecomposition [363]. In addition, it links graph matching to a clique search.

Let $X$ and $Y$ be attributed graphs from $\mathcal{G}_{\mathcal{A}}$ of order $|X| = n$ and $|Y| = m$. Assume that we are given a compatibility function $\kappa$ defined on $\mathcal{G}[I] \times \mathcal{G}[I]$. For any item $(i,j)$ of $X$, the compatibility function $\kappa$ induces a weighted $(m \times m)$-matrix $\kappa_{ij}\boldsymbol{Y}$ of the form

$$\kappa_{ij}\boldsymbol{Y} = \begin{pmatrix} \kappa_{ij,11} & \cdots & \kappa_{ij,1m} \\ \vdots & \vdots & \vdots \\ \kappa_{ij,m1} & \cdots & \kappa_{ij,mm} \end{pmatrix}.$$

The matrix $\kappa_{ij}\boldsymbol{Y}$ is the *compatibility matrix* of graph $Y$ and item $\boldsymbol{i} \in I(X)$. Using compatibility matrices, we can extend the classical definition of a Kronecker product of matrices (see [136]). The *Kronecker* $\kappa$*-product* of $\boldsymbol{X}$ and $\boldsymbol{Y}$ is a real-valued $(n \cdot m \times n \cdot m)$-matrix of the form

$$\boldsymbol{X} \otimes \boldsymbol{Y} = \begin{pmatrix} \kappa_{11}\boldsymbol{Y} & \kappa_{12}\boldsymbol{Y} & \cdots & \kappa_{1n}\boldsymbol{Y} \\ \kappa_{21}\boldsymbol{Y} & \kappa_{22}\boldsymbol{Y} & \cdots & \kappa_{2n}\boldsymbol{Y} \\ \vdots & \vdots & \vdots & \vdots \\ \kappa_{n1}\boldsymbol{Y} & \kappa_{n2}\boldsymbol{Y} & \cdots & \kappa_{nn}\boldsymbol{Y} \end{pmatrix}.$$

To rewrite the BGMP in terms of the Kronecker $\kappa$-product, we need a matrix representation for the partial morphisms from $\mathcal{M}_{X,Y}$. The matrix representation of $\phi \in \mathcal{M}_{X,Y}$ is a matrix $\boldsymbol{M} = (m_{ij})$ of the form

$$m_{ij} = \begin{cases} 1 & : \text{ if } i^{\phi} = j \\ 0 & : \text{ otherwise} \end{cases}.$$

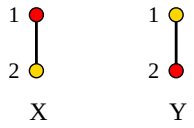



**Figure 3.1** Shown are two attributed graphs $X$ and $Y$ of order 2. Different colors of vertices correspond to different attributes. The edges of $X$ and $Y$ are labeled with the same attribute.

In general terms, a *match matrix* is a matrix $\boldsymbol{M} = (m_{ij}) \in \mathbb{B}^{n \times m}$ with

$$\sum_{j=1}^{m} m_{ij} \leq 1$$

for all $i \in [1\!:\!n]$. A match matrix has, at most, one entry with value 1 in each row. It is easy to see that the set $\mathcal{M}^{n \times m}$ of $(n \times m)$-match matrices are in one-to-one correspondence with the set set of partial morphisms $\mathcal{M}_{X,Y}$.

Given a compatibility function $\kappa$, the BGMP is equivalent to the following constrained quadratic integer program

$$\begin{array}{ll} \text{maximize} & f(\boldsymbol{M}) = \operatorname{vec}(\boldsymbol{M})^{\mathsf{T}} \boldsymbol{K} \operatorname{vec}(\boldsymbol{M}) \\ \text{subject to} & \boldsymbol{M} \in \mathcal{M}, \end{array} \tag{3.3}$$

where $\mathcal{M}$ is a subset of $\mathcal{M}^{n \times m}$, $\boldsymbol{K} = \boldsymbol{X} \otimes \boldsymbol{Y}$ is the Kronecker $\kappa$-product of $X$ and $Y$, and $\operatorname{vec}(\boldsymbol{M})$ is the vector obtained by stacking the rows of matrix $\boldsymbol{M}$ to a column vector

$$\operatorname{vec}(\boldsymbol{M}) = (m_{11}, \ldots, m_{1m}, \ldots, m_{n1}, \ldots, m_{nm})^{\mathsf{T}}.$$

Problem (3.3) is the desired *quadratic integer program of the basic graph matching problem* (QIP) .

***Example 3.4***
Consider the attributed graphs $X$ and $Y$ shown in Figure 3.1. Assume that our goal is to solve the graph isomorphism problem. As compatibility values, we choose

$$\kappa_{\boldsymbol{ij}} = \begin{cases} 1 & : \quad \boldsymbol{x_i} = \boldsymbol{y_j} \text{ and } |\{i_1, i_2\}| = |\{j_1, j_2\}| \\ 0 & : \quad \text{otherwise} \end{cases}$$

for all items $\boldsymbol{i} = (i_1, i_2)$ of $X$ and $\boldsymbol{j} = (j_1, j_2)$ of $Y$.[2] The chosen compatibility values correspond to those given in (2.2) by setting $\alpha_V = \alpha_E = \alpha_{\overline{E}} = 1$. Thus, arbitrary graphs $X'$ and $Y'$ of order $n$ are isomorphic, if the optimal solution of the BGMP has value $n^2$. This can be seen as follows: An isomorphism consistently maps $n$ vertex items and

2. The formula for $\kappa_{\boldsymbol{ij}}$ is based on the following trick: for a vertex item $\boldsymbol{i} = (i_1, i_2)$ the set $\{i_1, i_2\}$ consists of one element, because $i_1 = i_2$. For non-vertex items $\boldsymbol{i} = (i_1, i_2)$, the set $\{i_1, i_2\}$ consists of exactly two elements. Hence, $\kappa_{\boldsymbol{ij}} = 1$ if, and only if, items $\boldsymbol{i} \in I(X)$ and $\boldsymbol{j} \in I(Y)$ are both either vertex or non-vertex items with the same attribute.

$n(n-1)$ non-vertex items. Since $\kappa$ counts each consistent mapping of an item once, we obtain $n+n(n-1)=n^2$. Therefore, in our example, we expect that the matching objective of the QIP has maximum value 4.

The $\kappa$-Kronecker product $\boldsymbol{X} \otimes \boldsymbol{Y}$ is of the form

$$\boldsymbol{X} \otimes \boldsymbol{Y} = \left(\begin{array}{cc|cc} \kappa_{11,11} & \kappa_{11,12} & \kappa_{12,11} & \kappa_{12,12} \\ \kappa_{11,21} & \kappa_{11,22} & \kappa_{12,21} & \kappa_{12,22} \\ \hline \kappa_{21,11} & \kappa_{21,12} & \kappa_{22,11} & \kappa_{22,12} \\ \kappa_{21,21} & \kappa_{21,22} & \kappa_{22,21} & \kappa_{22,22} \end{array}\right) = \left(\begin{array}{cc|cc} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{array}\right)$$

where both matrices are partitioned into blocks of compatibility matrices of $Y$ with items from $X$. For example, the upper left submatrix $\kappa_{11}\boldsymbol{Y}$ of $\boldsymbol{X} \otimes \boldsymbol{Y}$ shows that vertex 1 of $X$ is only compatible to vertex 2 of $Y$.

For the graph isomorphism problem, the set of feasible match matrices are all $(2 \times 2)$-permutation matrices

$$\boldsymbol{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{P} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Stacking the rows of $\boldsymbol{I}$ and $\boldsymbol{P}$ yields

$$\operatorname{vec}(\boldsymbol{I}) = (1,0,0,1)^\mathsf{T} \quad \text{and} \quad \operatorname{vec}(\boldsymbol{P}) = (0,1,1,0)^\mathsf{T}.$$

Evaluating the matching objective $f$ of the QIP for $\boldsymbol{I}$ and $\boldsymbol{P}$ gives

$$f(\boldsymbol{I}) = \operatorname{vec}(\boldsymbol{I})^\mathsf{T} \boldsymbol{X} \otimes \boldsymbol{Y} \operatorname{vec}(\boldsymbol{I}) = 2$$

and

$$f(\boldsymbol{P}) = \operatorname{vec}(\boldsymbol{P})^\mathsf{T} \boldsymbol{X} \otimes \boldsymbol{Y} \operatorname{vec}(\boldsymbol{P}) = 4.$$

This shows that $\boldsymbol{P}$ is the optimal solution of the QIP. Since $f(\boldsymbol{P}) = 4$, both graphs $X$ and $Y$ are — as expected — isomorphic. The isomorphism represented by $\boldsymbol{P}$ gives the assignments $1 \mapsto 2$ and $2 \mapsto 1$.

## 3.4 The Maximum Weighted Clique Problem

The *maximum clique problem* (MCP) is a classical problem in combinatorial optimization. It was one of the first problems shown to be NP-complete [200]. An important generalization of the MCP is the *maximum vertex-weight clique problem* (MVCP). Given a weighted graph, the MVCP asks for a clique with the largest total weight over its vertices. Obviously, the MCP is a special case of the MVCP when all vertices are labeled with identical positive weights. Therefore, the MVCP is at least of the same complexity as the MCP. Both the MCP and the MVCP have many applications such as information retrieval [16], stereo vision correspondence [151], and shape recognition [284]. Further applications can be found in [36, 276].

A largely unknown and barely investigated extension of the classical MCP and MVCP is the *maximum weight b-clique problem* (MWCP$_b$). Given a complete weighted graph $X$, the MWCP$_b$ asks for a clique $C$ with, at most, $b$ vertices and the largest total weight over all vertices and edges of $X[C]$. The MWCP$_b$ is NP-complete, since the classical MCP reduces polynomially to it [161, 248].

Applications of the MWCP$_b$ include, for example, location theory [81, 221, 293, 339] and molecular biology [160].

In this section, we suggest a generalization of the MCP and the MVCP that is closely related to the MWCP$_b$, but more appropriate in representing graph matching problems. We obtain the *maximum weight clique problem* (MWCP) from the MWCP$_b$ by abandoning completeness of the underlying graph and the limiting threshold $b$. Though the formulation of the MWCP is a straightforward extension of the classical MCP in the same line with the MVCP, neither its mathematical formalization nor its theoretical analysis apparantly have been considered in the respective literature.

We will show that from the perspective of complexity theory, the MWCP is uninteresting, because it can be polynomially converted to the MVCP. Hence, for computer theorists and mathematicans there is no reason to consider the MWCP. But from a practical point of view, converting the MWCP to a MVCP is computationally unacceptable, particularly for graph matching problems. Nevertheless, potential applications of the MWCP are either unknown in the community of combinatorial otimization and graph theory, or did not lead to a mathematical formalization as in [31, 314].

Let $X = (V, E, \boldsymbol{X})$ be a weighted graph from $\mathcal{G}_{\underline{\mathbb{R}}}$. In the following, we assume that the void symbol $\epsilon$ represents $-\infty$ with $\epsilon \cdot 0 = 0$. The *weight* $\omega(C)$ of a clique $C$ of $X$ is defined by

$$\omega(C) = \frac{1}{2}\Big(\mathbf{1}_C^\top \boldsymbol{O}_{\boldsymbol{X}} \mathbf{1}_C\Big) + \mathbf{1}_C^\top \boldsymbol{d}_{\boldsymbol{X}}, \tag{3.4}$$

where $\boldsymbol{O}_{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{D}_{\boldsymbol{X}}$ is the off-diagonal matrix of $\boldsymbol{X}$, $\boldsymbol{d}_{\boldsymbol{X}}$ is the diagonal of $\boldsymbol{X}$, and $\mathbf{1}_C$ is the characteristic vector of clique $C$. The next result shows that definition (3.4) makes sense and is a well defined scalar. That is, $\omega(C)$ has no undefined term $a\epsilon$ with $a \neq 0$.

***Proposition 3.1***
*Let $C$ be a clique of a weighted graph $X$. Then $\omega(C) \in \mathbb{R}$.*

***Proof*** Let $\boldsymbol{O} = \boldsymbol{O}_{\boldsymbol{X}}$ be the off-diagonal matrix of $\boldsymbol{X}$, and let $\boldsymbol{x} = \mathbf{1}_C$ denote the characteristic vector of clique C. By definition of a weighted graph, vertices are associated with real valued attributes. Hence, from $\boldsymbol{d}_{\boldsymbol{X}} \in \mathbb{R}^n$ follows $\boldsymbol{x}^\top \boldsymbol{d}_{\boldsymbol{X}} \in \mathbb{R}$.

Now suppose that $\boldsymbol{x}^\top \boldsymbol{O} \boldsymbol{x} \notin \mathbb{R}$. Then there are indices $i, j \in [1:n]$ such that $o_{ij} x_i x_j \notin \mathbb{R}$. Thus, we have $o_{ij} = \epsilon$ and $x_i = x_j = 1$. From $x_i = x_j = 1$, it follows that $i$ and $j$ are members of $C$. Since $o_{ij} = \epsilon$ implies that $(i, j)$ is a non-edge, we have the contradiction that vertices $i$ and $j$ are not connected in $X[C]$. This implies $\boldsymbol{x}^\top \boldsymbol{O} \boldsymbol{x} \in \mathbb{R}$. Combining the results of both parts yields the assertion. ■

The *maximum weight clique problem* is an integer quadratic problem of the form

$$\begin{aligned} &\text{maximize} && f(\mathbf{1}_C) = \frac{1}{2}\left(\mathbf{1}_C^\top \boldsymbol{O}_{\boldsymbol{X}} \mathbf{1}_C\right) + \mathbf{1}_C^\top \boldsymbol{d}_{\boldsymbol{X}} \\ &\text{subject to} && C \in \mathcal{C}_X, \end{aligned} \tag{3.5}$$

where $\mathcal{C}_X$ is the set of all cliques of $X$.

It is useful to distinguish between global and local maxima of $f$.[3] The local maxima of $f$ correspond to the maximal weight cliques of $X$. A *maximal weight clique* of $X$ is a clique $C \in \mathcal{C}_X$ such that

$$C \subseteq C' \;\Rightarrow\; \omega(C) \geq \omega(C')$$

for all cliques $C'$ of $X$. It is impossible to enlarge a maximal weight clique $C$ to a clique with higher weight. If all vertices and edges of $X$ are associated with positive weights, a maximal weight clique is not a proper subset of another clique.

The global maxima of $f$ correspond to the maximum weight cliques of $X$. A *maximum weight clique* of $X$ is a clique $C \in \mathcal{C}_X$ with maximum total weight over its vertices and edges. Note that, because of the factor $1/2$ in (3.4), the weight of each edge of $X[C]$ contributes only once to $\omega(C)$. Obviously, a maximum weight clique is a maximal weight clique, whereas the converse does not generally hold.

The next examples show some special cases of the MWCP, including the standard MCP and MVCP.

***Example 3.5***

We assume that $X = (V, E, \boldsymbol{X})$ is a normalized graph from $\mathcal{G}_{\mathbb{U}}$. Vertices and edges of a normalized graph are associated with positive weights from $]0, 1]$. We may set $\epsilon = 0$ for the void attribute.

1. *Maximum Clique Problem* (MCP)
   The MCP calls for a clique with maximum cardinality of vertices. Hence, we can neglect vertex and edge weights and assume that $X$ is a binary graph. Then the MCP is a special instance of a MWCP.

2. *Maximum Vertex Weight Clique Problem* (MVCP)
   The MVCP lies in finding a clique with maximum total weight over the vertices only. Thus, we may assume that all edges of $X$ have identical weight $x_E$. If
   $$x_E < \frac{\min_{i \in V} x_{ii}}{|X|^2},$$
   then the MVCP is a special instance of the MWCP. The upper bound of $x_E$ ensures that edges do not outweigh any vertex weight.

3. *Maximum Edge Weight Clique Problem* (MECP)
   The MECP asks for a clique with maximum total weight over the edges only. We may assume that all vertices of $X$ have identical weight $x_V$. If
   $$x_V < \frac{\min_{(i,j) \in E} x_{ij}}{|X|},$$

3. Note that the term *local maximum* of an integer problem depends on how we define a local neighborhood of some point.

then the MECP is a special instance of the MWCP. The upper bound of $x_V$ ensures that vertices do not outweigh edge weights.

The following remark shows that the MWCP and MECP polynomially reduce to the classical MVCP.

***Remark 3.6***
*The MWCP and MECP can be easily converted to a MVCP. Just introduce for each weighted edge a new vertex with the corresponding weight and connect that vertex to all (old and new) vertices that may be in one clique with that edge.*

Conversion of the MWCP to a MVCP expands the size $n$ of the original graph by an order of magnitude $O(n^2)$. The quadratic order expansion is unacceptable in a practical setting. This holds in particular for graph matching problems.

We conclude this section with some notations for later use.

***Notation 3.7***
Let $X$ be a weighted graph.

- $\mathcal{C}_X$ denotes the set of all cliques of $X$.
- $\mathcal{C}_X^{\times}$ denotes the set of all maximal weight cliques of $X$.
- $\mathcal{C}_X^{*}$ denotes the set of all maximum weight cliques of $X$.

## 3.5 A Clique Formulation of the Graph Matching Problem

Graph matching as a clique search has a long tradition since the pioneering work of Ambler et al. [10]. They transformed the maximum common subgraph problem[4] to the problem of finding a maximum clique in an *association graph*. What makes their framework so useful is that they reduce the matching problem to an important graph-theoretic problem, for which a solid theory and powerful algorithms have been devised. Since then, this framework has been applied to several graph matching problems in computer vision and chemoinformatics [19, 151, 165, 208, 284, 295, 372, 389].

To generalize association graph techniques, Chen & Yun [55] compiled results from [19, 54, 199], showing that the maximum common (induced) subgraph problem and its derivations can be casted to a MCP. Pelillo [282] extended this collection by transforming the problem of matching free trees[5] to the MVCP so that connectivity is preserved. Bunke [46] showed that for special cost functions, the error correcting graph matching problem and the maximum common subgraph are equivalent. As a consequence, special graph edit distances can be computed via clique search in an

4. The classical maximum common subgraph problem asks for a common subgraph with maximum cardinality of vertices. Hence, we may choose $\alpha_V = 1$, $\alpha_E = 0$, $\alpha_{\overline{E}} = 0$.
5. A free tree is a directed acyclic graph without a *root*.

association graph. Schädler & Wysotzki [314] mapped the best monomorphic graph matching problem to a MWCP without presenting a sound theoretical justification of their transformation.

In this section, we provide the key result of this chapter, an equivalence relationship between graph matching and clique search, as long as some necessary and sufficient condition is satisfied. This condition demands that the feasible region is closed with respect to some local property.

### 3.5.1 Closed Sets of $\mathfrak{p}$-Morphisms

The formulation of the BGMP given in (3.1) is too general to admit an equivalent maximum weight clique formulation. The reason is that the feasible region $\mathcal{M}$ is not *closed* in some sense that will now be specified in detail.

The set of partial morphisms $\mathcal{M}_{X,Y}$ from $X$ to $Y$ determines the binary relation

$$\mathcal{R}_{X,Y} = \big\{(\boldsymbol{i},\boldsymbol{j}) \in I(X) \times I(Y) \,:\, \exists\, \phi \in \mathcal{M}_{X,Y} \text{ such that } \boldsymbol{i}^{\phi} = \boldsymbol{j}\big\}.$$

Obviously, the *function graph*

$$\Gamma(\phi) = \big\{(\boldsymbol{i},\boldsymbol{j}) \in \mathcal{D}(\phi) \times \mathcal{R}(\phi) \,:\, \boldsymbol{i}^{\phi} = \boldsymbol{j}\big\}$$

of a partial morphism $\phi \in \mathcal{M}_{X,Y}$ is a subset of $\mathcal{R}_{X,Y}$.[6] Now let $\mathfrak{p}$ be a property defined on $\mathcal{R}_{X,Y}$. Then $\mathfrak{p}$ induces a binary relation

$$\mathcal{R}^{\mathfrak{p}}_{X,Y} = \{\boldsymbol{r} \in \mathcal{R}_{X,Y} \,:\, \boldsymbol{r} \text{ has property } \mathfrak{p}\}.$$

Items $\boldsymbol{i}$ of $X$ and $\boldsymbol{j}$ of $Y$ are called $\mathfrak{p}$-*similar*, written as $\boldsymbol{i} \sim_{\mathfrak{p}} \boldsymbol{j}$, if $(\boldsymbol{i},\boldsymbol{j})$ is in $\mathcal{R}^{\mathfrak{p}}_{X,Y}$. A $\mathfrak{p}$-*morphism* is a partial morphism $\phi \in \mathcal{M}_{X,Y}$ such that the function graph $\Gamma(\phi)$ is a subset of $\mathcal{R}^{\mathfrak{p}}_{X,Y}$. Thus, a $\mathfrak{p}$-morphism locally preserves the property $\mathfrak{p}$. Two subgraphs $X'$ of $X$ and $Y'$ of $Y$ are called $\mathfrak{p}$-*morphic* if there is a $\mathfrak{p}$-morphism from $X$ to $Y$ with $X' = X\big[\mathcal{D}(\phi)\big]$ and $Y' = Y\big[\mathcal{R}(\phi)\big]$. By $\mathcal{M}^{\mathfrak{p}}_{X,Y}$, we denote the subset of all $\mathfrak{p}$-morphisms from $\mathcal{M}_{X,Y}$. A subset $\mathcal{M}$ of $\mathcal{M}_{X,Y}$ is called $\mathfrak{p}$-*closed* if $\mathfrak{p}$ is a property on $\mathcal{R}_{X,Y}$ and $\mathcal{M} = \mathcal{M}^{\mathfrak{p}}_{X,Y}$.

The next result provides some important examples of $\mathfrak{p}$-closed sets.

***Proposition 3.2***
*The following subsets of $\mathcal{M}_{X,Y}$ are $\mathfrak{p}$-closed:*

1. $\mathcal{M}^{1}_{X,Y}$ = *set of all partial morphisms*

6. The term *graph* is uniformly used by mathematicans to denote two different objects: (1) a structure consisting of a set of vertices and edges and (2) a relation consisting of all pairs of input-output values determined by a function. To distinguish between graphs from graph theory and graphs from analytical geometry, we refer to the former as graphs and to the latter as function graphs.

2. $\mathcal{M}^2_{X,Y}$ = *set of all partial monomorphisms*
3. $\mathcal{M}^3_{X,Y}$ = *set of all partial homomorphisms*
4. $\mathcal{M}^4_{X,Y}$ = *set of all partial isomorphisms*

***Proof*** We only show the assertion for the set $\mathcal{M}^4_{X,Y}$. The proofs for the other sets are similar. Let $X$ and $Y$ be attributed graphs with adjacency matrices $\boldsymbol{X} = (\boldsymbol{x}_{ij})$ and $\boldsymbol{Y} = (\boldsymbol{y}_{ij})$. We define a property $\mathfrak{p}$ on $\mathcal{R}_{X,Y}$ such that

$$\mathcal{R}^{\mathfrak{p}}_{X,Y} = \Big\{(\boldsymbol{i},\boldsymbol{j}) \in I(X) \times I(Y) : \boldsymbol{x}_{\boldsymbol{i}} = \boldsymbol{y}_{\boldsymbol{j}}, \big|\{i_1,i_2\}\big| = \big|\{j_1,j_2\}\big|\Big\},$$

where $\boldsymbol{i} = (i_1,i_2)$ and $\boldsymbol{j} = (j_1,j_2)$. The condition

$$\big|\{i_1,i_2\}\big| = \big|\{j_1,j_2\}\big| \tag{3.6}$$

requires that both $\boldsymbol{i}$ and $\boldsymbol{j}$ are either vertex items or non-vertex items. Thus, $\mathcal{R}^{\mathfrak{p}}_{X,Y}$ relates isomorphic items in the sense that there exists a partial isomorphism $\phi$ between $X$ and $Y$ such that $\boldsymbol{i}^{\phi} = \boldsymbol{j}$.

Next, we show that $\mathcal{M}^4_{X,Y} = \mathcal{M}^{\mathfrak{p}}_{X,Y}$. Let $\phi$ be a partial isomorphism from $X$ to $Y$, let $\boldsymbol{i}$ be an item of $X$, and let $\boldsymbol{i}^{\phi} = \boldsymbol{j}$ be the image of $\boldsymbol{i}$ in $Y$. Since $\phi$ is a partial isomorphism, we have $\boldsymbol{x}_{\boldsymbol{i}} = \boldsymbol{y}_{\boldsymbol{j}}$. In addition, from the bijectivity of $\phi$ follows (3.6). Hence, the function graph $\Gamma(\phi)$ of $\phi$ is a subset of $\mathcal{R}_{X,Y}$. This proves $\mathcal{M}^4_{X,Y} \subseteq \mathcal{M}^{\mathfrak{p}}_{X,Y}$.

Now assume that $\phi$ is a $\mathfrak{p}$-morphism from $\mathcal{M}^{\mathfrak{p}}_{X,Y}$. Since $\Gamma(\phi)$ is a subset of $\mathcal{R}_{X,Y}$, it is sufficient to show that $\phi$ is bijective. Assume that $\phi$ is not bijective. Then $\phi$ is not injective, because $\phi$ is a partial morphism. Hence, there are distinct vertices $i_1, i_2$ of $X$ with $i_1^{\phi} = i_2^{\phi}$ contradicting (3.6). This proves $\mathcal{M}^{\mathfrak{p}}_{X,Y} \subseteq \mathcal{M}^4_{X,Y}$. Combining both results yields the assertion. ■

Next, we provide an example of a subset of partial morphisms that is not $\mathfrak{p}$-closed.

***Example 3.8***
Consider the set $\mathcal{M}^{(m)}_{X,Y}$ of partial morphisms $\phi$ with

$$\big|\mathcal{D}(\phi)\big| \leq m < |X|,$$

where $m \geq 2$. There is no property $\mathfrak{p}$ on $\mathcal{R}_{X,Y}$ such that $\mathcal{M}^{(m)}_{X,Y}$ is $\mathfrak{p}$-closed.

***Proof*** Let $\mathfrak{p}$ denote the property on $\mathcal{R}_{X,Y}$ such that $\mathcal{M}_{X,Y}$ is $\mathfrak{p}$-closed. According to Proposition 3.2, such a property exists. Consider the subset $\mathcal{M}^{(2)}_{X,Y}$. It is easy to see that

$$\bigcup_{\phi \in \mathcal{M}^{(2)}_{X,Y}} \Gamma(\phi) = \mathcal{R}^{\mathfrak{p}}_{X,Y}.$$

On one hand, the relation $\mathcal{R}^{\mathfrak{p}}_{X,Y}$ is too large, because it admits arbitrary partial morphisms as $\mathfrak{p}$-morphisms. On the other hand, $\mathcal{R}^{\mathfrak{p}}_{X,Y}$ is a minimal set in the following sense: If we remove an element $(\boldsymbol{i},\boldsymbol{j})$ from $\mathcal{R}^{\mathfrak{p}}_{X,Y}$, then the morphism $\phi \in \mathcal{M}^{(2)}_{X,Y}$ with $\boldsymbol{i}^{\phi} = \boldsymbol{j}$ is no longer a $\mathfrak{p}$-morphism. This shows the assertion. ■

The following proposition summarizes some simple but useful rules to express complex $\mathfrak{p}$-closed sets in terms of simple ones.

***Proposition 3.3***
*Let $\mathfrak{p}_1$ and $\mathfrak{p}_2$ be properties on $\mathcal{R}_{X,Y}$. Then*

1. *$\mathfrak{p} = \mathfrak{p}_1 \wedge \mathfrak{p}_2$ is a property on $\mathcal{R}_{X,Y}$ and*
$$\mathcal{M}^{\mathfrak{p}}_{X,Y} = \mathcal{M}^{\mathfrak{p}_1}_{X,Y} \cap \mathcal{M}^{\mathfrak{p}_2}_{X,Y}$$
2. *$\mathfrak{p} = \mathfrak{p}_1 \vee \mathfrak{p}_2$ is a property on $\mathcal{R}_{X,Y}$ and*
$$\mathcal{M}^{\mathfrak{p}}_{X,Y} = \mathcal{M}^{\mathfrak{p}_1}_{X,Y} \cup \mathcal{M}^{\mathfrak{p}_2}_{X,Y}$$
3. *$\mathfrak{p} = \neg\mathfrak{p}_1$ is a property on $\mathcal{R}_{X,Y}$ and*
$$\mathcal{M}^{\mathfrak{p}}_{X,Y} = \mathcal{M}_{X,Y} \setminus \mathcal{M}^{\mathfrak{p}_1}_{X,Y}$$

***Proof*** The proof is almost trivial. Therefore, we only exemplify the proof of the first assertion. Clearly, with $\mathfrak{p}_1$ and $\mathfrak{p}_2$, the conjunction $\mathfrak{p}$ is also a property on $\mathcal{R}_{X,Y}$. Now observe that

$$\mathcal{R}^{\mathfrak{p}}_{X,Y} = \{\boldsymbol{r} \in \mathcal{R}_{X,Y} \,:\, \boldsymbol{r} \text{ has property } \mathfrak{p} = \mathfrak{p}_1 \wedge \mathfrak{p}_2\} = \mathcal{R}^{\mathfrak{p}_1}_{X,Y} \cap \mathcal{R}^{\mathfrak{p}_2}_{X,Y}.$$

Hence,

$$\phi \in \mathcal{M}^{\mathfrak{p}}_{X,Y} \Leftrightarrow \Gamma(\phi) \subseteq \mathcal{R}^{\mathfrak{p}}_{X,Y} \Leftrightarrow \Gamma(\phi) \subseteq \mathcal{R}^{\mathfrak{p}_1}_{X,Y} \cap \mathcal{R}^{\mathfrak{p}_2}_{X,Y} \Leftrightarrow \phi \in \mathcal{M}^{\mathfrak{p}_1}_{X,Y} \cap \mathcal{M}^{\mathfrak{p}_2}_{X,Y}.$$

■

### 3.5.2 The $\mathfrak{p}$-Graph Matching Problem

For a property $\mathfrak{p}$ on $\mathcal{R}_{X,Y}$, the $\mathfrak{p}$-*graph matching problem* ($\mathfrak{p}$GMP) is defined by

$$\begin{aligned} &\text{maximize} && f(\phi, X, Y) = \sum_{\boldsymbol{i} \in \mathcal{D}(\phi)} \kappa_{\boldsymbol{i}\boldsymbol{i}^\phi} \\ &\text{subject to} && \phi \in \mathcal{M}^{\mathfrak{p}}_{X,Y}. \end{aligned} \tag{3.7}$$

From Proposition 3.2 and 3.3 follows that problem (3.7) is sufficiently general to encompass a broad range of graph matching problems including those listed in Section 2.4.

In order to cast a $\mathfrak{p}$GMP to an equivalent MWCP, it is useful to distinguish between global and local maxima of (3.7).[7] The local maxima of $f$ correpond to maximal weight $\mathfrak{p}$-morphisms of $\mathcal{M}^{\mathfrak{p}}_{X,Y}$. A *maximal weight* $\mathfrak{p}$-*morphism* of $\mathcal{M}^{\mathfrak{p}}_{X,Y}$

7. As for the maximum weight clique formulation, the notion *local maximum* depends on how we define a neighborhood of a $\mathfrak{p}$-morphism.

is a $\mathfrak{p}$-morphism $\phi : V(X) \to V(Y)$ such that

$$\mathcal{D}(\phi) \subseteq \mathcal{D}(\psi) \;\Rightarrow\; f(\phi) \geq f(\psi)$$

for all $\mathfrak{p}$-morphisms $\psi \in \mathcal{M}^{\mathfrak{p}}_{X,Y}$. Thus, it is impossible to extend the domain of a maximal weight $\mathfrak{p}$-morphism in order to increase the matching objective $f$. The global maxima of $f$ correspond to the maximum weight $\mathfrak{p}$-morphism of $\mathcal{M}^{\mathfrak{p}}_{X,Y}$. A *maximum weight $\mathfrak{p}$-morphism* of $\mathcal{M}^{\mathfrak{p}}_{X,Y}$ is a $\mathfrak{p}$-morphism that maximizes $f$.

***Notation 3.9***
Consider the $\mathfrak{p}$-GMP (3.7).

- $\mathcal{S}_f = \mathcal{M}^{\mathfrak{p}}_{X,Y}$ denotes the feasible region of all $\mathfrak{p}$-morphisms.
- $\mathcal{S}^{\times}_f$ denotes the subset of all maximal weight $\mathfrak{p}$-morphisms.
- $\mathcal{S}^{*}_f$ denotes the subset of all maximum weight $\mathfrak{p}$-morphisms.

We use $\mathcal{S}_f$ instead of $\mathcal{M}^{\mathfrak{p}}_{X,Y}$ to indicate an optimization point of view. The set $\mathcal{S}_f$ is the set of feasible solution points of the matching objective $f$. We write $\mathcal{M}^{\mathfrak{p}}_{X,Y}$ to indicate a descriptive approach of the graph matching problem.

### 3.5.3 A Clique Formulation of the $\mathfrak{p}$GMP

The link between the $\mathfrak{p}$GMP and the MWCP is the quadratic integer program (3.3) presented in Section 3.3. We transform (3.3) to a MWCP by reshaping the Kronecker $\kappa$-product $\boldsymbol{X} \otimes \boldsymbol{Y}$. This transformation leads us to the notion of the $\kappa$-association graph.

Let $\boldsymbol{K} = (\kappa_{ij,kl})$ be the Kronecker $\kappa$-product of $\boldsymbol{X}$ and $\boldsymbol{Y}$. Consider the matrix $\widetilde{\boldsymbol{Z}}$ with elements

$$\tilde{z}_{ik,jl} = \begin{cases} \kappa_{ij,kl} & : \text{ if } (i,j) \sim_{\mathfrak{p}} (k,l) \\ \epsilon & : \text{ otherwise} \end{cases}$$

for all items $(i,j) \in I(X)$ and $(k,l) \in I(Y)$. We obtain $\widetilde{\boldsymbol{Z}}$ from the Kronecker $\kappa$-product by rearranging the elements and deleting entries corresponding to incompatible items with respect to $\sim_{\mathfrak{p}}$. A *$\kappa$-association graph* $Z = X \diamond Y$ is a complex graph determined by the adjacency matrix

$$\boldsymbol{Z} = \widetilde{\boldsymbol{Z}} + \widetilde{\boldsymbol{Z}}^{\mathsf{T}} - \boldsymbol{D}_{\widetilde{\boldsymbol{Z}}}. \tag{3.8}$$

This definition may require some elucidatory remarks:

- Recall that a complex graph is not related to the field of complex numbers. A complex graph is composed of a proper graph in a graph-theoretical sense and of dangling edges. See also Section 2.2.7 for a detailed definition.
- Every graph isomorphic to $Z$ is a $\kappa$-association graph. Hence, we write $Z$ is *a* $\kappa$-association graph rather than *the* $\kappa$-association graph of $X$ and $Y$.

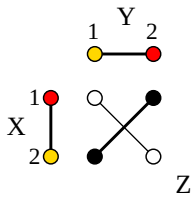



**Figure 3.2** Shown are the two attributed graphs $X$ and $Y$ from Example 3.4 and their $\kappa$-association graph $Z = X \diamond Y$ defined by the matrix $\boldsymbol{Z}$. The real part $\Re[Z]$ of $Z$ is highlighted. The imaginary part $\Im[Z]$ consists of the rest of $Z$. The thin line is a dangling edge connecting the two imaginary vertices of $Z$.

- $Z$ is a complex graph, because it may have vertices associated with the void attribute $\epsilon$. The real part $\Re[Z]$ of $Z$ is a weighted graph with vertex and edge set

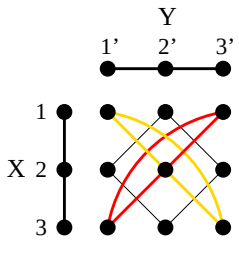
$$V\big(\Re[Z]\big) = \Big\{(i,j) \in V(X) \times V(Y) \,:\, (i,i) \sim_{\mathfrak{p}} (j,j)\Big\}$$
$$E\big(\Re[Z]\big) = \Big\{\big((i,k),(j,l)\big) \in V(Z) \times V(Z) \,:\, (i,j) \sim_{\mathfrak{p}} (k,l)\Big\}.$$

- Definition (3.8) of $\boldsymbol{Z}$ is motivated to accommodate matching problems of directed and undirected attributed graphs. Recall from Remark 3.2 that either, both, or neither of the compatibility values $\kappa_{ij,kl}$ and $\kappa_{ji,lk}$ contribute to the matching objective of the $\mathfrak{p}$GMP, provided that $(i,j)$ is a non-vertex item of $X$ that can be mapped to $(k,l) \in I(Y)$.
  As opposed to the $\mathfrak{p}$GMP, the weight clique formulation (3.5) is defined for undirected graphs and counts edges only once. This definition is in line with the definitions of the classical MCP and MVCP. To ensure equivalence between graph matching and clique search, we define $\boldsymbol{Z}$ as in (3.8).
- Definition (3.8) involves undefined arithmetic operations with the symbol $\epsilon$. We set $x \pm \epsilon = \epsilon$ for all $x \in \underline{\mathbb{R}}$.

In the following, we consider some examples. To illustrate the transformation from the Kronecker $\kappa$-product to a $\kappa$-association graph, let us enhance Example 3.4.

***Example 3.10***
We again consider the graphs shown in Figure 3.1 of Section 3.3 with $\kappa$-Kronecker product $\boldsymbol{K} = \boldsymbol{X} \otimes \boldsymbol{Y}$ of the form

$$\boldsymbol{K} = \left(\begin{array}{cc|cc} \kappa_{11,11} & \kappa_{11,12} & \kappa_{12,11} & \kappa_{12,12} \\ \kappa_{11,21} & \kappa_{11,22} & \kappa_{12,21} & \kappa_{12,22} \\ \hline \kappa_{21,11} & \kappa_{21,12} & \kappa_{22,11} & \kappa_{22,12} \\ \kappa_{21,21} & \kappa_{21,22} & \kappa_{22,21} & \kappa_{22,22} \end{array}\right) = \left(\begin{array}{cc|cc} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{array}\right).$$




| X − Y | X − Y |
|---|---|
| 1 − 1' | 1 − 3' |
| 2 − 2' | 2 − 2' |
| 3 − 3' | 3 − 1' |

**Figure 3.3** A $\kappa$-association graph $Z$ of binary graphs $X$ and $Y$ for the maximum common induced subgraph problem. For convenience, we assume compatibility values such that each vertex and each edge of $Z$ is colored with weight 1. Maximum cliques are highlighted. The table on the right hand side shows the vertex correspondences derived from both maximum cliques.

Rearranging the elements of $\boldsymbol{K}$ in the order of $\widetilde{\boldsymbol{Z}}$ gives

$$\widetilde{\boldsymbol{K}} = \begin{pmatrix} \kappa_{11,11} & \kappa_{11,12} & \kappa_{11,21} & \kappa_{11,22} \\ \kappa_{12,11} & \kappa_{12,12} & \kappa_{12,21} & \kappa_{12,22} \\ \kappa_{21,11} & \kappa_{21,12} & \kappa_{21,21} & \kappa_{21,22} \\ \kappa_{22,11} & \kappa_{22,12} & \kappa_{22,21} & \kappa_{22,22} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

In this particular example, the matrices $\boldsymbol{K}$ and $\widetilde{\boldsymbol{K}}$ coincide. Next we delete matrix elements corresponding to incompatible items and multiply off-diagonal elements by 2 in order to obtain the adjacencey matrix $\boldsymbol{Z}$ of the $\kappa$-association graph $Z = X \diamond Y$. Here, incompatibilities correspond to zero entries in $\widetilde{\boldsymbol{K}}$. We arrive at

$$\boldsymbol{Z} = \begin{pmatrix} \epsilon & \epsilon & \epsilon & 2 \\ \epsilon & 1 & 2 & \epsilon \\ \epsilon & 2 & 1 & \epsilon \\ 2 & \epsilon & \epsilon & \epsilon \end{pmatrix}.$$

Figure 3.2 shows the resulting complex graph $Z$ and its real part $\Re[Z]$. In the plot, the vertices of $Z$ are arranged in a matrix form. A vertex of $Z$ positioned in the $i$-th row and $j$-th column is colored with $z_{ij,ij}$. All edges of $Z$ are labeled with 2. Since $\Re[Z]$ is a complete and positively weighted graph, its vertices form a maximum weight clique with weight 4. This is in exact correspondence with the optimal value obtained in Example 3.4.

Deriving a $\kappa$-association graph from the Kronecker $\kappa$-product is counter-intuitive, but useful to establish a link to the $\mathfrak{p}$GMP. The next example shows how we can construct an association graph directly from the graph $X$ and $Y$ to be matched.

***Example 3.11***
Figure 3.3 shows a $\kappa$-association graph $Z = X \diamond Y$ of two isomorphic binary graphs $X$ and $Y$ for solving the common induced subgraph isomorphism problem. We choose the compatibility function (2.2) with $\alpha_V = 1$, $\alpha_E = \alpha_{\overline{E}} = 0.5$. Thus, all vertices and edges of $Z$ are colored with 1.

Items are inserted into $Z$ if pairs of items from $X$ and $Y$ are $\mathfrak{p}$-similar:

1. Since all vertices of $X$ and $Y$ have the same color 1, the $\kappa$-association graph $Z$ is a graph consisting of nine vertices and with an empty imaginary part. Each vertex of $Z$ can be written as $(i, j)$ with $i \in V(X)$, $j \in V(Y)$, and $i, j \in [1:3]$. Note that we have annotated a stroke to vertices of $Y$ for convenience of distinction. In the plot, the vertices of $Z$ are arranged in matrix form so that vertex $(i, j)$ is positioned in the $i$-th row and $j$-th column.

2. Edges are inserted into $Z$ whenever the relations in $X$ and $Y$ coincide. Vertices $(1, 1')$ and $(2, 2')$ are adjacent in $Z$, because $(1, 2)$ is an edge of $X$ and $(1', 2')$ is an edge of $Y$. Similarly, vertices $(1, 1')$ and $(3, 3')$ are adjacent in $Z$, because $(1, 3)$ and $(1', 3')$ are both non-edges in $X$ and $Y$, respectively.

3. Non-edges occur in $Z$ to incorporate constraints or whenever the relations in $X$ and $Y$ differ. Vertices $(1, 1')$ and $(1, 2')$ are not connected by an edge in $Z$, because the correspondences $1 \mapsto 1'$ and $1 \mapsto 2'$ violate the constraint that we maximize over well-defined partial mappings. Similarly, vertices $(1, 1')$ and $(2, 1')$ are not adjacent in $Z$, because the correspondences $1 \mapsto 1'$ and $2 \mapsto 1'$ violate the constraint that we maximize over partial isomorphisms. Finally, vertices $(1, 1')$ and $(2, 3')$ are not adjacent, because $(1, 2)$ is an edge in $X$ but $(1', 3')$ is a non-edge in $Y$.

Although we are only concerned with the real part $\mathfrak{R}[Z]$ of $Z$, we refer to the complex graph $Z$ for the sake of technical presentation.[8] With the concept of a $\kappa$-association graph, we are now in the position to cast a $\mathfrak{p}$GMP to an equivalent MWCP. To formulate Theorem 3.1, we make use of Notation 3.9 on page 50.

***Theorem 3.1 (Clique Formulation of Graph Matching)***
*Let $X \diamond Y$ be a $\kappa$-association graph of attributed graphs $X$ and $Y$. Then the $\mathfrak{p}$GMP of $X$ and $Y$ is equivalent to the MWCP of the real part $Z = \mathfrak{R}\left[X \diamond Y\right]$, that is, there are bijections*

1. $\Phi : \mathcal{C}_Z \to \mathcal{S}_f$
2. $\Phi^\times : \mathcal{C}_Z^\times \to \mathcal{S}_f^\times$
3. $\Phi^* : \mathcal{C}_Z^* \to \mathcal{S}_f^*$

*with*

$$\omega(C) = f\big(\Phi(C)\big)$$

*for all cliques $C \in \mathcal{C}_Z$.*

***Proof*** For the real part $Z$ of $X \diamond Y$ we refer to the ordering of vertices of the complex graph $X \diamond Y$. According to Notation 3.9 on page 50, we may write $\mathcal{S}_f = \mathcal{M}_{X,Y}^{\mathfrak{p}}$. Since we can identify morphisms with match matrices, we consider the subset $\mathcal{M}_{\mathfrak{p}}^{n \times m}$ of match matrices representing morphisms from $\mathcal{M}_{X,Y}^{\mathfrak{p}}$. We first

8. Referring to $Z$ allows us to retain the numbering of vertices and to refer to relevant elements of the Kronecker $\kappa$-product.

show that there is a bijection

$$\Phi : \mathcal{C}_Z \to \mathcal{M}_{\mathfrak{p}}^{n\times m}, \quad C \mapsto \boldsymbol{M}_C$$

such that $\omega(C) = f(\boldsymbol{M}_C)$ for all $C \in \mathcal{C}_Z$, where $f$ is the matching objective of the $\mathfrak{p}$GMP (3.7). With each clique $C \in \mathcal{C}_Z$ we associate a matrix $\boldsymbol{M}_C = (m_{ij})$ such that

$$m_{ij} = \begin{cases} 1 & : \text{ if } (i,j) \in C \\ 0 & : \text{ otherwise} \end{cases}.$$

We show that $\boldsymbol{M}_C$ is feasible, that is, $\boldsymbol{M}_C \in \mathcal{M}_{\mathfrak{p}}^{n\times m}$. Let $i, j \in [1\!:\!n]$ and $k, l \in [1\!:\!m]$ be indices such that $m_{ik} = m_{jl} = 1$. By construction, $(i,k)$ and $(j,l)$ are members of clique $C$. Since $Z[C]$ is complete, there is an edge incident with $(i,k)$ and $(j,l)$. Hence, $(i,j) \sim_{\mathfrak{p}} (k,l)$ and, therefore, $\Gamma(\boldsymbol{M}_C) \subseteq \mathcal{R}^{\mathfrak{p}}_{X,Y}$. This implies $\boldsymbol{M}_C \in \mathcal{M}_{\mathfrak{p}}^{n\times m}$.

Similarly, with each feasible match matrix $\mathcal{M} = (m_{ij})$ we associate a subset $C_{\boldsymbol{M}}$ of $V(X) \times V(Y)$ with

$$(i,j) \in C_{\boldsymbol{M}} \;\Leftrightarrow\; m_{ij} = 1.$$

Since $\boldsymbol{M}$ is feasible, $C_{\boldsymbol{M}}$ is a clique in $Z$. It is straightforward to show that both associations give rise to well-defined mappings

$$\Phi : \mathcal{C}_Z \to \mathcal{M}_{\mathfrak{p}}^{n\times m}, \quad C \mapsto \boldsymbol{M}_C$$

and

$$\Phi' : \mathcal{M}_{\mathfrak{p}}^{n\times m} \to \mathcal{C}_Z, \quad \boldsymbol{M} \mapsto C_{\boldsymbol{M}}.$$

From $\Phi \circ \Phi' = \mathrm{id}$ on $\mathcal{M}_{\mathfrak{p}}^{n\times m}$ and $\Phi' \circ \Phi = \mathrm{id}$ on $\mathcal{C}_Z$, it follows that $\Phi$ is bijective.

It remains to show that $\omega(C) = f(\boldsymbol{M}_C)$ for all $C \in \mathcal{C}_Z$. Let $C$ be a clique of $Z$ with adjacency matrix $\boldsymbol{Z} = (z_{ik,jl})$. From $\mathbf{1}_C = \mathrm{vec}(\boldsymbol{M}_C)$ follows

$$\begin{aligned} \omega(C) &= \frac{1}{2}\Big(\mathbf{1}_C^{\mathsf{T}} \boldsymbol{O}_{\boldsymbol{Z}} \mathbf{1}_C\Big) + \mathbf{1}_C^{\mathsf{T}} \boldsymbol{d}_{\boldsymbol{Z}} \\ &= \frac{1}{2}\Big(\,\mathrm{vec}(\boldsymbol{M}_C)^{\mathsf{T}} \boldsymbol{O}_{\boldsymbol{Z}}\,\mathrm{vec}(\boldsymbol{M}_C)\,\Big) + \mathrm{vec}(\boldsymbol{M}_C)^{\mathsf{T}} \boldsymbol{d}_{\boldsymbol{Z}}. \end{aligned}$$

We define

$$\boldsymbol{Z'} = \frac{1}{2}\boldsymbol{O}_{\boldsymbol{Z}} + \boldsymbol{D}_{\boldsymbol{Z}}.$$

Then we can rewrite $\omega(C)$ in terms of $\boldsymbol{Z'} = (z'_{ik,jl})$ by

$$\omega(C) = \mathrm{vec}(\boldsymbol{M}_C)^{\mathsf{T}} \boldsymbol{Z'}\,\mathrm{vec}(\boldsymbol{M}_C)\,.$$

Let $(i,j) \in I(X)$ and $(k,l) \in I(Y)$, and let $\boldsymbol{M} = (m_{ij}) \in \mathcal{M}_{\mathfrak{p}}^{n\times m}$ be a feasible match matrix. We distinguish between two cases:

*Case 1* ($(i,j) \sim_{\mathfrak{p}} (k,l)$)*:* We have $m_{ik}m_{jl} = 1$ and $z'_{ik,jl} = \kappa_{ij,kl}$.

*Case 2* ($(i,j) \not\sim_{\mathfrak{p}} (k,l)$)*:* We have $m_{ik}m_{jl} = 0$ and $z'_{ik,jl} = \epsilon$.

Hence, for feasible match matrices $\boldsymbol{M} = (m_{ij})$, all terms $z'_{ik,jl}m_{ik}m_{jl}$ are well defined and we have the equality

$$z'_{ik,jl}m_{ik}m_{jl} = \kappa_{ij,kl}m_{ik}m_{jl}.$$

Thus, we may replace $\boldsymbol{Z'}$ by $\boldsymbol{K}$ and obtain

$$\omega(C) = \mathrm{vec}(\boldsymbol{M}_C)^{\mathsf{T}}\boldsymbol{K}\,\mathrm{vec}(\boldsymbol{M}_C) = f(\boldsymbol{M}_C).$$

Bijection of $\Phi^{\times}$ and $\Phi^{*}$ follows from the bijection of $\Phi$ and the relation $\omega(C) = f(\boldsymbol{M}_C)$ for all $C \in \mathcal{C}_Z$. This completes the proof. ■

From Proposition 3.2 together with Theorem 3.1, it follows that all graph matching problems of Section 2.4 are equivalent to the MWCP in an association graph.

***Corollary 3.1***
*Let $X$ and $Y$ be attributed graphs. Then the following problems are equivalent to the MWCP in $\mathfrak{R}\left[X \diamond Y\right]$:*

1. *maximum common subgraph problem*
2. *maximum common induced subgraph problem*
   (a) *induced subgraph isomorphism problem*
   (b) *graph isomorphism problem*
3. *maximum common homomorphic subgraph problem*
   (a) *subgraph homomorphism problem*
4. *best $\mathcal{M}$-morphic graph matching problem*
5. *probabilistic graph matching problem*
6. *error correcting graph matching problem*

*For problems* (5) *and* (6)*, the graphs $X$ and $Y$ refer to the extended graphs (see Section 3.2).*

Corollary 3.1 is interesting for two reasons. First, it indicates that in order to show equivalence between graph matching and clique search, it must be shown that the feasible region of a graph matching problem is $\mathfrak{p}$-closed. Second, it shows that a variety of common graph matching problems can be transformed to the MWCP in an association graph.

***Remark 3.12***
*Note that $\mathfrak{p}$-closedness depends on an appropriate description of the feasible region. Consider, for example, the problem of matching trees in order to find the maximum common connected and induced subgraph. This matching problem arises in e.g. shape recognition formulated in terms of* shock *tree matching [284]. It is unclear, how to express the set of partial isomorphisms that preserve connectivity as a*

*local property on $\mathcal{R}_{X,Y}$. An appropriate description of the feasible region $\mathcal{M}$ would be that $\mathcal{M}$ is the set of all partial isomorphisms that can be extended to a connectivity preserving partial isomorphism. Using a property $\mathfrak{p}$ as derived in [284], we can show that $\mathcal{M}$ is $\mathfrak{p}$-closed. The feasible region then also consists of partial isomorphisms between unconnected induced subgraphs. Only the maximal and maximum weight $\mathfrak{p}$-morphisms preserve connectivity.*

The next example enhances Example 3.11 in order to illustrate the equivalence between the $\mathfrak{p}$GMP and the MWCP in an associated $\kappa$-association graph.

***Example 3.13***
Consider the graphs $X$ and $Y$ shown in Figure 3.3. In this example the MWCP reduces to the classical MCP. Since $X$ and $Y$ are isomorphic, the maximum weight $\mathfrak{p}$-morphisms are the isomorphisms between $X$ and $Y$ given in the table on the right hand side of Figure 3.3. As highlighted in the plot, the two isomorphisms are in one-to-one correspondence with the maximum cliques

$$C_1^* = \{(1,1'),(2,2'),(3,3')\}$$
$$C_2^* = \{(1,3'),(2,2'),(3,1')\}.$$

Furthermore, we have four maximal cliques of order 2 corresponding to four partial isomorphisms that cannot be extended:

$$C_1^\times = \{(1,2'),(2,1')\}$$
$$C_2^\times = \{(1,2'),(2,3')\}$$
$$C_3^\times = \{(2,1'),(3,2')\}$$
$$C_4^\times = \{(3,2'),(2,3')\}.$$

Thus, maximal cliques are in one-to-one correspondence with the maximal weight $\mathfrak{p}$-morphisms.

Since all cliques with at least two vertices are maximal, the remaining cliques of $Z$ are singletons. Clearly, each vertex of $Z$ is also a partial isomorphism. In addition, there are no other partial isomorphisms. Hence, putting it all together, we have shown a one-to-one correspondence between the cliques of $Z$ and the partial isomorphisms from $X$ to $Y$.

We can use the $\mathfrak{p}$GMP of Examples 3.11 and 3.13 to show that $\mathfrak{p}$-closedness is a necessary condition to establish equivalence between the $\mathfrak{p}$GMP and the MWCP. Suppose, for example, that our goal is to find the maximum common induced subgraph of order 2 at most. This corresponds to a feasible region $\mathcal{M}$ consisting of all partial isomorphisms $\phi$ with $\left|\mathcal{D}(\phi)\right| \leq 2$. As in Example 3.8, we can show that there is no property $\mathfrak{p}$ on $\mathcal{R}_{X,Y}$ such that $\mathcal{M}$ is $\mathfrak{p}$-closed. Moreover, we have

$$\bigcup_{\phi\in\mathcal{M}} \Gamma(\phi) = \mathcal{R}_{X,Y}^{\mathfrak{q}},$$

where $\mathfrak{q}$ denotes the property for arbitrary partial isomorphisms. Hence, we arrive at the same $\kappa$-association graph as in Example 3.11 showing that the union of three cliques of order 2 can result in a clique of order 3. Therefore, our matching problem is not equivalent to the MWCP. This supports that $\mathfrak{p}$-closedness is not only a sufficient, but also a necessary condition.

***Interpretation***

The common view of the graph matching problem in contemporary literature is that of an optimization or search problem. From both points of view, the graph matching problem asks for a feasible morphism between two given graphs that maximizes some matching objective. Theorem 3.1 provides a pure graph-theoretical view of the matching problem.

Given two graphs $X$ and $Y$, a partial morphism $\phi$ is a mapping between the vertex sets $V(X)$ and $V(Y)$. The basic components of $\phi$ are correspondences of the form $i \mapsto j$ with $i \in V(X)$ and $j \in V(Y)$. A $\kappa$-association graph $Z = X \diamond Y$ represents the space of all possible correpondences between vertices from $X$ and $Y$. Vertices of the imaginary part of $Z$ represent incompatible correspondences and can therefore be ignored. The remaining vertices from the real part of $Z$ represent the space of compatible correspondences. Edges of $\mathfrak{R}[Z]$ serve to represent compatible correspondences between relations (edges and non-edges) in $X$ and $Y$. Compatibility values assigned to the vertices and edges of $\mathfrak{R}[Z]$ measure to which extent both kinds of correspondences are compatible. Non-edges of $\mathfrak{R}[Z]$, as well as the imaginary part of $Z$, represent infeasible correspondences and thus implement the constraints of the graph matching problem. By construction, the cliques of $\mathfrak{R}[Z]$ represent the feasible morphisms from $X$ to $Y$ and the maximum weight cliques of $\mathfrak{R}[Z]$, the solutions to the original graph matching problem.

## 3.6 Continuous Formulation of the Maximum Weight Clique Problem

The MWCP is a discrete or combinatorial optimization problem in which the term discrete refers to the discrete feasible domain. Solution methods for discrete optimization problems can be classified into search-based and continuous-based approaches. A typical search-based method explores a finite discrete space to construct a solution by generating a sequence of partial solutions. Continuous-based methods extremize continuous characterizations of the discrete problem. These characterizations include equivalent continuous formulations of the discrete optimization problem or embed the discrete domain in a larger continuous space (relaxation). The topological and geometric properties of continuous approaches can be exploited to apply existing or to develop new algorithms for solving discrete optimization problems.

An appropriate formulation of an optimization problem is essential in solving that problem. The Motzkin-Strauss formulation [271] and its spurious-free extensions [35, 112, 283] are popular continuous characterizations of the classical MCP, for which efficient heuristics have been devised. What makes the expanded version of the Motzkin-Strauss Theorem so useful is a one-to-one correspondence between its optimal (local) solutions and the maximum (maximal) cliques of the underlying graph. Though the Motzkin-Strauss Theorem has been generalized to the MVCP [112, 37], it is unclear how to derive a generalization to the MWCP.

This section therefore follows a different approach and maps the MWCP to a continuous quadratic form constrained over the unit hypercube so that there is a clear one-to-one correspondence between the optimal (local) solution of the quadratic form and the maximum (maximal) weight cliques.

Assume that $X = (V, E, \boldsymbol{X})$ is a normalized graph from $\mathcal{G}_{\mathbb{U}}$, where $\varepsilon = 0$. In the context of graph matching, we can make this assumption without loss of generality whenever the compatibility values are nonnegative. In order to provide a continuous characterization of the MWCP, we introduce the *constraint matrix* $\boldsymbol{A} = (a_{ij})$ with elements

$$a_{ij} = \begin{cases} 1 & : \text{ if } (i,j) \in \overline{E} \\ 0 & : \text{ otherwise.} \end{cases}$$

From a similar argumentation as in the proof of Proposition 3.1, it follows that

$$\boldsymbol{x}^{\mathsf{T}} \boldsymbol{A} \boldsymbol{x} = 0$$

if $\boldsymbol{x}$ is a characteristic vector of a clique in $X$. Using the constraint matrix $\boldsymbol{A}$, we define the matrix

$$\boldsymbol{Q} = \boldsymbol{X} - \boldsymbol{D}_{\boldsymbol{X}} - \mu \boldsymbol{A},$$

where $\mu$ is an appropriately chosen *penalty term* parameter. By

$$\boldsymbol{c} = \boldsymbol{d}_{\boldsymbol{X}}$$

we denote the diagonal of $\boldsymbol{X}$. Consider the continuous quadratic problem

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) = -\dfrac{1}{2} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{Q} \boldsymbol{x} - \boldsymbol{c}^{\mathsf{T}} \boldsymbol{x} \\ \text{subject to} & \boldsymbol{x} \in \mathbb{U}^n. \end{array} \tag{3.9}$$

Theorem 3.2 shows that formulation (3.9) and the MWCP are equivalent, provided that the penalty term $\mu$ is sufficiently large. Recall from Section 2.2.3 that $\Delta_w(X)$ denotes the maximum weight degree of $X$.

***Theorem 3.2 (Continuous Formulation of the MWCP)***
*Consider the quadratic program (3.9). Assume that the penalty term $\mu$ satisfies the following inequality*

$$\mu > \Delta_{\mathrm{w}}\left(X\right) + \|\boldsymbol{c}\|_{\infty}. \tag{3.10}$$

*Then*

1. $C \in \mathcal{C}_X^{\times} \;\Leftrightarrow\; \boldsymbol{x}_C \in \mathcal{S}_f^{\times}$,
2. $C \in \mathcal{C}_X^{*} \;\Leftrightarrow\; \boldsymbol{x}_C \in \mathcal{S}_f^{*}$,
3. *all minima of $f$ in $\mathbb{U}^n$ are strict,*

*where $\mathcal{S}_f^{\times}$ ($\mathcal{S}_f^*$) denotes the set of local (global) minima of $f$.*

In the remainder of this section we prove Theorem 3.2.

### 3.6.1 Proof

We first summarize the notations we use in our derivation.

***Notation 3.14***

| | |
|---|---|
| $\boldsymbol{u}_i$ | $i$-th unit vector with $u_j = \delta_{ij}$ for all $1 \leq j \leq n$ |
| $\mathrm{tr}(\boldsymbol{A})$ | trace of matrix $\boldsymbol{A}$ |
| $\|\boldsymbol{A}\|$ | norm of matrix $\boldsymbol{A}$ |
| $\boldsymbol{O}$ | the off-diagonal matrix $\boldsymbol{O_X} = \boldsymbol{X} - \boldsymbol{D_X}$ of matrix $\boldsymbol{X}$ |
| $\mathbf{0}_{p,q}$ | $(p \times q)$-matrix whose elements are all equal to 0 |

Note that the existence of a minimum point is established by Weierstrass' Theorem stating that any continuous function on a compact set $\mathcal{S}$ attains its minimum in $\mathcal{S}$. To show that $\boldsymbol{Q}$ is indefinite, we require the following Lemma.

***Lemma 3.1***
*Let $\boldsymbol{Q}$ be the matrix of the quadratic program (3.9). Then*

$$\|\boldsymbol{Q}\| > 0.$$

***Proof*** We have $\boldsymbol{Q} = \boldsymbol{O} - \mu\boldsymbol{A}$, where $\boldsymbol{O} = (o_{ij})$ and $\boldsymbol{A} = (a_{ij})$ are complementary $(n \times n)$-matrices, i.e. $o_{ij}a_{ij} = 0$ for all $i, j \in V$. Since $\mu > 0$ and at least one of both matrices $\boldsymbol{O}$ and $\boldsymbol{A}$ is non-zero, we have $\boldsymbol{Q} = \boldsymbol{O} - \mu\boldsymbol{A} \neq \mathbf{0}_{n,n}$. From $\boldsymbol{Q} \neq \mathbf{0}_{n,n}$ follows $\|\boldsymbol{Q}\| > 0$. ■

***Lemma 3.2***
*The matrix $\boldsymbol{Q}$ of the nonlinear program (3.9) is indefinite.*

***Proof*** By definition of $\boldsymbol{O}$ and $\boldsymbol{A}$, we have $\boldsymbol{d_O} = \boldsymbol{d_A} = \mathbf{0}$. This implies that $\boldsymbol{d_Q} = \boldsymbol{d_O} - \mu\boldsymbol{d_A} = \mathbf{0}$. From $\boldsymbol{d_Q} = \mathbf{0}$ follows $tr(\boldsymbol{Q}) = \sum_{i=1}^{n} q_{ii} = 0$. Since $\boldsymbol{Q}$ is symmetric, it has $n$ real eigenvalues $\gamma_1, \ldots, \gamma_n$, which are not necessarily distinct. Now the norm and trace of $\boldsymbol{Q}$ can be expressed in terms of its eigenvalues

$$\|\boldsymbol{Q}\| = \left(\sum_{i=1}^{n} \gamma_i^2\right)^{1/2}$$

$$\mathrm{tr}(\boldsymbol{Q}) = \sum_{i=1}^{n} \gamma_i.$$

According to Lemma 3.1, we have $\|\boldsymbol{Q}\| > 0$. Hence, there exists an eigenvalue

$\gamma_i \neq 0$. From $\gamma_i \neq 0$ and $\mathrm{tr}(\boldsymbol{Q}) = 0$, it follows that there is an an eigenvalue $\gamma_j \neq 0$ such that $\gamma_i \gamma_j < 0$. Thus, $\boldsymbol{Q}$ has positive and negative eigenvalues. This proves that $\boldsymbol{Q}$ is indefinite. ■

The next result states that the minima of $f$ are located at the extreme points of the hypercube $\mathbb{U}^n$. Extreme points of $\mathbb{U}^n$ are the points from the subset $\mathbb{B}^n$. The set of interior points of $\mathbb{U}^n$ is defined by $]0,1[^n$.

***Lemma 3.3***
*Consider the nonlinear program (3.9). Let $\boldsymbol{x}^* \in \mathcal{S}_f^\times$ be a minimum point of $f$ in $\mathbb{U}^n$. Then $\boldsymbol{x}^*$ is an extreme point.*

***Proof*** According to Lemma 3.2, the Hessian $\boldsymbol{H}_f(\boldsymbol{x}) = -\boldsymbol{Q}$ of $f$ is indefinite for all $\boldsymbol{x} \in \mathbb{R}^n$. Thus, $f$ does not satisfy the second-order necessary conditions for the unconstrained case ([239], 6.1, Prop. 3). Therefore, the minimum point $\boldsymbol{x}^*$ is not an interior point of the unit hypercube $\mathbb{U}^n$.

Now assume that $\boldsymbol{x} = \boldsymbol{x}^*$ is neither an extreme point nor an interior point of $\mathbb{U}^n$. We show that there is a feasible direction $\boldsymbol{d}$ at $\boldsymbol{x}$ along which $f$ is decreasing. We have

$$\begin{aligned} f(\boldsymbol{x}) - f(\boldsymbol{x}+\boldsymbol{d}) &= -\frac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} - \boldsymbol{c}^\mathsf{T}\boldsymbol{x} + \frac{1}{2}(\boldsymbol{x}+\boldsymbol{d})^\mathsf{T}\boldsymbol{Q}(\boldsymbol{x}+\boldsymbol{d}) + \boldsymbol{c}^\mathsf{T}(\boldsymbol{x}+\boldsymbol{d}) \\ &= \frac{1}{2}\boldsymbol{d}^\mathsf{T}\boldsymbol{Q}\boldsymbol{d} + \boldsymbol{d}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\mathsf{T}\boldsymbol{d} \\ &= \frac{1}{2}\boldsymbol{d}^\mathsf{T}\boldsymbol{Q}\boldsymbol{d} + \boldsymbol{d}^\mathsf{T}\boldsymbol{f}_{\boldsymbol{x}}, \end{aligned} \tag{3.11}$$

where $\boldsymbol{f}_{\boldsymbol{x}} = (f_{x_1}, \ldots, f_{x_n})^\mathsf{T}$ is the gradient of $f$ at $\boldsymbol{x}$. Since $\boldsymbol{x}$ is not an extreme point, there exists at least one $k \in [1\!:\!n]$ with $0 < x_k < 1$. We distinguish between two cases.

*Case 1:* Assume that there is a $k \in [1\!:\!n]$ such that $0 < x_k < 1$ and $f_{x_k} \neq 0$. Let $\varepsilon$ be a positive constant satisfying $x_k \pm \varepsilon \in \mathbb{U}$. Then $\boldsymbol{d} = -\varepsilon\,\mathrm{sgn}(f_{x_k})\boldsymbol{u}_k$ is a feasible direction at $\boldsymbol{x}$, where $\boldsymbol{u}_k$ denotes the $k$-th unit vector of $\mathbb{R}^n$. Since the diagonal of $\boldsymbol{Q}$ is zero, the first term of (3.11) is

$$\frac{1}{2}\boldsymbol{d}^\mathsf{T}\boldsymbol{Q}\boldsymbol{d} = q_{kk}\,d_k^2 = 0.$$

Substituting $\boldsymbol{d}$ into the second term of (3.11) yields

$$\boldsymbol{d}^\mathsf{T}\boldsymbol{f}_{\boldsymbol{x}} = -\varepsilon\,\mathrm{sgn}(f_{x_k})f_{x_k} = -\varepsilon\,|f_{x_k}| < 0.$$

Putting it all together we obtain $f(\boldsymbol{x}) - f(\boldsymbol{x}+\boldsymbol{d}) < 0$. Hence, $\boldsymbol{x}$ is not a minimum point.

*Case 2:* Assume that $f_{x_k} = 0$ for all $k \in [1:n]$ with $0 < x_k < 1$. The partial

derivative $f_{x_k}$ of $f$ at $x_k$ is of the form

$$f_{x_k} = \underbrace{\sum_{j=1}^{n} q_{kj} x_j}_{=S} + c_k = 0.$$

Since $c_k > 0$ by definition, there is at least one negative term $q_{kl}x_l$ in the sum $S$ for some $l \in [1:n]$. Hence, we have $x_l > 0$ and $q_{kl} = -\mu$. Separating the sum $S$ into the special term $j = l$ and all the rest yields

$$f_{x_k} = -\mu x_l + \sum_{\substack{j=1 \\ j \neq l}}^{n} q_{kj} x_j + c_k = 0.$$

From

$$\mu > \Delta_{\mathrm{w}}(X) + \|\boldsymbol{c}\|_\infty \geq \sum_{\substack{j=1 \\ j \neq l}}^{n} q_{kj} x_j + c_k,$$

it follows that $x_l < 1$; otherwise $f_{x_k}$ would be negative. Since $x_l \in ]0,1[$, we have $f_{x_l} = 0$ by assumption. Thus, there exists a constant $\varepsilon > 0$ such that $x_k + \varepsilon \in [0,1]$ and $x_l - \varepsilon \in [0,1]$. Obviously, $\boldsymbol{d} = \varepsilon(\boldsymbol{u}_k - \boldsymbol{u}_l)$ is a feasible direction at $\boldsymbol{x}$. Substituting $\boldsymbol{d}$ into (3.11) yields

$$f(\boldsymbol{x}) - f(\boldsymbol{x} + \boldsymbol{d}) = q_{kl} d_k d_l + d_k f_{x_k} + d_l f_{x_l}.$$

Since $f_{x_k} = f_{x_l} = 0$, the term $d_k f_{x_k} + d_l f_{x_l}$ vanishes. With $q_{kl} = -\mu$ and $d_k d_l = -\varepsilon^2$ we arrive at

$$f(\boldsymbol{x}) - f(\boldsymbol{x} + \boldsymbol{d}) = \mu\varepsilon^2 > 0.$$

Thus, $\boldsymbol{x}$ is not a minimum point.

Combining cases 1 and 2 proves that $\boldsymbol{x}$ is not a minimum point. Consequently, if a minimum point exists, then it is an extreme point. ■

A direct implication of Lemma 3.3 results in the following corollary.

***Corollary 3.2***
*All minima of the nonlinear program (3.9) are strict.*

The next result shows that the characteristic vectors of maximal cliques are local minimizers of $f$.

***Lemma 3.4***
*Consider the nonlinear program (3.9). Let $C$ be a maximal weight clique of $X$. Then the characteristic vector $\mathbf{1}_C$ of $C$ is a local minimum of $f$.*

***Proof*** Let $\boldsymbol{f_x} = (f_{x_1}, \ldots, f_{x_n})^\mathsf{T}$ be the gradient of $f$ at $\boldsymbol{x} = \mathbf{1}_C$. To prove that $\boldsymbol{x} = (x_1, \ldots, x_n)^\mathsf{T}$ is a local minimum of $f$, it is sufficient to show that $\boldsymbol{f_x^\mathsf{T} d} \geq 0$ for each feasible direction $\boldsymbol{d}$. We first show the assertion for feasible directions $\boldsymbol{d} = \varepsilon_k \boldsymbol{u}_k$ parallel to the axes of the hypercube with suitable chosen $\varepsilon_k \in [-1, +1] \setminus \{0\}$. We have

$$\boldsymbol{f_x^\mathsf{T} d} = \varepsilon_k f_{x_k},$$

where

$$f_{x_k} = -\sum_{j \in C} q_{kj} x_j - c_k.$$

We consider two cases:

*Case 1:* Assume that $x_k = 1$. A feasible direction along $\boldsymbol{u}_k$ is of the form $\boldsymbol{d} = \varepsilon_k \boldsymbol{u}_k$ with $\varepsilon_k < 0$. Since $C$ is a clique containing vertex $k$, we have $a_{kj} = 0$ and therefore,

$$q_{kj} = o_{kj} - \mu a_{kj} = o_{kj} \geq 0$$

for all $j \in C$. Together with $c_k > 0$, this yields $f_{x_k} < 0$. Since $\varepsilon < 0$, we arrive at $\boldsymbol{f_x^\mathsf{T} d} = \varepsilon_k f_{x_k} > 0$.

*Case 2:* Assume that $x_k = 0$. In this case, a feasible direction $\boldsymbol{d}$ along $\boldsymbol{u}_k$ is of the form $\boldsymbol{d} = \varepsilon_k \boldsymbol{u}_k$ with $\varepsilon_k > 0$. By assumption, $C$ is maximal. Hence, $C \cup \{k\}$ is not a clique. Consequently, there is a vertex $l \in C$ with $(k, l) \in \overline{E}$ giving

$$q_{kl} = o_{kl} - \mu a_{kl} = -\mu < 0.$$

From $l \in C$ follows $x_l = 1$. Hence, we have

$$f_{x_k} = -\sum_{j \in C} q_{kj} x_j - c_k = \mu x_l - \sum_{\substack{j \in C \\ j \neq l}} q_{kj} x_j - c_k \geq \mu - (\Delta_\omega(X) + \|\boldsymbol{c}\|_\infty) > 0.$$

From $\varepsilon_k > 0$ and $f_{x_k} > 0$ follows $\boldsymbol{f_x^\mathsf{T} d} = \varepsilon_k f_{x_k} > 0$.

It remains to show that $\boldsymbol{f_x^\mathsf{T} d} \geq 0$ for any feasible direction. A feasible direction can be written as a linear combination of the standard basis $\boldsymbol{d} = \varepsilon_1 \boldsymbol{u}_1 + \cdots + \varepsilon_n \boldsymbol{u}_n$, where $\varepsilon_1, \ldots, \varepsilon_n$ are appropriately chosen so that $\boldsymbol{x} + \boldsymbol{d} \in \mathbb{U}^n$. Then we have

$$\boldsymbol{f_x^\mathsf{T} d} = \sum_{i=1}^{n} \varepsilon_i f_{x_i} = \sum_{i \in C} \varepsilon_i f_{x_i} + \sum_{j \notin C} \varepsilon_j f_{x_j}.$$

From our distinction of cases, it follows that $\varepsilon_i f_{x_i}$ is positive for all $i \in V$. Hence, we have $\boldsymbol{f_x^\mathsf{T} d} > 0$. ■

Next we establish a one-to-one correspondence between the maximal cliques of $X$ and the local minima of $f$.

***Lemma 3.5***
*Consider the nonlinear program (3.9). The mapping*

$$\phi : \mathcal{C}_X^{\times} \to \mathcal{S}_f^{\times}, \quad C \mapsto \boldsymbol{x}_C$$

*is bijective.*

***Proof*** Let $\mathcal{P}(V)$ denote the set of all subsets of $V$. The mapping

$$\psi : \mathcal{P}(V) \to \mathbb{B}^n, \quad U \mapsto \mathbf{1}_U$$

associates each subset $U$ of $V$ with its characteristics vector $\mathbf{1}_U$. It is easy to show that $\psi$ is well-defined and bijective. Hence, the extreme points of $\mathbb{U}^n$ can be identified with the subsets of $V$. To prove that the restriction $\psi|_{\mathcal{C}_X^{\times}} = \phi$ is bijective is equivalent to showing that $C \in \mathcal{C}_X^{\times} \Leftrightarrow \mathbf{1}_C \in \mathcal{S}_f^{\times}$.

$\Rightarrow$**:** Follows from Lemma 3.4.

$\Leftarrow$**:** Let $\boldsymbol{x} \in \mathcal{S}_f^{\times}$ be a local minimum point. Then there is a subset $C$ with $\mathbf{1}_C = \boldsymbol{x}$. Assume that $C$ is not a maximal clique. Let us consider the following two cases.

*Case 1:* Assume that $C$ is a clique, but not maximal. Then there is a vertex $k \in V$ such that $C \cup \{k\}$ is a clique. Moreover, $\boldsymbol{d} = \varepsilon \boldsymbol{u}_k$ with $0 < \varepsilon \leq 1$ is a feasible direction such that

$$\boldsymbol{f}_{\boldsymbol{x}}^{\mathsf{T}} \boldsymbol{d} = \varepsilon f_{x_k} < 0.$$

*Case 2:* Assume that $C$ is not a clique. Then $|C| > 1$, because a singleton $C$ is always a clique. Consequently, there are vertices $k, l \in V$ such that $(k, l) \in \overline{E}$. Let $\boldsymbol{d} = \varepsilon \boldsymbol{u}_k$ with $-1 \leq \varepsilon < 0$. Then $\boldsymbol{d}$ is a feasible direction at $\boldsymbol{x}$ such that

$$\boldsymbol{f}_{\boldsymbol{x}}^{\mathsf{T}} \boldsymbol{d} = \varepsilon f_{x_k} = \varepsilon \left( \mu x_l - \sum_{j \neq l} q_{kj} x_j - c_k \right) < 0.$$

Combining both cases proves that there is a direction at $\boldsymbol{x}$ along which $f$ decreases. This contradicts the assumption that $\boldsymbol{x}$ is a local minimum. Hence, $\boldsymbol{x}$ is a maximal clique. ■

***Proof of Theorem 3.2***
The first assertion follows from Lemma 3.5. The third assertion follows from Corollary 3.2.

Thus it remains to show the second assertion. Observe that maximizing

$$\omega(\mathbf{1}_C) = \frac{1}{2} \mathbf{1}_C^{\mathsf{T}} \boldsymbol{O} \mathbf{1}_C + \mathbf{1}_C^{\mathsf{T}} \boldsymbol{d}_{\boldsymbol{X}}$$

over all $C \in \mathcal{C}_X$ is equivalent to maximizing

$$\omega(\mathbf{1}_C) = \frac{1}{2} \mathbf{1}_C^{\mathsf{T}} \boldsymbol{O} \mathbf{1}_C - \frac{\mu}{2} \underbrace{\mathbf{1}_C^{\mathsf{T}} \boldsymbol{A} \mathbf{1}_C}_{=0} + \mathbf{1}_C^{\mathsf{T}} \boldsymbol{d}_{\boldsymbol{X}}$$

over all $C \in \mathcal{C}_X$. This in turn is equivalent to minimizing

$$f(\mathbf{1}_C) = -\frac{1}{2}\mathbf{1}_C^\top \boldsymbol{Q}\,\mathbf{1}_C - \mathbf{1}_C^\top \boldsymbol{c}$$

over all $C \in \mathcal{C}_X$, where $\boldsymbol{Q} = \boldsymbol{O} - \mu\boldsymbol{A}$ and $\boldsymbol{c} = \boldsymbol{d}_{\boldsymbol{X}}$. Together with Lemma 3.5, this proves the second assertion. ■

## 3.7 Conclusion

In this chapter, we showed that the $\mathfrak{p}$GMP is equivalent to the MWCP in a $\kappa$-association graph. This result is interesting for the following reasons:

- It provides a generic view of diverse graph matching problems, including probabilistic matching and error correcting graph matching.
- It provides a pure graph-theoretical view of the problem, for which generic solution methods can be adopted from the classical MCP and MVCP.
- It reveals a common misconception that the association graph framework is a special technique to solve special graph matching problems.

A necessary prerequisite to establish this key result is a formal extension of the standard MCP and MVCP to the MWCP.

The proposed necessary and sufficient condition of $\mathfrak{p}$-closedness simplifies proofs for equivalence between graph matching and clique search in a $\kappa$-association graph. Using the *Clique Formulation of Graph Matching Theorem*, proofs can now be reduced to show that the set of feasible morphisms is $\mathfrak{p}$-closed for some local property $\mathfrak{p}$.

To apply existing and develop new continuous optimization methods, we presented an equivalent continuous formulation of the MWCP. The next chapter is devoted to this issue.

# 4 Hopfield Clique Networks

In the last chapter, we showed that graph matching problems constrained over closed feasible regions are equivalent to the maximum weight clique problem. The present chapter proposes a new continuous Hopfield model for solving the maximum weight clique problem. Its distinguishing features are as follows. The proposed system

- provides a generic optimization-based solution to $\mathfrak{p}$-graph matching problems,
- ensures convergence to feasible solutions,
- requires no tuning of system parameters, and
- optimally adapts the system parameters to its current state.

The last feature allows us to interpret the model in terms of selective attention from the field of cognitive psychology. In experiments on the MWCP and on graph matching problems, we assess the performance of our novel approach.

## 4.1 Introduction

For any NP-complete combinatorial optimization problem, exact search algorithms, such as branch and bound or dynamic programming, have exponential computation time in the worst case. Because of this, exact approaches limit us to solve only moderately sized problem instances. Therefore, in practice, heuristic search algorithms are necessary to find satisfactory solutions within an acceptable time limit.

The combinatorial optimization problem considered here are graph matching problems in disguise of the MWCP. Since it is our primary concern to construct neural learning machines for attributed graphs within a pure connectionist framework, we require that a solution for the MWCP is (i) implemented by a neural network and (ii) sufficiently fast enough to process huge datasets of large graphs.

Several neural and related energy minimizing methods have been devised to solve the standard MCP [37, 99, 100, 102, 162, 167, 284, 378, 393]. Apparently, only two approaches that address the maximum vertex-weight clique problem (MVCP) are reported in the literature . The first approach by Ballard et al. [17] encodes the `MVCP` into a neural network architecture without presenting any experimental results. In addition, their approach lacks theoretical guarantees of convergence to a feasible solution. The second approach by Bomze et al. [37] expands the Motzkin-Strauss

formulation of the MCP [271] to the MVCP. To solve the expanded Motzkin-Strauss formulation, Bomze et al. applied the replicator equations from evolutionary game theory [144, 381]. It is, however, unclear how to further expand the Motzkin-Strauss formulation to the MWCP.

In disguise of the graph matching problem, the MWCP, in a more or less strict form, has been implicitly solved by [30, 230, 311, 345]. Previous approaches put the main emphasis on the graph matching problem. In [311], Schädler first recognizes that her graph matching problem corresponds to a MWCP without providing a rigorous mathematical formalization. In order to formulate a generic solver for graph matching problems, this chapter takes a different approach. We abstract from the graph matching problem and put the main emphasis on the MWCP.

In this chapter, we propose a Hopfield model for the MWCP, called *Attention Control System* (ACS). To ensure convergence to a feasible solution corresponding to a maximal weight clique, we derive bounds for the system parameters. The bounds are simple to compute and provide a basis for optimal parameter selection so that no tuning of system parameters is required. To speed up convergence without loss of solution quality, the dynamics of the ACS model optimally adapts its system parameters with respect to the subset of active units. Since this approach allows a metaphor of *selective attention*, we coined the model Attention Control System.

In the next section, we briefly review Hopfield models. Applications of Hopfield models and its variant to combinatorial optimization problems are discussed in Section 4.3. Section 4.4 establishes the Hopfield Clique Theorem. The Hopfield Clique Theorem provides bounds for the system parameters to ensure convergence to a fixed point. The theoretical results give rise to a basic form of a Hopfield clique model and its improved variant, the Attention Control System. Both algorithms are described in Section 4.5. Section 4.6 presents experiments. Finally, we conclude with a discussion in Section 4.7.

## 4.2 Hopfield Models

The aim of this section is to introduce Hopfield models and their variants from an engineering perspective. Since we are primarily interested in numerical solutions for combinatorial optimization problems, only discrete-time models are considered. Detailed elaborations on Hopfield networks that discuss aspects and properties not mentioned here can be found in [40, 138, 166]. For a comprehensive analysis of the dynamical behavior of Hopfield models see [11, 119, 141]. An extensive overview of biological neural networks is provided in [198].

In analogy to physical systems, Hopfield [147, 148, 149, 150] proposed networks of simple interacting artificial neurons that exhibit useful *collective emergent phenomena* such as content addressable memories or solutions to combinatorial optimization problems. As any model of an artificial neural network, Hopfield networks are specified by three basic entities:

- models of neurons (*processing units*),
- models of synaptic interconnections (*topology*), and
- models of neural interactions (*dynamics*).[1]

In the following, we examine these entities in detail.

### 4.2.1 Processing Units: Models of a Neuron

A simplistic model of a biological neuron is a *processing unit* or, simply, a *unit* that constitutes a generic building block of any artificial neural network. We identify the following components of a neuronal model:

- *Processing unit.* A processing unit represents the cell body (*soma*) of a biological neuron. At time $t$, each unit $i$ has an *internal state* or *activation* $u_i(t) \in \mathbb{R}$ and an *external state* or *output* $x_i(t) \in \mathbb{R}$.
- *Weights.* Each unit $i$ receives input signals $x_j(t)$ from other units $j$ via its connecting links or *synapses*. The weight $w_{ij}$ characterizes the strength of a synapse connecting pre-synaptic unit $j$ with post-synaptic unit $i$. Weights can be positive or negative depending on whether the corresponding units interact in an *excitatory* or *inhibitory* manner.
- *Activation function.* An activation function updates the internal state of unit $i$ according to

$$u_i(t+1) = (1-d_i)u_i(t) + \sum_j w_{ij}x_j(t) + h_i,$$

where $d_i \in [0,1]$ denotes the *decay-term* of unit $i$, and $h_i$ is an *external input*.
- *Transfer function.* The output $x_i(t)$ of unit $i$ is related to its activation $u_i(t)$ by a non-decreasing transfer function

$$x_i(t) = g_i\big(u_i(t)\big).$$

For the sake of simplicity, we sometimes omit the time index $t$ and write $u_i$ and $x_i$ instead of $u_i(t)$ and $x_i(t)$.

#### *Transfer Functions*

A transfer function characterizes the output behavior of a unit and therefore the overall dynamics of a neural network. Depending on the choice of transfer function, we divide units into three categories:

1. A treatment of feed-forward and recurrent neural networks from a dynamical systems' point of view is presented in [119].

- *discrete units*,
- *analog units*, and
- *stochastic units*.

Discrete units can only assume a finite number of possible output values. Analog units have a continuous-valued response. The output values of stochastic units are discrete values drawn from a probability distribution. Though stochastic units are also discrete-valued, the term *discrete unit* only refers to units for which the transfer function is discrete-valued *and* deterministic.

In the following, we list some examples of typical transfer functions.

- *Discrete units*
  - *Threshold function.* The threshold function is defined by

$$\Theta(u_i) = \begin{cases} 1 & : \text{ if } u_i > 0 \\ 0 & : \text{ if } u_i \leq 0 \end{cases} .$$

    Units with threshold function are commonly referred to as *McCulloch-Pitts units*, in recognition of the pioneering work by McCulloch and Pitts [253].

- *Analog units*
  - *Logistic function.* A logistic or Fermi function is of the form

$$g_\beta(u_i) = \frac{1}{1 + \exp(-\beta u_i)},$$

    where the parameter $\beta > 0$ denotes the *gain*. The logistic function assumes values from $]0, 1[$ and the gain controls the slope of the transfer function. In the high-gain limit ($\beta \to \infty$), the logistic function $g_\beta$ approximates the threshold function $\Theta$.
  - *Limiter function.* The piecewise-linear limiter function is defined by

$$[u_i]_\beta = \begin{cases} 1 & : \text{ if } \beta u_i > 1 \\ \beta u_i & : \text{ if } 0 \leq \beta u_i \leq 1 \\ 0 & : \text{ if } u_i < 0 \end{cases} ,$$

    where $\beta > 0$ is the gain. The limiter function assumes its lower and upper saturation point at 0 and 1, respectively.

- *Stochastic units*
  - *Probabilistic threshold function.* The probabilistic threshold function is of the form

$$\Theta_P(u_i) = \begin{cases} 1 & : \text{ with probability } P(u_i) \\ 0 & : \text{ with probability } 1 - P(u_i) \end{cases} ,$$

    where $P(u_i)$ is the *activation probability*. A standard choice of $P(u_i)$ is given

by the logistic function

$$P(u_i) = g_{\frac{1}{T}}(u_i) = \frac{1}{1 + \exp\Big(-\frac{u_i}{T}\Big)},$$

where $T$ is a *pseudo-temperature* or, simply, *temperature*.

### 4.2.2 Hopfield Topologies: Models of Synaptic Interconnections

Processing units as generic building blocks can be assembled into complex networks. We focus on recurrent neural networks that generalize Hopfield's original models [147, 148].

The topology of a general Hopfield network is composed of $n$ fully interconnected units of the same type such that

1. $w_{ij} = w_{ji}$
2. $w_{ii} = 0$
3. $h_i \geq 0$

for all $i, j \in [1 : n]$. The first property requires that the weights are *symmetric*, the second that there is no self-interaction, and the third that the externally applied inputs are nonnegative. Analysis of Hopfield networks with weaker restrictions on the weights and external inputs can be found in [42, 43, 124].

### 4.2.3 Hopfield Dynamics: Models of Neural Interactions

Given an initial activation, a Hopfield model repeatedly updates its internal and external states according to an *update rule* of the form

$$u_i(t+1) = (1 - d_i)u_i(t) + \sum_j w_{ij}x_j(t) + h_i \tag{4.1a}$$

$$x_i(t+1) = (1 - \lambda_i)x_i(t) + \lambda_i g_i\big(u_i(t+1)\big), \tag{4.1b}$$

where $\lambda_i \in ]0, 1]$ is the *step size* parameter of unit $i$.

There are different ways of implementing the update. Here, we consider the following *sequential* and *parallel* mode:

- *Sequential Mode.* At each time step, a randomly selected unit updates its internal and external state.
- *Parallel Mode.* At each time step, all units simultaneously update their internal and external state.

Examples of classical discrete-time Hopfield models are shown in Table 4.1. Note that the discrete-time Hopfield′84 model is derived from Hopfield's original continuous-time analog model [148] using the Euler discretization.

To study the long-term behavior of interacting units that evolve under update

| Model | Update mode | Unit | $g(u_i)$ | $d_i$ | $\lambda_i$ | Reference |
|---|---|---|---|---|---|---|
| Hopfield'82 model | sequential | discrete | $\Theta(u_i)$ | 1 | 1 | [147] |
| Hopfield'84 model | sequential | analog | $g_\beta(u_i)$ | 1 | ]0,1] | [148, 379] |
| Little model | parallel | discrete | $\Theta(u_i)$ | 1 | 1 | [233] |
| Iterated-map model | parallel | analog | $g_\beta(u_i)$ | 1 | 1 | [51], [252] |

**Table 4.1** Four models of discrete-time Hopfield dynamics.

rule (4.1), it is useful to view Hopfield models as dynamical systems. We informally present some basic concepts of neurodynamics necessary for further understanding and refer to [138, 143] for mathematical treatment.

The *state* of a Hopfield model at time step $t$ is defined by the vector $\boldsymbol{x}_t$ composed of all outputs $x_i(t)$. The set of all possible states a Hopfield model can assume is referred to as its *state space*. Starting from an initial activation, update rule (4.1) generates a sequence of states $(\boldsymbol{x}_t)_{t\geq 0}$ called the *trajectory* of the dynamical system, given the initial activation $\boldsymbol{x}_0$.

The relevant question here is what kind of dynamical behavior do we expect from our system? To discuss this, we categorize different types of trajectories according to their convergence properties. We distinguish between convergence to *fixed points*, convergence to *limit cycles*, and *chaotic behavior*.

1. *Fixed Points*. A fixed point is a state that repeats itself during the dynamical process. A fixed point $\boldsymbol{\xi}$ is *stable* if nearby trajectories remain close to $\boldsymbol{\xi}$. Otherwise a fixed point is *unstable*.
2. *Limit Cycle*. A limit cycle is a finite sequence of at least two states that periodically repeat themselves under the dynamical process.
3. *Chaos*. Chaos refers to trajectories that evolve in the state space in an uncorrelated and irregular fashion.

From an optimization point of view, we want convergence to stable fixed points, which encode feasible solutions of the underlying problem. Fixed points, which represent infeasible solutions, limit cycles, or chaotic behavior are all undesirable.

A useful mathematical tool to determine global stability of dynamical systems is *Liapunov's direct method*. The basic idea is to construct a scalar-valued *Liapunov* or *energy function* over the state space of the dynamical system, which is bounded below and non-increasing along all trajectories. If such an energy function exists and the system is *well-behaved*, then the long-term behavior of a Hopfield model can be characterized in a qualitative way.[2]

What makes Liapunov's direct method so useful is that it enables us to analyze the long-term behavior without solving the state space equations of the system. The main problem with this method is that constructive principles to find an energy

2. A rigorous mathematical introduction to Liapunov functions can be found in [224, 305].

function are unknown.

As an example, consider the sequential discrete Hopfield′82 model. Given an initial activation, the dynamics of that model minimizes the energy function

$$E(\boldsymbol{x}_t) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i(t)x_j(t) - \sum_{i=1}^{n} h_i x_i(t) \tag{4.2}$$

until convergence to a fixed point corresponding to a local minimum point of $E$.

Table 4.2 summarizes the main convergence characteristics of the models presented in Table 4.1. From the table we see that sequential updating schemes lead to stable fixed points, whereas trajectories evolving in parallel mode may converge to fixed points and period-two oscillations. At first glance, sequential dynamics seems to be more appropriate for application to optimization problems than parallel dynamics, because of its *well-behaved* convergence properties. But in numerical simulations, in particular on parallel computers, sequential models are much slower than their parallel counterparts. Thus, the design of Hopfield models is a trade-off between stability and speed.

| Model | Fixed points | Limit cycles | Chaos | Reference |
|---|---|---|---|---|
| Hopfield′82 model | + | – | – | [147, 141] |
| Hopfield′84 model | + | – | – | [379] |
| Little's model | + | + | – | [141] |
| Iterated-map network | + | + | – | [141, 252] |

**Table 4.2** Main characteristics of discrete-time Hopfield models, provided that the weights are symmetric, self-interactions are zero, and external inputs are nonnegative. Limit cycles occurring in parallel models have length two. References indicate where proofs can be found.

## 4.3 Hopfield Networks for Combinatorial Optimization Problems

In this section, we describe how to apply Hopfield networks to combinatorial optimization problems (COP). In addition, we discuss some improved variants of Hopfield's original model.

From now on, we use the more concise vector notation to describe the Hopfield dynamics, rather than the classical scalar notation. We use the following notations:

| | |
|---|---|
| $\boldsymbol{u}_t$ | activation vector $\boldsymbol{u}_t = \big(u_1(t), \ldots, u_n(t)\big)^\mathsf{T}$ |
| $\boldsymbol{x}_t$ | state vector $\boldsymbol{x}_t = \big(x_1(t), \ldots, x_n(t)\big)^\mathsf{T}$ |
| $\boldsymbol{W}$ | symmetric weight matrix $\boldsymbol{W} = (w_{ij})$ with zero diagonal |
| $\boldsymbol{w}_{i:}$ | $i$-th row $\boldsymbol{w}_{i:} = (w_{i1}, \ldots, w_{in})^\mathsf{T}$ of $\boldsymbol{W}$ |
| $\boldsymbol{h}$ | external input vector $\boldsymbol{h} = (h_1, \ldots, h_n)^\mathsf{T}$ |
| $g(\boldsymbol{u})$ | component-wise transfer function $g(\boldsymbol{u}) = \big(g(u_1), \ldots, g(u_n)\big)^\mathsf{T}$ |

The Hopfield & Tank [149] approach to approximately solve COPs maps the objective function of the optimization problem to an equivalent energy function of the network. The constraints of the problem are included in the energy function as penalty terms, such that the global minima of the energy function correspond to the solutions of the COP.
Assume that we are given a COP defined by the following quadratic integer program

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} - \boldsymbol{c}^\mathsf{T}\boldsymbol{x} \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & && \boldsymbol{x} \in \mathbb{B}^n, \end{aligned} \tag{4.3}$$

where $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ is symmetric, $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$. The Hopfield & Tank (HT) energy function is of the form

$$E(\boldsymbol{x}) = f(\boldsymbol{x}) - \sum_{i=1}^{m} \mu_i(\boldsymbol{a}_{i:}^\mathsf{T}\boldsymbol{x} - b_i)^2, \tag{4.4}$$

where $\boldsymbol{a}_{i:}$ is the column vector representing the $i$-th row of $\boldsymbol{A}$, and $\mu_i \geq 0$ is a penalty parameter that is chosen to reflect the relative importance of the $i$-th constraint $\boldsymbol{a}_{i:}^\mathsf{T}\boldsymbol{x} = b_i$. It is easy to verify that a constrained minimum of $f$ also minimizes the energy function $E$.

To infer a standard energy function in terms of weights and external inputs as in (4.2), we expand the HT energy function (4.4). Then weights $w_{ij}$ are the coefficients of quadratic terms $x_i x_j$, and external inputs $h_i$ are the coefficients of linear terms $x_i$. Constant terms can be neglected, because they do not effect minimization. Thus, we arrive at an energy function of the form

$$E(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}_t^\mathsf{T}\boldsymbol{W}\boldsymbol{x} - \boldsymbol{h}^\mathsf{T}\boldsymbol{x}, \tag{4.5}$$

which corresponds to the energy function (4.2) of the Hopfield'82 model. The main challenge is how to minimize $E$ in order to find a satisfactory solution within an acceptable period of time. The simplest and most straightforward possibility is to apply the Hopfield'82 dynamics, because it ensures gradient descent of the energy function until convergence to a fixed point. As an example, Algorithm 1 describes

the Hopfield'82 method to solve the optimization problem (4.3).

**Algorithm 1** (Hopfield'82 Algorithm)

**Input:**
$\boldsymbol{W}$ – weights
$\boldsymbol{h}$ – external inputs
**Initialization:**
set activation $\boldsymbol{u}$
**Procedure:**
**repeat**
randomly select a unit $i$
$u_i = \boldsymbol{w}_{i:}^{\mathsf{T}} \boldsymbol{x} + h_i$
$x_i = \Theta(u_i)$
**until** convergence to a fixed point
**Output:** $\boldsymbol{x}$

The Hopfield'82 Algorithm and its variations discussed in the previous section performed unsatisfactory or even failed when applied to problems of combinatorial optimization [385]. The main defects of the general Hopfield model are:

- infeasible solutions caused by spurious local minima,
- feasible solutions of poor quality,
- oscillations in parallel mode, and
- slow convergence.

These shortcomings have inspired modifications of the general Hopfield model. Perhaps the most successful methods are based on principles of statistical mechanics and their deterministic approximations. In the remainder of this section, we first discuss three types of stochastic machines, the Boltzmann, Cauchy, and Gaussian machines. Next, we describe mean-field and Potts mean-field annealing, two efficient approximations of stochastic machines. As final methods, we consider chaotic networks and exterior-point penalty networks. We conclude this section with some final remarks.

### 4.3.1 Boltzmann Machine

The *Boltzmann machine* proposed by Ackley et al. [3] is apparently one of the first neural models inspired by principles from statistical mechanics. The topology of a Boltzmann machine complies with a stochastic Hopfield model. The activation probabilities $P(u_i)$ are of the form

$$P(u_i) = g_{\frac{1}{T}}(u_i) = g_{\frac{1}{T}}\big(\boldsymbol{w}_{i:}^{\mathsf{T}} \boldsymbol{x} + h_i\big) \,.$$

To minimize the energy $E$, the Boltzmann machine generates a controlled sequence of local updates (*balancing*) combined with a reduction of the global temperature $T$ (*annealing*).

*Balancing:* Suppose that $\boldsymbol{x}$ is the current state of the Boltzmann machine. A local update step proceeds as follows:

1. Randomly select a unit $i$.
2. Apply the following update rule
$$u_i = \boldsymbol{w}_{i:}^{\mathsf{T}}\boldsymbol{x} + h_i$$
$$x_i = \Theta_P(u_i).$$

This process is repeated until convergence to *thermal equilibrium*. A fundamental result from statistical mechanics shows that in thermal equilibrium, the probability distribution of the states $\boldsymbol{x} \in \mathbb{B}^n$ obeys the *Boltzmann-Gibbs distribution*

$$P(\boldsymbol{x}) = \frac{1}{Z} \exp\left(\frac{-E(\boldsymbol{x})}{T}\right), \tag{4.6}$$

where $Z$ is a normalizing factor.

*Annealing:* Decrease (anneal) the temperature $T$. Boltzmann machines use a logarithmic schedule of the form

$$T = T_k = \frac{T_0}{\log(k)},$$

where $T_0$ is the initial temperature and $k$ is the annealing-time index.

Starting with a large initial value $T = T_0$, the Boltzmann machine iteratively performs balancing and annealing until no increase of the energy is accepted for $K$ consecutive decrements of the temperature $T$. The final *frozen* state is then returned as solution of the COP.

For $T > 0$, stochastic units assign any neighboring state a nonzero probability to be accepted. This mechanism enables the machine to escape from local minima. From the Boltzmann-Gibbs distribution, it follows that high temperatures reduce energy differences among the states. In this phase, the system performs coarse exploration of the state space. As the temperature is lowered, the probabilities concentrate on low-energy states. Consequently, the search exploits local terrain of low energy.

If annealing of the temperature is governed by the logarithmic schedule, the Boltzmann machine will converge to the global minimum, provided that new candidate states are drawn from a Gaussian distribution [109]. The problem with this approach is its prohibitively slow convergence.

***Remark 4.1***

*The difference of the Boltzmann machine algorithm and* simulated annealing *[207] lies in the choice of the activation probability $P(u_i)$. But the difference is only minor, because both methods lead to the same stationary distribution and have the*

*same convergence properties [1].*

### 4.3.2 Cauchy Machines

The Cauchy Machine [348, 350] is an improved variant of the Boltzmann machine. The modifications are:

- Stochastic units with activation probability

$$P(u_i) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{u_i}{T}\right).$$

- Faster annealing schedule

$$T_k = \frac{T_0}{k}, \tag{4.7}$$

  where $T_0$ is the initial temperature and $k$ is the annealing-time index.

The random sampling obeys a Cauchy distribution rather than a Boltzmann distribution. Since a Cauchy distribution has a *fatter* tail than the Gaussian form of a Boltzmann distribution, not only local random walks as in Gaussian sampling are produced, but also global random leaps. This permits easier access to test local minima in the search for a global minimum. As a consequence, the Cauchy machine can apply an exponentially faster annealing schedule than the Boltzmann machine to statistically find a global minimum. Convergence to a global optimum is shown in [348] for an annealing schedule no faster than (4.7).

### 4.3.3 Gaussian Machines

Gaussian Machines [7] are analog Hopfield models that update their state according to the following dynamical rule

$$\begin{aligned} u_i &= \boldsymbol{w}_{i:}^{\mathsf{T}} \boldsymbol{x} + h_i + \nu_i \\ x_i &= g_{\frac{1}{T}}(u_i), \end{aligned}$$

where $\nu_i$ is a stochastic term caused by random noise. This noise is Gaussian distributed with a zero mean and a variance controlled by the temperature parameter $T$. At high values of $T$ the noise has a large variance and therefore enforces coarse exploration of the state space. As $T$ decreases, variance becomes smaller and leads to a fine search in a local neighborhood.

By appropriate parameter selection of the Gaussian machine, we can derive the the Hopfield′82 model, the Hopfield′84 model, and the Boltzmann machine. As the Boltzmann machine, the Gaussian noise term only produces local random walks.

### 4.3.4 Mean-Field Annealing

Mean-field annealing [285] is a deterministic approximation of stochastic machines, which is computationally more efficient than the Boltzmann machine and its variations. The mean-field system replaces the binary outputs $x_i \in \mathbb{B}$ of the Boltzmann machine by their mean or average behavior $< x_i > \in \mathbb{U}$. Since $1 - g_\beta(u) = g_\beta(-u)$, we can derive from $P(u_i) = g_{\frac{1}{T}}(u_i)$ the following transition probabilities

$$P(x_i = 1) = g_{\frac{1}{T}}(u_i)$$
$$P(x_i = 0) = g_{\frac{1}{T}}(-u_i).$$

For a single unit $i$, the average state at temperature $T$ is

$$< x_i > = 1 \cdot P(x_i = 1) + 0 \cdot P(x_i = 0) = g_{\frac{1}{T}}(u_i).$$

Using this expression, we first equate the average $< x_i >$ in terms of the averages $< x_j >$ and then apply the Euler discretization to obtain the discrete-time mean-field dynamics

$$< x_i > = (1 - \lambda_i) < x_i > + \lambda_i g_{\frac{1}{T}} \left( \boldsymbol{w}_{i:}^{\mathsf{T}} < \boldsymbol{x} > + h_i \right), \tag{4.8}$$

where $\lambda_i > 0$. This update rule minimizes the energy of an analog Hopfield model

$$E_T(\boldsymbol{x}) = -\frac{1}{2} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{W} \boldsymbol{x} - \boldsymbol{h}^{\mathsf{T}} \boldsymbol{x} - T \cdot H(\boldsymbol{x}),$$

where $H$ is a convex term of the form

$$H(\boldsymbol{x}) = -\sum_{i=1}^{n} x_i \log(x_i) + (1 - x_i) \log(1 - x_i).$$

At high temperature $T$, the energy is dominated by the convex term $H(\boldsymbol{x})$ and has a unique global minimum. This convex term serves to smooth out local minima. By gradually lowering $T$, the influence of the convex term vanishes until the original energy at $T = 0$ is recovered.

Like the Boltzmann machine, the mean-field algorithm performs balancing and annealing. The stochastic dynamics of the Boltzmann machine is replaced by a sequential or parallel deterministic approximation (4.8). Given a temperature $T$, balancing aims at approximating the Boltzmann-Gibbs distribution. In the limit, lowering the temperature $T \to 0$ gradually transforms $g_{1/T}$ to a threshold function $\Theta$ and enforces the trajectory into a corner of the hypercube.

We present an example illustrating the behavior of the mean-field approach. This example will turn out to be useful when discussing the limitations of mean-field annealing on the MWCP.

***Example 4.2***
Figure 4.1 illustrates the mean-field procedure and shows a potential problem when using deterministic annealing.

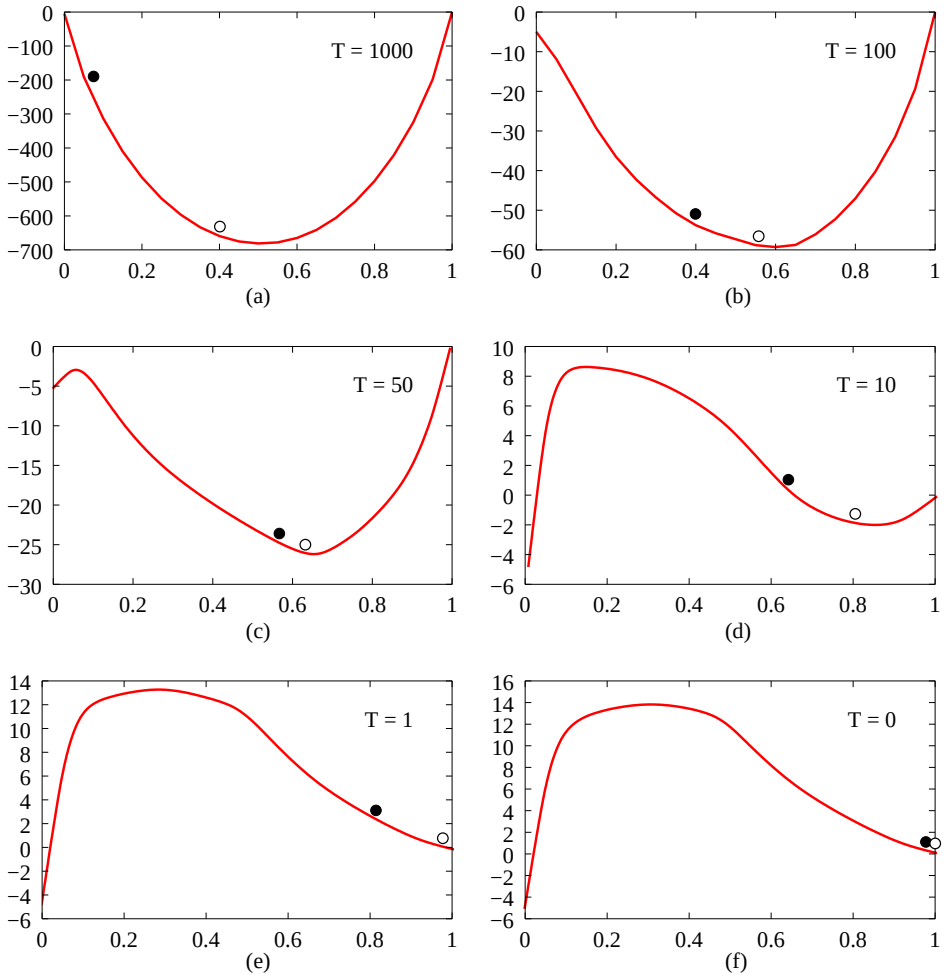



**Figure 4.1** Energy function of mean-field annealing for different values of the temperature $T$. The original energy function to be minimized is shown in (f). The mean-field procedure solves a sequence of energy functions ($E_T$). For each $T$, filled-in circles indicate initial states, and outlined circles final states. Although the global minimum of $E$ is at $x_g = 0$, the mean-field procedure will converge to $x_l = 1$.

The original energy function $E = E_0$ to be minimized is depicted for $T = 0$. To minimize $E$, the mean-field procedure solves a sequence ($E_T$) of minimization problems, where $E_T$ is composed of the original energy $E$ and the convex term $-TH$. For the sake of presentation, we assume that the initial temperature $T = 1000$ is lowered in large steps to the values $T = 100, 50, 10, 1, 0$. At $T = 0$, we replace $g_{1/T}$ by $\Theta$. For each temperature, filled-in circles indicate the energy of the initial state, and outlined circles the energy of the final state of the system. Note that the final state of the system at a given temperature $T$ is usually close to a local minimum of $E_T$.

From the plots, we see that at high temperatures the convex term $-TH$ dominates the energy $E_T$. By lowering $T$, more and more structure of the original energy $E$ emerges.

The structure of $E$ turns out to be inconvenient for the mean-field algorithm. The energy $E$ has a global minimum at $x_g = 0$ and a local minimum at $x_l = 1$. The basin of $x_g$ is a narrow kloof, whereas the basin of $x_l$ is a wide valley dominating the energy functions $E_T$. The trajectories have no chance to escape to the basin of attraction of $x_g$, and therefore converge to the local minimum $x_l$.

### 4.3.5 Potts Mean-Field Annealing

For a number of combinatorial optimization problems, the corresponding energy function to be minimized can be written as

$$E(\boldsymbol{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{l=1}^{m} w_{ijkl}x_{ij}x_{kl} - \sum_{i=1}^{n}\sum_{j=1}^{m} h_{ij}x_{ij},$$

where feasible solutions satisfy the constraints

$$\sum_{j=1}^{m} x_{ij} = 1 \tag{4.9}$$

for all $i \in [1:n]$.

From an algorithmic point of view, the Potts system and the standard mean-field method differ only in the way they update their state. The Potts equations are given by

$$u_{ij} = \sum_{k=1}^{n}\sum_{l=1}^{m} w_{ijkl}x_{kl} + h_{ij} \tag{4.10a}$$

$$x_{ij} = \frac{\exp\left(-\frac{u_{ij}}{T}\right)}{\sum_{r=1}^{m}\exp\left(-\frac{u_{ir}}{T}\right)}. \tag{4.10b}$$

As mean-field annealing, the Potts dynamics solves a controlled sequence $(E_T)$ of problems, where each subproblem $E_T$ involves the original energy function $E$ and a convex term $T \cdot H(\boldsymbol{x})$. Thus, everything said about mean-field annealing also holds for the Potts mean-field annealing.

The advantage of the Potts dynamics is that the search is performed in a relevant lower dimensional subspace. To see this, we collect all units $i1, \ldots, im$ in a row to a *Pott unit* with output state $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{im})$. The *softmax* function (4.10b) enforces the outputs of the Potts units to satisfy the constraints (4.9). Hence, the trajectories of each Potts unit evolve in an $(m-1)$-dimensional constraint plane instead of in the original $m$-dimensional unit hypercube. It has been shown empirically that systems using Potts units as *hard* implementation of (4.9) exhibit better performance than similar systems using *soft constraints* via penalty terms [285, 286].

***Remark 4.3***
*Note that the Potts dynamics is only applicable to graph matching problems constrained over subsets of morphisms. From a conceptual point of view, all* $\mathfrak{p}$*-graph matching problems considered in Section 3.5 fall into this realm. But for reasons of computational efficiency, it is advisable to reduce error correcting graph matching problems and certain graph matching problems in computer vision [30, 31] to problems constrained over a subset of relations instead of morphisms (see also Remark 3.3). Hence, there are no longer one-way constraints induced by the well-definedness of a morphism. For such problems, solutions like the Potts dynamics, which implement hard constraints via softmax become inapplicable.*

### 4.3.6 Chaotic Hopfield Models

Inspired by biological neurons with chaotic characteristics, Aihara et al. [4] suggested Hopfield models consisting of refractory units that exhibit chaotic behavior. Chaotic dynamics is useful for improving solution quality, but their unstable behavior is difficult to control in order to obtain a feasible solution. Chen & Aihara [56] combined the advantages of conventional and chaotic neurodynamics and introduced a temperature parameter together with the usual annealing process. At high temperatures, the transiently chaotic dynamics of the Chen-Aihara model searches for a global minimum. During annealing, the system becomes gradually stable and converges to a fixed point at low temperatures.

Chaotic Hopfield models minimize energy functions of the form

$$E(\boldsymbol{x}) = E_H(\boldsymbol{x}) + H_C(\boldsymbol{x}),$$

where $E_H$ is the original energy function of the associated non-chaotic Hopfield model and $H_C$ is an additional term that modifies the energy landscape. Chen and Aihara used the energy modifier

$$H_C = \frac{T}{2}\boldsymbol{x}^\mathsf{T}(\boldsymbol{x} - \boldsymbol{e}),$$

where $\boldsymbol{e}$ is the vector of all ones. The discrete-time dynamics that minimizes the modified energy $E$ can be derived by using the Euler discretization of the system

$$\frac{du_i}{dt} = -\frac{dE}{dt}.$$

As $T$ approaches 0, the term $H_C$ vanishes and the original objective function to be minimized is recovered. In contrast to the Boltzmann machine and its variations and approximations, the temperature $T$ declines monotonically at each time step.

### 4.3.7 Penalty Annealing

The HT energy function (4.5) includes several terms, each of which competes to be minimized. Appropriate choice of the penalty parameters $\mu_i$ is crucial to avoid

spurious local minima and to ensure feasible solutions. Parameter tuning of several penalty parameters $\mu_i$ and their analysis is intricate. A more convenient formulation of the HT energy function is of the form

$$E(\boldsymbol{x}) = f(\boldsymbol{x}) + \mu E_P(\boldsymbol{x}), \tag{4.11}$$

where the term $E_P$ incorporates the constraints given by $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ of problem (4.3).

Aiyer [5] showed that a sufficiently large penalty parameter $\mu$ ensures validity of a solution. A large value for $\mu$, which is necessary to ensure feasible solutions, is impractical for the following reasons:

- The term $\mu E_P$ dominates the energy function $E$. Hence, more emphasis is placed on minimizing $E_P$ rather than the objective function $f(\boldsymbol{x})$ of the original COP.
- With $\mu$, the gradient of $E$ becomes large. Thus, the system evolves according to the Hopfield dynamics with a large step size, which is susceptible to unstable oscillations or premature termination to a feasible solution of poor quality.

To overcome these difficulties, Jagota [167] and Schädler [311] augmented Hopfield models with exterior-point penalty methods from nonlinear programming [239]. Starting with a low initial value $\mu_0$ of the penalty parameter, a penalty annealing method minimizes a finite sequence $(E_k)$ of problems

$$E_k(\boldsymbol{x}) = f(\boldsymbol{x}) + \mu_k E_P(\boldsymbol{x}),$$

where the sequence $(\mu_k)$ is strictly increasing. Each subproblem $E_k$ can be solved with an appropriate Hopfield dynamics.

### 4.3.8 Concluding Remarks

Since Hopfield & Tank [149] demonstrated that quadratic COP could be solved using Hopfield networks, several alternatives have been proposed to improve solution quality. Some approaches, including Boltzmann machines, their stochastic variants, and their deterministic approximations, are considered to be competitive with other meta-heuristics in terms of solution quality [336].

The main limitations of the systems described in this section are twofold: first, they require tuning of several problem dependent system parameters, and second, satisfactory results usually go hand in hand with long computation times. Both disadvantages constitute the starting point of a new technique for solving COPs with Hopfield networks. We present this technique in the next section.

## 4.4 The Hopfield Clique Theorem

This section is devoted to the presentation and mathematical analysis of a parallel analog Hopfield dynamics for the MWCP. Provided that the system parameters

satisfy some simple and relatively tight bounds, we prove that the proposed dynamics ensures convergence to fixed points. In conjunction with Theorem 3.2 on the continuous formulation of the MWCP, we conclude that the stable fixed points of the proposed dynamics are in one-to-one correspondence to the maximal weight cliques of the graph under consideration.

Let $X = (V, E, \boldsymbol{X})$ be a normalized graph from $\mathcal{G}_{\mathbb{U}}$. Recall from Section 3.6 that the continuous formulation of the MWCP is of the form

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^{\mathsf{T}}\boldsymbol{Q}\boldsymbol{x} - \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ &\text{subject to} && \boldsymbol{x} \in \mathbb{U}^n, \end{aligned} \tag{4.12}$$

where $\boldsymbol{Q} = \boldsymbol{X} - \boldsymbol{D}_{\boldsymbol{X}} - \mu\boldsymbol{A}$, $\mu$ is a penalty parameter, $\boldsymbol{A}$ the constraint matrix, and $\boldsymbol{c} = \boldsymbol{d}_{\boldsymbol{X}}$ is the diagonal of $\boldsymbol{X}$. We map the optimization problem to an energy function of the form

$$E(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{x} - \boldsymbol{h}^{\mathsf{T}}\boldsymbol{x}, \tag{4.13}$$

where $\boldsymbol{W} = \lambda\boldsymbol{Q}$ and $\boldsymbol{h} = \lambda\boldsymbol{c}$. The positive constant $\lambda$ controls the step size of the following parallel discrete-time dynamics

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t + \lambda\big(\boldsymbol{Q}\boldsymbol{x}_t + \boldsymbol{c}\big) \tag{4.14a}$$

$$\boldsymbol{x}_{t+1} = g\big(\boldsymbol{u}_{t+1}\big), \tag{4.14b}$$

where $g : \mathbb{R} \to [0, 1]$ is a non-decreasing transfer function with Lipschitz constant $L$. Examples of transfer function satisfying the requirements of (4.14) are the logistic function $g_\beta(\boldsymbol{u})$ with Lipschitz constant $L = \beta/4$ and the piecewise linear limiter function $[\boldsymbol{u}]_\beta$ with Lipschitz constant $L = \beta$.

Since $\lambda$ is positive, the objective $f$ of (4.12) and the energy function $E$ are equivalent. Hence, by Theorem 3.2, the problem of minimizing $E$ is equivalent to the MWCP of $X$. We show that the Hopfield dynamics (4.14) minimizes $E$ and converges to fixed points only, provided that the step size $\lambda$ is appropriately chosen.

***Theorem 4.1* (*Hopfield Clique Theorem*)**
*Consider an analog Hopfield model with parallel discrete-time dynamics (4.14). Let $T = 1/L$, where $L$ is the Lipschitz constant of the transfer function $g$. If*

$$\lambda < \frac{2T}{\delta(X) + \mu\,\Delta_{\mathrm{co}}(X)},$$

*then*

$$E(\boldsymbol{x}_{t+1}) \leq E(\boldsymbol{x}_t)$$

*for all $t \geq 0$. In addition, the dynamics (4.14) converges to stable fixed points only.*

Recall from Section 2.2.3 that $\delta(X)$ is the minimum degree of $X$ and $\Delta_{\mathrm{co}}(X)$ the

maximum co-degree of $X$. Note that the energy function (4.13) can have unstable fixed points. From the proof of Theorem 3.2, it follows that the set of unstable fixed points has Lebesgue-measure zero. Hence, we can force the trajectory to leave the unstable invariant set by imposing random noise to the activation vector of the network.

The remainder of this section proves Theorem 4.1.

### 4.4.1 Proof

First we introduce some notations.

***Notation 4.4***

| | | |
|---|---|---|
| $\boldsymbol{u}$ | $=$ | $\boldsymbol{u}_t$ |
| $\boldsymbol{u'}$ | $=$ | $\boldsymbol{u}_{t+1}$ |
| $\boldsymbol{x}$ | $=$ | $\boldsymbol{x}_t$ |
| $\boldsymbol{x'}$ | $=$ | $\boldsymbol{x}_{t+1}$ |
| $\boldsymbol{E_x}$ | $=$ | $-\boldsymbol{W}\boldsymbol{x}_t - \boldsymbol{h}$ |
| $\boldsymbol{d}$ | $=$ | $\boldsymbol{x}_{t+1} - \boldsymbol{x}_t = \boldsymbol{x'} - \boldsymbol{x}$ |

Next, we show two auxiliary results.

***Lemma 4.1***
*We have*

$$T \left\| d \right\|^2 \leq -\boldsymbol{E}_{\boldsymbol{x}}^{\mathsf{T}} \boldsymbol{d}$$

*for all* $T > 0$.

***Proof*** From $\boldsymbol{u'} = \boldsymbol{u} + \boldsymbol{W}\boldsymbol{x} + \boldsymbol{h}$ and $-\boldsymbol{E_x} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{h}$ follows $-\boldsymbol{E_x} = \boldsymbol{u'} - \boldsymbol{u}$. Hence

$$-\boldsymbol{E}_{\boldsymbol{x}}^{\mathsf{T}} \boldsymbol{d} = (\boldsymbol{u'} - \boldsymbol{u})^{\mathsf{T}} (\boldsymbol{x'} - \boldsymbol{x}) = \sum_{i=1}^{n} (u_i' - u_i)(x_i' - x_i).$$

Note that $x_i' - x_i = g(u_i') - g(u_i)$. Since $g$ is a non-decreasing function, the terms $(u_i' - u_i)(x_i' - x_i)$ are nonnegative. Therefore, we may write

$$(u_i' - u_i)(x_i' - x_i) = |u_i' - u_i| \, |x_i' - x_i|$$

for all $i \in V$. By assumption, $g$ satisfies the Lipschitz condition with Lipschitz constant $L$. This implies that

$$|x_i' - x_i| \leq L \, |u_i' - u_i|$$

for all $i \in V$. From $L = T^{-1}$ follows

$$T\,|x_i' - x_i| \leq |u_i' - u_i|$$

and therefore

$$T\|\boldsymbol{d}\|^2 = T\sum_{i=1}^{n} |x_i' - x_i|^2 \leq \sum_{i=1}^{n} |u_i' - u_i|\,|x_i' - x_i| = -\boldsymbol{E}_{\boldsymbol{x}}^{\mathsf{T}}\boldsymbol{d}.$$

■

***Lemma 4.2***
*Let $T > 0$. Then $\boldsymbol{W} + 2\,T\,\boldsymbol{I}_n$ is positive definite.*

***Proof*** It is sufficient to show that $\boldsymbol{W} + 2\,T\,\boldsymbol{I}_n$ is diagonal dominant, i.e.

$$\sum_{\substack{j=1\\ j\neq i}}^{n} |w_{ij}| < 2\,T - w_{ii}$$

for all $i \in V$. Since $\boldsymbol{W} = \lambda\boldsymbol{Q} = \lambda\big(\boldsymbol{X} - \boldsymbol{D}_{\boldsymbol{X}} - \mu\boldsymbol{A}\big)$, the absolute values of the entries of $\boldsymbol{W}$ are of the form

$$|w_{ij}| = \begin{cases} 0 & : \text{ if } i = j \\ \lambda x_{ij} & : \text{ if } (i,j) \in E \\ \lambda\mu & : \text{ if } (i,j) \in \overline{E} \end{cases}$$

for all $i, j \in V$. Hence, $\boldsymbol{d}_{\boldsymbol{W}} = \boldsymbol{0}$ and therefore the proof reduces to the statement

$$\sum_{\substack{j=1\\ j\neq i}}^{n} |w_{ij}| < 2\,T.$$

Substituting the expressions for $w_{ij}$ yields

$$\sum_{\substack{j=1\\ j\neq i}} |w_{ij}| = \lambda\left(\sum_{j\in N(i)} x_{ij} + \sum_{j\in\overline{N}(i)} \mu\right) = \lambda\big(\deg_{\mathrm{w}}(i) + \mu\deg_{\mathrm{co}}(i)\big)$$

for all $i \in V$. We define

$$\rho = \max_{i\in V}\ \big\{\deg_{\mathrm{w}}(i) + \mu\deg_{\mathrm{co}}(i)\big\}.$$

Since $\mu$ satisfies

$$\mu > \Delta_{\mathrm{w}}(X) + \|\boldsymbol{c}\|_\infty > \Delta_{\mathrm{w}}(X),$$

we have

$$\rho \geq \mu\,\Delta_{\mathrm{co}}(X) > \Delta_{\mathrm{w}}(X) + \mu\big(\Delta_{\mathrm{co}}(X) - 1\big).$$

Hence, the maximum $\rho$ can be attained only for vertices $i$ with maximal co-degree $\deg_{\mathrm{co}}(i) = \Delta_{\mathrm{co}}(X)$. Such vertices $i$ have minimum degree, that is $\deg(i) = \delta(X)$. Since all weights $x_{ij}$ are from $\mathbb{U}$, we obtain

$$\rho \leq \delta(X) + \mu\, \Delta_{\mathrm{co}}(X).$$

Using this bound gives

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} |w_{ij}| \leq \lambda\rho \leq \lambda\big(\delta(X) + \mu\, \Delta_{\mathrm{co}}(X)\big)$$

for all $i \in V$. Finally, from

$$0 < \lambda < \frac{2\,T}{\delta(X) + \mu\, \Delta_{\mathrm{co}}(X)}$$

follows

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} |w_{ij}| < 2\,T$$

for all $i \in V$. This shows that $\boldsymbol{W} + 2\,T\boldsymbol{I}_n$ is diagonal dominant. ■

**Proof of Theorem 4.1:** We have

$$\begin{aligned} E(\boldsymbol{x'}) - E(\boldsymbol{x}) &= -\frac{1}{2}\boldsymbol{x'}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{x'} - \boldsymbol{h}^{\mathsf{T}}\boldsymbol{x'} + \frac{1}{2}\boldsymbol{x}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{x} + \boldsymbol{h}^{\mathsf{T}}\boldsymbol{x} \\ &= -\frac{1}{2}\left(\boldsymbol{x'}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{x'} - \boldsymbol{x}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{x}\right) - \boldsymbol{h}^{\mathsf{T}}\boldsymbol{d} \\ &= -\frac{1}{2}\left(\boldsymbol{d}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{d} + 2\boldsymbol{d}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{x}\right) - \boldsymbol{h}^{\mathsf{T}}\boldsymbol{d} \\ &= -\frac{1}{2}\boldsymbol{d}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{d} - \boldsymbol{d}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{x} - \boldsymbol{h}^{\mathsf{T}}\boldsymbol{d} \\ &= -\frac{1}{2}\boldsymbol{d}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{d} + \boldsymbol{E}_{\boldsymbol{x}}^{\mathsf{T}}\boldsymbol{d}. \end{aligned}$$

Using the inequality $-\boldsymbol{E}_{\boldsymbol{x}}^{\mathsf{T}}\boldsymbol{d} \geq T\|\boldsymbol{d}\|^2$ proven in Lemma 4.1, we obtain

$$E(\boldsymbol{x'}) - E(\boldsymbol{x}) \leq -\frac{1}{2}\boldsymbol{d}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{d} - T\|\boldsymbol{d}\|^2 = -\frac{1}{2}\boldsymbol{d}^{\mathsf{T}}\Big(\boldsymbol{W} + 2T\boldsymbol{I}_n\Big)\boldsymbol{d}.$$

The last equation follows from $\|\boldsymbol{d}\|^2 = \boldsymbol{d}^{\mathsf{T}}\boldsymbol{d} = \boldsymbol{d}^{\mathsf{T}}\boldsymbol{I}_n\boldsymbol{d}$. According to Lemma 4.2, the matrix $\boldsymbol{W} + 2T\boldsymbol{I}_n$ is positive definite. Hence, we have $E(\boldsymbol{x'}) - E(\boldsymbol{x}) \leq 0$ and $E(\boldsymbol{x'}) - E(\boldsymbol{x}) < 0$ if and only if $\boldsymbol{x} \neq \boldsymbol{x'}$. This shows that no cycles can occur and therefore establishes convergence to fixed points only.

■

## 4.5 Hopfield Clique Networks

Theorems 3.6 and 4.1 establish the mathematical foundation to formulate a Hopfield model that requires no tuning of system parameters. On this basis, Section 4.5.1 proposes a simple *Hopfield Clique Network* and discusses some design issues, including optimal parameter selection in particular. To improve computation speed without loss of solution quality, we incorporate a novel technique in the dynamics of the simple Hopfield Clique Network. This technique adapts the system parameters according to the subproblem induced by the current set of active units. Such an approach is possible, because we know how to select an optimal parameter setting for each problem instance. We call the improved model *Attention Control System*, as it allows an interpretation in terms of selective attention. Section 4.5.2 is devoted to the attention model.

### 4.5.1 A Basic Hopfield Clique Algorithm

Algorithm 2 describes the basic *Hopfield Clique Network* (HCN) procedure based on the results of Theorems 3.6 and 4.1.

**Algorithm 2** (Hopfield Clique Network Algorithm)

**Input:**

$X$ – weighted graph $X = (V, E, \boldsymbol{X})$

$\varepsilon$ – positive constant

**Initialization:**

set activation $\boldsymbol{u}$

$$\mu = \Delta_{\mathrm{w}}\left(X\right) + \|\boldsymbol{c}\|_{\infty} + \varepsilon$$

$$\lambda = \frac{2T}{\delta(X) + \mu\,\Delta_{\mathrm{co}}(X)} - \varepsilon$$

$$\boldsymbol{W} = \lambda\big(\boldsymbol{X} - \boldsymbol{D}_{\boldsymbol{X}} - \mu\boldsymbol{A}\big)$$

$$\boldsymbol{h} = \lambda\boldsymbol{d}_{\boldsymbol{X}}$$

**Procedure:**

**repeat**

$$\boldsymbol{u} = \boldsymbol{u} + \boldsymbol{W}\boldsymbol{x} + \boldsymbol{h}$$

$$\boldsymbol{x} = g(\boldsymbol{u})$$

**until** convergence to a stable fixed point

**Output:** $\boldsymbol{x}$

The critical system parameters of the HCN algorithm are the penalty $\mu$ and the step size $\lambda$. Theorem 3.2 on the continuous formulation of the MWCP provides a lower bound

$$B_{\mu} = \Delta_{\mathrm{w}}\left(X\right) + \|\boldsymbol{c}\|_{\infty}$$

of $\mu$ to ensure feasibility.[3] The Hopfield Clique Theorem 4.1 provides an upper bound

$$B_\lambda = \frac{2T}{\delta(X) + \mu\,\Delta_{\mathrm{co}}(X)}$$

of $\lambda$ to ensure energy descent.[4] In the following, we show how to optimally select both parameters and discuss some other design decisions.

***Transfer Function***

The choice of transfer function $g$ is a problem dependent design decision for which a general statement regarding optimality is unknown. But there are some qualitative properties of transfer functions that are worth mentioning.

Consider the logistic function $g_\beta(u)$ and the piecewise-linear limiter function $[u]_\beta$. The gain $\beta$ determines the Lipschitz constant $L$ and controls the slope of the transfer function. Lowering the gain shifts internal states (activations) closer to the origin and refines the search. Conversely, at high gains, the internal states are displaced far from the origin and a coarse-grained search near the corners of the hypercube is performed. Since the upper bound $B_\lambda$ is proportional to $T = L^{-1}$, the step size $\lambda$ largely absorbs the effects of the gain.

The main difference between both transfer functions is that logistic dynamics bounds trajectories to the interior of the unit hypercube and trajectories generated by limiter dynamics can also evolve through boundary points of the hypercube. This difference may affect the convergence behavior. In contrast to units with limiter transfer functions, logistic units with negative activation still contribute to the activation of all other units.

***Step Size Parameter $\lambda$***

The step size $\lambda$ controls the convergence behavior of any steepest descent method. Small values of $\lambda$ lead to slow convergence, and too large values of $\lambda$ may cause jamming (zig-zagging).

Jamming occurs when the descent directions $\boldsymbol{d}_t$ and $\boldsymbol{d}_{t+1}$ at two consecutive time steps point in opposite directions. In mathematical terms, we have

$$\boldsymbol{d}_t^{\mathsf{T}} \boldsymbol{d}_{t+1} \leq 0. \tag{4.15}$$

From the proof of the Hopfield Clique Theorem 4.1 follows that $E$ decreases along the line segment $\left[\boldsymbol{x}_t, \boldsymbol{x}_{t+1}\right]$. This implies that

$$\boldsymbol{d}_t^{\mathsf{T}} \boldsymbol{E}_{t+1} > 0,$$

3. Recall from Section 2.2.3 that $\Delta_{\mathrm{w}}(X)$ is the maximum weight degree of $X$.
4. Recall from Section 2.2.3 that $\delta(X)$ is the minimum degree and $\Delta_{\mathrm{co}}(X)$ the maximum co-degree of $X$.

where $\boldsymbol{E}_{t+1}$ is the gradient of $E$ at time step $t+1$. If $-\boldsymbol{E}_{t+1}$ is a feasible direction, then the next descent direction $\boldsymbol{d}_{t+1} = -\boldsymbol{E}_{t+1}$ satisfies (4.15). Thus, jamming does not occur in the interior of the hypercube, but can be caused if descent along $\boldsymbol{d}_t$ runs against a face of the hypercube.

If the transfer function is the logistic function, all trajectories remain in the interior of the hypercube and no jamming occurs. For limiter transfer functions, jamming may occur in a limited controlled manner without effecting convergence speed. Uncontrolled oscillation that degrades the convergence rate is unlikely, because $E$ decreases along the line segment $[\boldsymbol{x}_t, \boldsymbol{x}_{t+1}]$.

Since uncontrolled jamming does not occur, we may choose any value for the step size $\lambda$ bounded above by $B_\lambda$. To present an optimal choice of $\lambda$ with respect to the requirements of neural learning machines for attributed graphs, we examine the upper bound $B_\lambda$. A simple calculation for $T = 1$ shows that the magnitude of the upper bound $B_\lambda$ of $\lambda$ is of order

$$O\left(\frac{1}{n}\right) \leq B_\lambda \leq O\left(\frac{1}{n^2}\right). \tag{4.16}$$

From the estimations of $B_\lambda$, we can expect a degrading convergence rate for increasing dimension $n$. Figure 4.2 illustrates the major problem of this phenomenon. The plot shows that even for problem instances of small order, the step size becomes small compared to the distance to be covered. As a consequence, the HCN will also exhibit slow convergence for the largest admissible step size $\lambda < B_\lambda$. Hence, the best we can do to meet the requirements of neural learning machines for graphs is to choose

$$\boxed{\lambda^* = B_\lambda - \varepsilon}$$

as optimal step size, where $\varepsilon > 0$ is as small as possible.

***Penalty Parameter*** $\mu$

As pointed out in Section 4.3.7, choosing too large a value for $\mu$ makes our system

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t + \boldsymbol{W} g(\boldsymbol{x}_t) + \boldsymbol{h}$$

*ill-conditioned* in the sense that small changes in $\boldsymbol{u}_t$ can lead to relatively large changes in $\boldsymbol{u}_{t+1}$. As a consequence, more emphasis is placed on minimizing constraint violations and the system prematurely terminates at a solution point of poor quality. Examining the upper bound $B_\lambda$ of $\lambda$ reveals that the unfavorable effects of larger values for $\mu$ are mostly compensated by a smaller step size $\lambda$. From our discussion on the optimal choice of $\lambda$, smaller step sizes further slow down the convergence process. Hence, the optimal choice for the penalty parameter is

$$\boxed{\mu^* = B_\mu + \varepsilon},$$

where $\varepsilon > 0$ is as small as possible.

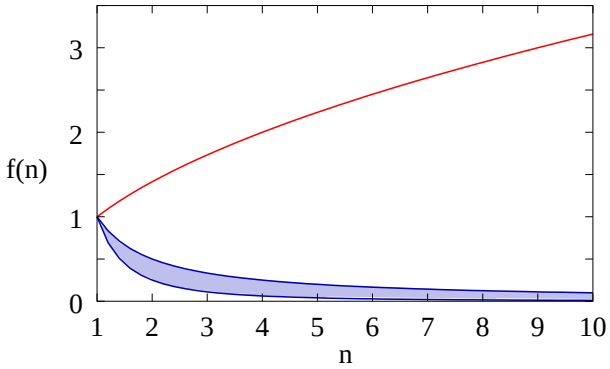



**Figure 4.2** The red line shows the length of the diagonal of an $n$-dimensional hypercube $\mathbb{U}^n$ as a function of $n \in [1:10]$. The shaded region indicates where we may expect the upper bound $B_\lambda$ of the step size $\lambda$. Assume that the initial state is approximately at the center of $\mathbb{U}^n$. The problem is that the distance to be covered increases of order $O(\sqrt{n})$, but the step size decreases of order $O(1/n^2)$ in the worst case.

### *Initial Activation*

As a gradient descent method, solution quality of HCN strongly depends on an appropriate choice of the initial point. The problem of choosing a suitable initial solution is one of the major shortcomings of gradient descent methods. Though HCN requires no tuning of system parameters to find a solution, it requires domain knowledge to obtain satisfactory solutions by appropriate initialization of the activation vector.

### *Termination*

Models that use the limiter transfer function will converge after a finite number of updates to a corner of the unit hypercube. Checking stability for termination is straightforward and requires no additional parameters.

In contrast, trajectories evolving under logistic dynamics converge to a stable fixed point but never reach it. The common way to terminate the logistic system is after a prespecified number of iterations or when the change of the output states is less than a given threshold. Since we want to avoid problem dependent parameters that must be selected manually, we formulate a termination criterion for both transfer functions that requires no tuning parameters.

***Termination Criterion:*** *Terminate Algorithm 2 when the following condition is satisfied*

$$\Theta(\boldsymbol{u}_t) = \Theta\big(\boldsymbol{W}\Theta(\boldsymbol{u}_t) + \boldsymbol{h}\big). \tag{4.17}$$

The threshold function $\Theta$ transforms an internal state $\boldsymbol{u}$ into a characteristic vector $1_U$ of a subset $U$ of vertices from $X$. From the proof of Theorems 3.2 and 4.1, it follows that (4.17) holds if and only if $\Theta(\boldsymbol{u}_t)$ is a fixed point. Once (4.17) holds, there is no need to wait until the trajectory saturates in a corner of the hypercube.

Noteworthy is that in practical implementation, computation of the termination criterion (4.17) does not require a second quadratic iteration cycle at each update step.

### 4.5.2 Selective Attention Control Systems

Even for simple problem instances, the small step size $\lambda$ of the HCN Algorithm induces slow convergence behavior. To improve convergence speed without loss of solution quality, the *Attention Control System* (ACS) optimally adapts the system parameters $\mu$ and $\lambda$ of the Hopfield Clique Network at each iteration step.

Suppose that we want to solve the MWCP for the graph $X = (V, E, \boldsymbol{X})$ of order $|X| = n$. At each update step, the ACS algorithm proceeds as follows:

1. *Stimuli Priorization.* Partition the vertex set $V$ into a set $V_a$ of *active units* $i$ with $u_i > 0$ and a set $V_p$ of *passive units* $j$ with $u_j \leq 0$.[5]
2. *Attention Selection.* Determine the subgraph $X_a = X[V_a]$ induced by the active units and recompute the penalty $\mu$ and the step size $\lambda$ with respect to $X_a$.
3. *Competition.* Update the state of *all* $n$ units according to the HCN dynamics.

Algorithm 3 summarizes the complete ACS procedure. For convenience of presentation, we consider the limiter transfer function. By

$$B_\mu(X_a) = \Delta_{\mathrm{w}}(X_a) + \|\boldsymbol{c}\|_\infty$$
$$B_\lambda(X_a) = \frac{2T}{\delta(X_a) + \mu\,\Delta_{\mathrm{co}}(X_a)},$$

we denote the lower and upper bound of $\mu$ and $\lambda$ with respect to the subgraph $X_a$.

Whenever at least one unit changes from being active to passive, the trajectory hits a face $\mathcal{F}_a$ of the unit hypercube $\mathbb{U}^n$. Unless no unit changes from being passive to active, the trajectory moves along the surface $\mathcal{F}_a$. The surface $\mathcal{F}_a$ corresponds to the search space of the MWCP for the induced subgraph $X_a$. Since $X_a$ has less vertices than $X$, we have $B_\mu(X_a) < B_\mu(X)$ and $B_\lambda(X_a) > B_\lambda(X)$. For fastest convergence, we first lower $\mu$ to $B_\mu(X_a) - \varepsilon$ and then increase the step size $\lambda$ to the value $B_\lambda(X_a) - \varepsilon$. As a consequence, the dynamics forces the pace to cover smaller distances in a reduced subspace. Figure 4.3 illustrates a typical behavior of ACS.

The dynamics updates the state of all its active and passive units. Only active

5. For the sake of simplicity, we identify vertices of $X$ with the corresponding units of the Hopfield model.

**Algorithm 3** (Attention Control System Algorithm)

**Input:**
$X$ – weighted graph $X = (V, E, \boldsymbol{X})$
$\varepsilon$ – positive constant

**Initialization:**
set activation $\boldsymbol{u}$

**Procedure:**
**repeat**
Adjust system parameters $\mu$ and $\lambda$:
- $V_a = \{i \in V : u_i > 0\}$
- $X_a = X[V_a]$
- $\mu = B_\mu(X_a) + \varepsilon$
- $\lambda = B_\lambda(X_a) - \varepsilon$

Update network
- $\boldsymbol{u} = \boldsymbol{u} + \lambda(\boldsymbol{X} - \boldsymbol{D_X} - \mu\boldsymbol{A})\boldsymbol{x} + \lambda\boldsymbol{d_X}$
- $\boldsymbol{x} = [\boldsymbol{u}]_{\frac{1}{T}}$

**until** convergence to a stable fixed point

**Output:** $\boldsymbol{x}$

units contribute to the activation of a unit. Passive units remain quiescent and do not stimulate other units. Limiter transfer functions automatically satisfy this condition. The dynamics for logistic transfer functions requires some minor modifications to behave as desired. Note that units corresponding to members of a maximal weight clique can become passive during evolution and must therefore be reactivated.

### *Controlled Forgetting*

To further improve speed, a forgetting mechanism can be introduced. Several criteria to forget a passive unit are possible. We suggest three simple criteria:

1. Delete a passive unit in the network when its activation decays below a given (negative) threshold.
2. Delete a passive unit in the network when its activation has been decreasing for a given period of time.
3. Delete a passive unit in the network when the gradient of its activation is below a given (negative) threshold.

Once a unit is deleted, it is excluded from the updating process. The difference between a forgotten (or deleted) and a passive unit is that the former is excluded from the updating process. Thus, this mechanism is useful for problem instances of large order ($> 10\,000$), because it enables us to reduce the problem to a computationally manageable size. Appropriate choice of the parameters for a

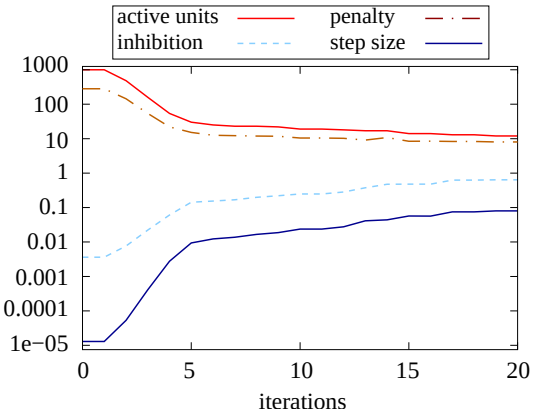



**Figure 4.3** Illustration of how selected parameters of ACS vary with evolving time. A random weighted graph of order 1000 is given. Vertex and edge weights were sampled from a uniform distribution over $\mathbb{U}$. The red solid line at the top shows the number $n$ of active units, the brown dashed-dotted line is the penalty parameter $\mu$, the light blue dashed line presents the inhibition $\lambda\mu$, and the solid blue line at the bottom shows the step size $\lambda$. At iteration step 0, ACS and HCS have the same values. After a few iteration steps, the number of active units of ACS sharply decreased. The step size was adapted according to the subgraph induced by the active units and therefore sharply increased.

forgetting criterion are essential for a satisfactory trade-off between accuracy and speed. We do not further consider controlled forgetting, because our focus is on a system that requires no tuning of critical system parameters.

### 4.5.3 Selective Attention – A Metaphor

This subsection provides an interpretation of ACS in terms of selective attention from the field of cognitive science. The purpose of this interpretation is an attempt to indicate possible links between neural network models from the pure optimization-based point of view and cognitive psychology. It is important to stress that the goal is neither to claim a new model for selective attention nor to provide a coherent neural implementation of a cognitive model.

Selective attention is a mechanism that preferentially selects a subset of relevant or important information from given data for further processing. Since the capacity for processing information is limited, the ability to filter out unwanted information is essential for effective cognitive performance.

In an extreme view, limited capacity for processing information allows us to conveniently classify limitations of attention in *divided* and *focused attention*. In divided attention paradigms, the subject is required to divide its attention between many stimuli. Limits are seen, because not many things can be attended to at once. As a general rule, subjects find it extremely difficult to divide their attention.

When there are more stimuli to be attended to, performance is reduced. In focused attention paradigms, subjects attempt to focus all available attention on just one type of stimuli, largely ignoring and/or excluding all other inputs.

Selectivity – the ability to focus attention on relevant stimuli – is achieved by an enhancement of processing relevant information. Attentive processing of relevant information may be achieved by mechanisms that impede the processing of irrelevant information, whereby inhibitory processes play an important role in selective attention.

On the basis of neuroimaging and psychological experiments, de Fockert et al. [70] conjectured that selective attention relies on the working memory to prioritize stimuli for processing relevant information. They predicted that a higher working memory load increases distractor processing. To clarify the conjecture, we briefly explain the terms working memory, memory load, and distractor processing. *Working memory* is a more contemporary term for *short-term memory*. It conceptualizes an active system for temporarily storing and manipulating information needed in the execution of complex cognitive tasks [14]. *Working memory load* is the amount of information kept in mind at any given time. Distractors are irrelevant stimuli. Thus *distractor processing* refers to the processing of unwanted information.

As indicated by Levy and Pashler [229], several authors assume that processing relevant stimuli (*targets*) requires and engages some amount of limited resources. When the resources are engaged in processing targets, they are not available for processing distractors. But if they are not engaged in target processing, they are *idle* and suspectible to being captured by the distracting stimuli, resulting in their processing [357].

To draw a metaphoric analogy of models from selective attention to the ACS, we consider a visual scene consisting of several isolated, abutting, or overlapping objects. The cognitive task is to detect the occurrence of a particular type of object. Visual stimuli received by the perceptual system are transformed to an internal mental model of the image. Information is encoded in the ACS working memory, where the units of maximal cliques are distributed representations of the objects in the scene. Competition by lateral inhibition filters out unattended stimuli and identifies regions of interest. The competitive bottom-up mechanism can be directed by a top-down control by applying an additional external input. Top-down controls the information relevant to the task. Active units model potential relevant stimuli and passive units represent distractor stimuli. Passive units decay or can be reactivated when attention shifts to focus on other parts of the region.

The inverse $\lambda^{-1}$ of the step size is a crucial system parameter system. It may be interpreted as a measure of memory load or resources engaged in processing stimuli. Large values for the step size correspond to low memory load and vice versa. Recall from our discussion on the step size parameter that the upper bound $B_\lambda$ decreases with increasing number $n$ of vertices of the given graph. Hence, $B_\lambda^{-1}$ increases with $n$, which corresponds with the property of memory load that load increases as items are presented. This correspondence is consistent in the sense that response time of both the metaphoric and the cognitive attentional models increase with the number

$n$ of items presented.[6]

At the same time, the quantity $\lambda^{-1}$ may serve as a measure for resources engaged in processing stimuli. The difference between ACS and HCN is that the former shifts all its resources to relevant regions by adapting the step size appropriately, whereas the latter becomes more and more idle in the sense that distracting stimuli are processed.

Hence, in our metaphoric ACS model, inhibition is essential to identify relevant stimuli, but it is not sufficient to shift attention to that stimuli. Shifting attention to relevant stimuli is modeled by adapting the step size $\lambda$.

We conclude with a final remark on an interpretation of ACS with controlled forgetting. In cognitive psychology, forgetting is the inability to retrieve previously stored information. Because it is unknown how memories are precisely stored, it is not possible to say exactly how an individual forgets. Among a number of theories, one model of forgetting is based on decay theory [25]. Decay theory suggests that stored information will be lost, unless it is revised regularly over time. Decay theory models motivated our model of controlled forgetting.

## 4.6 Experiments

This section discusses empirical issues related to the ACS. Section 4.6.1 serves to illustrate the effects of *selective attention*. In Section 4.6.2, we compare the performance of ACS applied to the MWCP against the performance of selected Hopfield network approaches. In Sections 4.6.3 and 4.6.4, we evaluate ACS on the problem of comparing segmented images and chemical graphs, respectively.

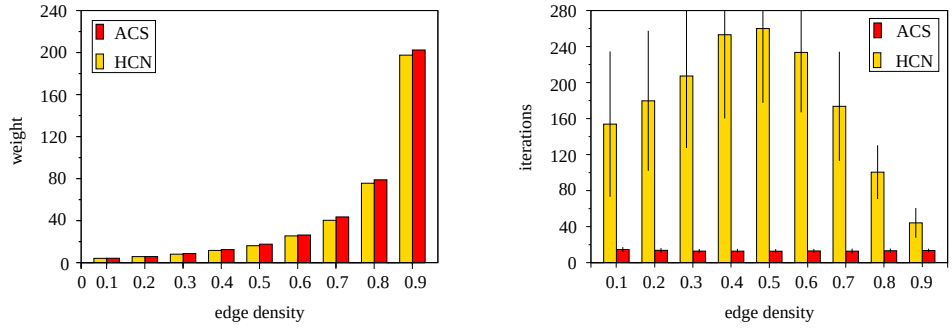
### 4.6.1 Selective Attention

To illustrate the effects of selective attention, we compared the performance of ACS with the basic HCN. Both algorithms were applied to the MWCP of random weighted graphs, where weights have been assigned according to an independent and identically distributed sampling scheme over $]0, 1]$. Details of the graph generator are described in Appendix B.3. We conducted two experiments:

1. In the first experiment, we investigated how selective attention depends on the edge probability, or density, $p$ of the test graphs. The chosen edge probabilities are $p = 0.1, 0.2, \ldots, 0.9$. For each edge probability $p$, we generated 100 weighted random graphs of order $n = 100$.

2. The second experiment serves to study how selective attention depends on the size $n$ of the test graphs. For each size $n \in [25 : 25 : 250]$, we generated 100

6. For example, Howard et al. [156] empirically showed that response time increases with the number of item presented.




(a) Average clique weights

(b) Average number of iterations

**Figure 4.4** Results of ACS (red) and HCN (gold) applied to the MWCP of random weighted graphs of order $n = 100$. Shown are (a) the average solution quality and (b) the average computation time of both algorithms for varying edge probabilities. Thin bars in (b) indicate the standard deviation of the average time.

weighted random graphs with edge probability $p = 0.75$.[7]

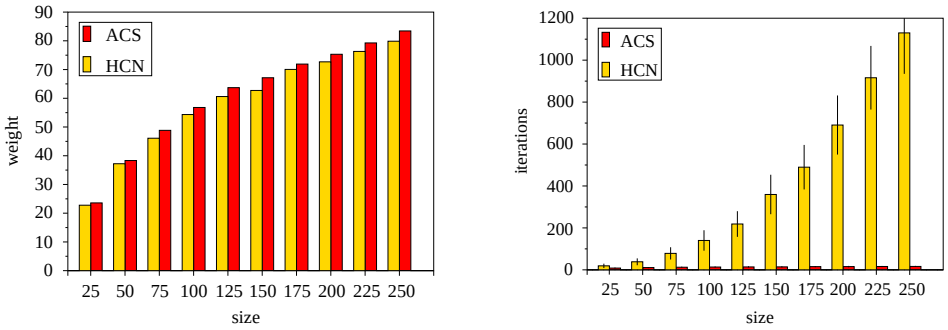
Figures 4.4 and 4.5 summarize the average performance of ACS and HCN for both experiments.

From the plots of Figures 4.4(a) and 4.5(a), we see that the solution quality of ACS and HCN is almost the same. This is exactly what we expect because of the shape of the energy landscape and the choice of the step size $\lambda$. A curious phenomenon is that on average the ACS always returned slightly better solutions than the HCN. Though the penalty parameter $\mu$ of HCN is incommensurately large after a number of units have been inhibited, its adverse effects are largely absorbed by the step size $\lambda$. The ratio $\omega_{ACS}/\omega_{HCN}$ of the average maximal clique weights found by ACS and HCN lies within the range $[1.0, 1.08]$ and independent of size and density.

As expected from the discussion on the effects of selective attention (see Figure 4.3), ACS significantly outperformed HCN with respect to computation time as shown in Figures 4.4(b) and 4.5(b). In addition, the plots indicate that the average number of iterations of the ACS scarcely increases with increasing size, and is almost constant for varying density.

To summarize, the principle of selective attention significantly improves convergence speed and scaling with problem size. In addition, it exhibits a robust convergence behavior with respect to varying edge probabilities.

7. Recall that $[n : s : m]$ denotes the set of all elements from $n$ to $m$ in steps of $s$.




(a) Average clique weight

(b) Average number of iterations

**Figure 4.5** Results of ACS (red) and HCN (gold) applied to the MWCP of weighted random graphs with fixed edge probability $p = 0.75$. Shown are (a) the average solution quality and (b) the average computation time of both algorithms for varying size. Thin bars in (b) indicate the standard deviation of the average time.

### 4.6.2 Maximum Weight Clique Problem

In this series of experiments, we assessed the performance of the ACS applied to the MWCP of synthetic test graphs. We considered the MWCP, because the focus is on generic solutions to $\mathfrak{p}$-GMPs, which are captured by MWCPs in a derived association graph.

#### *Selected Algorithms*

To compare the results of ACS, we selected the following Hopfield models:

| Acronym | Method | State space | Convergence |
|---|---|---|---|
| EPP | Exterior-Point Penalty Network | discrete | fast |
| SD | Steepest Descent | discrete | fast |
| SSD | Steepest Stochastic Descent | discrete | slow |
| MFA | Mean-Field Annealing | continuous | slow |
| RAC | Repeated Attention Control | continuous | fast |

The algorithms and their parameter settings are presented in Appendix A. For the standard MCP, the algorithms EPP, SD, SSD, and MFA have been systematically investigated by [167]. We slightly modified these algorithms to make them applicable to the MWCP. The fifth algorithm, RAC, runs ACS $n/10$ times with different random initial activations, where $n$ is the order of the input graph.

The selection of algorithms against which to compare ACS was motivated by the

following factors:

1. Our primary focus is on efficient neural network solutions to the MWCP. Therefore, the selected algorithms for comparison are extensions of known Hopfield models.

2. Since the MWCP has not been previously systematically investigated, the selected Hopfield models for comparison should be easily extendable to the MWCP.

3. The order of association graphs increases quadratically with the order of the input graphs to be matched. For problems that involve a huge number of comparisons of large graphs, good approximations of the MWCP obtained by slower methods are useless. The algorithms selected for comparison should therefore trade-off speed and accuracy in favor of speed.

We excluded the most powerful connectionist solutions to the standard MCP — the Binary Neural Network [99] and the RaClique algorithm [393] — from our comparative study, because both algorithms have been empirically proved to be too slow for large-scaled problems.

EPP and SD are fast solutions for the standard MCP. SD and ACS are both algorithms that neither can escape from local minima, nor use any other sophisticated technique to improve solution quality. Therefore, we also considered SSD and RAC. Both algorithms introduce some randomness and run SD and ACS, respectively, several times to obtain better approximations. SSD uses a stochastic version of SD. RAC varies the initial activation with each call of ACS using the principle of selective attention. For further details we refer to Appendix A.1.2. Here, the number of SD runs called by SDD is twice as much as the number of ACS runs called by RAC.

The MFA algorithm applied to the MCP performed reasonably well on various test suites and on DIMACS benchmarks graphs [191]. Though MFA is faster than the Binary Neural Network and RaClique, it is still too slow for our purposes. We nonetheless included MFA in our test suite in order to investigate the potential applicability of mean-field techniques to the MWCP.

#### *Selected Test Graphs*

We applied the algorithms to the MWCP of weighted random graphs ($G1$) and weighted $k$-random cliques graphs ($G2$). To assign weights to the vertices and edges, we used an irregular uniform sampling scheme for $G1$ graphs and an independent Gaussian distributed sampling scheme for $G2$ graphs. Appendix B.3 describes the generators of the test data. Note that the MWCP for large graphs with an independent and identically distributed sampling scheme basicly maximizes the cardinality of vertices. For this reason, we sampled the weights from an irregular and a Gaussian distribution.

Since Jagota's systematic investigations [167], unweighted random graphs and

$k$-random cliques graphs are commonly used in the neural network literature to evaluate the performance of various Hopfield models for the MCP. From a structural point of view, random graphs are attractive because they cover a wide variety of structural properties including, for example, (non)-connectivity, (non)-cyclic, and (non)-planarity. Random cliques graphs are interesting because they cover a broad range of clique sizes.

Statements about the hardness of test graphs for the the MCP are, in general, no longer valid for the MWCP. For this reason, we excluded the DIMACS benchmark set [191].

***Evaluation Procedure***

We conducted four series of experiments:

| ID | Graphs | Order $n$ | $p$ or $k$ | Trials |
|---|---|---|---|---|
| $E_{G1}^{100}$ | $G1$ | 100 | $p \in ]0,1[$ | 1000 |
| $E_{G1}^{1000}$ | $G1$ | 1000 | $p \in ]0,1[$ | 1000 |
| $E_{G2}^{100}$ | $G2$ | 100 | $k \in [3:10]$ | 1000 |
| $E_{G2}^{1000}$ | $G2$ | 1000 | $k \in [5:20]$ | 1000 |

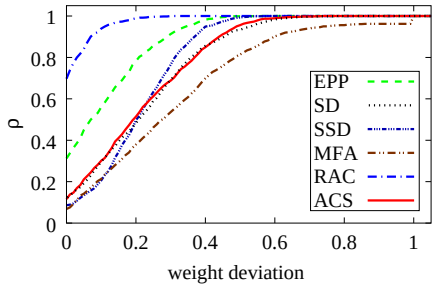
The parameter $p$ denotes the edge probability of weighted random graphs and $k$ is the the number of randomly generated cliques in the $k$-random cliques graph model. In each trial, the corresponding parameter $p$ or $k$, respectively, was drawn from a uniform distribution on the specified interval. Since SSD and MFA are too time consuming, we excluded both algorithms from experiments on test graphs with $n = 1000$ vertices.

***Numerical Results***

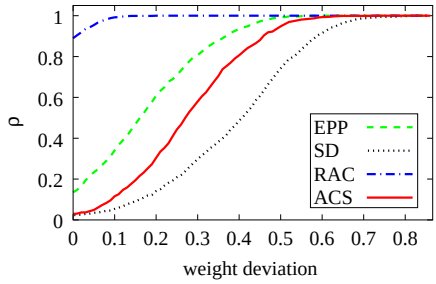
To analyze the results, we used performance profiles as described in Appendix B.1.

Figure 4.6 presents the performance profiles of each algorithm using the clique weight as performance measure. From the plots we see that RAC clearly outperforms all other algorithms in all four test series. For graphs of order 100, RAC returned the best solution in about 70% of all trials. This percentage increases for graphs of order 1000 to about 90%. This can be verified in Figure 4.6 by looking at the values $\rho(0)$ of RAC. In all 4000 trials, the solution found by RAC deviates at most about 10%-20% from the best solution found by any of the selected algorithms. All other algorithms deviate about 50%, or even more from the best solution.

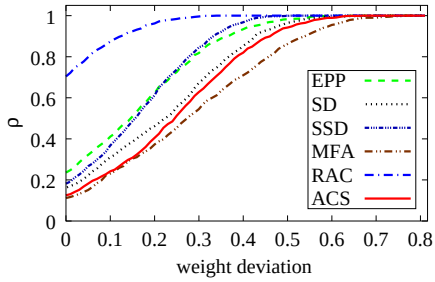
Among the remaining algorithms, ACS averages and is roughly equivalent to its discrete counterpart SD. It is notable that $R$ runs of ACS called by RAC using the principle of selective attention for different initial states are significantly superior than $2R$ runs of a stochastic version of SD called by SDD.



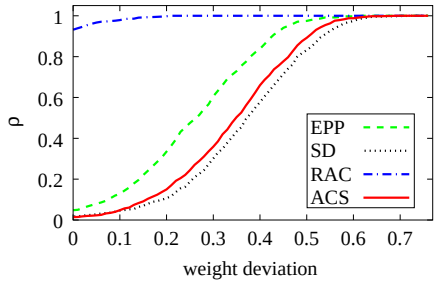

(a) $E_{G1}^{100}$ – Profiles for clique weight




(b) $E_{G1}^{1000}$ – Profiles for clique weight

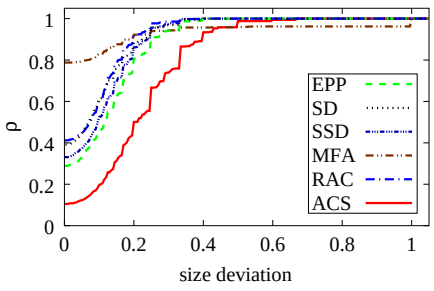



(c) $E_{G2}^{100}$ – Profiles for clique weight

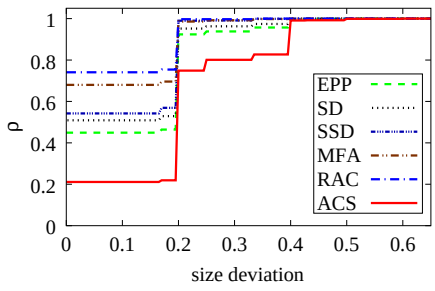



(d) $E_{G2}^{1000}$ – Profiles for clique weight

**Figure 4.6** Performance profiles of the selected algorithms using *clique weight* $\tau$ as performance measure. For each algorithm, the quantity $\rho(\tau)$ shows the fraction of solutions that deviate $100 \cdot \tau$ % at most from the best solution found by any of the algorithms.




(a) $E^{100}_{G1}$ – Profiles for clique size



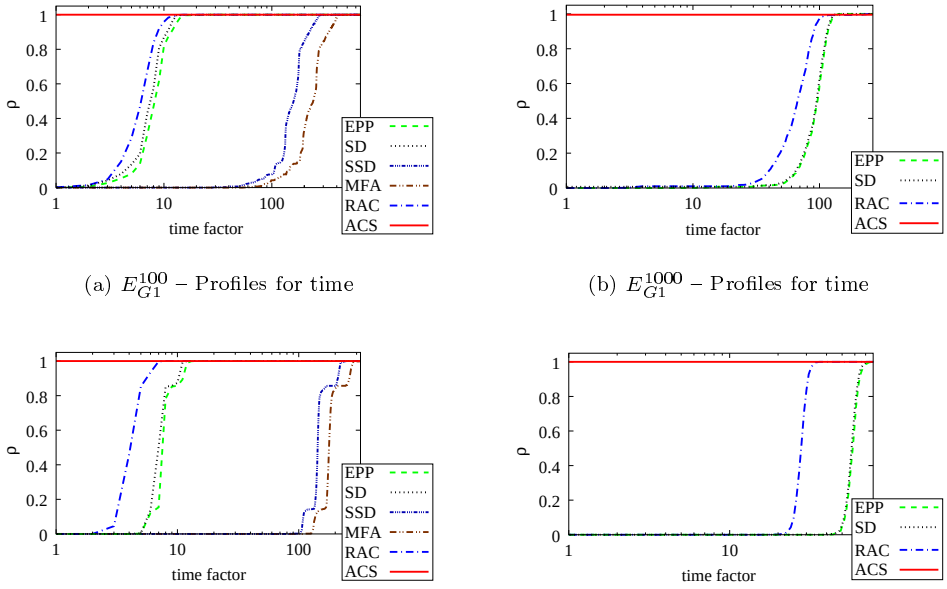

(b) $E^{100}_{G2}$ – Profiles for clique size

**Figure 4.7** Performance profiles of the selected algorithms using *clique size* (cardinality of vertices) $\tau$ as performance measure. For each algorithm, the quantity $\rho(\tau)$ shows the fraction of solutions that deviate $100 \cdot \tau$ % at most from the best solution found by any of the algorithms.

It is instructive to note that MFA performs worst and is even clearly outperformed by SD and EPP. The result contrasts to the results for the standard MCP on the unweighted counterparts of $G1$ and $G2$ graphs [167]. This phenomenon is closely related to the problem illustrated in Example 4.2 on page 76. The original energy landscape $E = E_0$ $(T = 0)$ to be minimized is dominated by the large penalty parameter $\mu$. Regions with less inhibitory connections have wider basins of attractions and vice versa. As indicated in Example 4.2, MFA is more likely to settle in wide basins. Consequently, this procedure works fine when cliques of large weight have, in average, more neighbors than cliques of less weight. This holds for the standard MCP on unweighted $G1$ and $G2$ graphs and explains the outstanding performance of MFA shown in [167]. For the MWCP, the weight of a clique is generally unrelated to its size. Hence, for $G1$ and $G2$ graphs, MFA tries to maximize the size of a clique rather than its weight. Figure 4.7 supports this claim, showing the performance profiles for the size of a clique. Together with Figures 4.6(a) and 4.6(c), we see that MFA has the largest disparity between weight and size of a clique. In contrast, ACS searches for maximal cliques with the highest average weight per item.

Figure 4.8 presents the performance profiles using clock time as performance measure. The plots show that ACS is the fastest algorithm in all 4000 trials. In addition, ACS scales better than EPP, SD, and RAC with the number of vertices.

If we regard ACS as a special case of RAC, the overall result of this empirical study is that RAC significantly outperforms all other algorithms with respect to solution quality and computation time.




(a) $E^{100}_{G1}$ – Profiles for time

(b) $E^{1000}_{G1}$ – Profiles for time

(c) $E^{100}_{G2}$ – Profiles for time

(d) $E^{1000}_{G2}$ – Profiles for time

**Figure 4.8** Performance profiles of the selected algorithms using *clock time* $\tau$ as performance measure. For each algorithm, the quantity $\rho(\tau)$ shows the fraction of solutions that are $\tau$ times slower than the fastest algorithm at most.

### 4.6.3 Similarity of Segmented Images

Image retrieval based on image features has attracted considerable attention in recent years [30, 31, 256, 331]. Results obtained by standard approaches using color and edge histograms are fast, but their results are unsatisfactory because they ignore semantic information. Bischoff et al. [31] suggested representing images by attributed graphs to enable the search of images, which contain prespecified semantic objects such as cars, houses, or trees. One problem with this approach is that graph-based representations in image retrieval are too slow due to the computational complexity of the graph matching problem and the lack of fast algorithms.

To reduce computation time, the goal of this empirical study is to propose and evaluate a compact framework using ACS in order to compute a structural similarity measure for segmented images as suggested by Bischoff et al. [31].

#### *Problem Formulation*

Given an input image, the goal of image retrieval is to identify a small subset of images from a dataset, which are most similar to the input. Here, we consider the task of emulating what humans would intuitively perceive as similar. Although this is a rather unspecific setting – at least from the perspective of a computer scientist – we might nevertheless assume, for example, that humans will usually perceive images 1 and 2 in Figure 4.9 as being more similar than images 1 and 3. Furthermore, images 1 and 3 might in turn be perceived as more similar than images 1 and 13. To provide a more precise setting, we divide the images in Figure 4.9 into different categories as shown in Table 4.3. The problem at hand is to provide a similarity measure and an algorithm that perceive images within the same category as similar, and images from different categories as dissimilar.

| Images | Category |
|---|---|
| 1-2 | FARMER I |
| 3-6 | FARMER II |
| 7-12 | FISH |
| 13-21 | NEWSREADER I |
| 22-24 | NEWSREADER II |
| 25-26 | OUTDOOR |
| 27-29 | MAN |
| 30-32 | WOMAN |
| 33-37 | ASIAN |
| 38-39 | CAMERA |

**Table 4.3** Categories of Images shown in Figure 4.9.

### *The Bischoff-Reuß-Wysotzki Measure*

To emulate human judgment of similarity, we transform the problem of comparing images to the problem of comparing attributed graphs. The resulting Bischoff-Reuß-Wysotzki measure was proposed in [31].

What makes comparison of segmented images so difficult is the unstable behavior of a segmentation procedure in regard to slightly varying illumination conditions, camera positions, and algorithm parameters. For example, image 1 and 2 in Figure 4.9 have 50 and 37 segments, respectively. To cope with this problem, we adopt the approach suggested by Bischoff et al. [31] and approximate a structural similarity measure based on the *Consistent Labeling Problem.*[8]

First, we transform each segmented image to a complete attributed graph. For a given image, each vertex represents a segment. A vertex is labeled with a 166-dimensional feature vector describing the visual properties of the corresponding segment. Two vertices are connected by an edge labeled with the Euclidean distance between the centers of the corresponding segments.

Next, we formulate the graph matching problem. Assume that $X$ and $Y$ are attributed graphs representing two given images. As usual, let $\boldsymbol{X} = (\boldsymbol{x}_{ij})$ and $\boldsymbol{Y} = (\boldsymbol{y}_{ij})$ be the attributed adjacency matrices. The compatibility values for items $\boldsymbol{i}$ of $X$ and $\boldsymbol{j}$ of $Y$ are of the form

$$\kappa_{\boldsymbol{ij}} = \begin{cases} s_V(\boldsymbol{x_i}, \boldsymbol{y_j}) & : \quad \boldsymbol{i} \in V(X), \boldsymbol{j} \in V(Y) \\ s_E(0, y_{\boldsymbol{j}}) & : \quad \boldsymbol{i} \in V(X), \boldsymbol{j} \in E(Y) \\ s_E(x_{\boldsymbol{i}}, 0) & : \quad \boldsymbol{i} \in E(X), \boldsymbol{j} \in V(Y) \\ s_E(x_{\boldsymbol{i}}, y_{\boldsymbol{j}}) & : \quad \boldsymbol{i} \in E(X), \boldsymbol{j} \in V(Y) \end{cases},$$

where $s_V$ and $s_E$ are functions that measure the similarity of vertex and edge attributes. The following remarks might be helpful:

1. Arguments of $s_V$ and $s_E$ have different dimensions (166 vs. 1). This difference is expressed in the notation of the arguments.
2. To define $s_E$, we implicitly assume that each vertex has a loop with attribute 0. We did not make loops explicit in order to avoid notational confusion.

Both similarity functions are parameterizations of

$$s(\boldsymbol{x_i}, \boldsymbol{y_j}) = \left[1 - \alpha \|\boldsymbol{x_i} - \boldsymbol{y_j}\|\right]_\beta,$$

where $[.]_\beta$ denotes the limiter function with gain $\beta = 1$. The chosen values were $\alpha = 5$ for $s_V$ and $\alpha = 2.5$ for $s_E$.

8. See [30] and [383] for a description of the Consistent Labeling Problem.


**Figure 4.9** Dataset of 39 segmented images.

Now consider the graph matching problem

$$\begin{aligned} \text{maximize} \quad & f(\phi, X, Y) = \frac{1}{|X|} \sum_{i \in D(\phi)} \kappa_{iijj} \\ \text{subject to} \quad & j = i^{\phi} \\ & \phi \in \mathcal{M}, \end{aligned} \tag{4.18}$$

where $\mathcal{M}$ denotes the subset of partial morphisms $\phi \in \mathcal{M}_{X,Y}$ from $X$ to $Y$ with

$$\boldsymbol{i} \in \mathcal{D}(\phi) \;\Rightarrow\; \kappa_{\boldsymbol{i}\boldsymbol{i}^{\phi}} > 0$$

for all items $\boldsymbol{i} \in \mathcal{D}(\phi)$. Note that the matching objective $f$ only sums vertex similarities. Edge similarities serve to formulate the constraints of the problem. Thus, problem (4.18) tries to find a consistent labeling of vertices from $X$ in terms of vertices from $Y$ with maximal accumulated similarity of the labelings. Consistency is specified by the feasible region $\mathcal{M}$.

What makes formulation 4.18 so useful is that it enables us to group small segments of $X$ within a local neighborhood to a single large segment of $Y$, provided that all segments involved have similar visual properties. The function $s_E$ thereby controls the locality of the neighborhood, and $s_V$ the similarity of segments. This approach slightly differs from its original formulation in [31], where the local neighborhood is expressed in terms of adjacency relations.

Since (4.18) is not symmetric, i.e. $f(\phi, X, Y) \neq f(\phi, Y, X)$, Bischoff et al. [31] suggested the structural similarity measure

$$s(X, Y) = \frac{1}{2}\big(f(\phi^*, X, Y) + f(\psi^*, Y, X)\big), \tag{4.19}$$

where $\phi^*$ and $\psi^*$ are optimal solutions of the respective matching problems. A characteristic feature of the Bischoff-Reuß-Wysotzki measure (4.19) is that it allows a bi-directional mapping of smaller segments within a local neighborhood of one image to a single large segment of the second image. This feature makes the Bischoff-Reuß-Wysotzki measure robust against unstable segmentation.

### *Computing the Bischoff-Reuß-Wysotzki Measure*

Computation of the Bischoff-Reuß-Wysotzki measure involves the solution of two graph matching problems (4.18). For each graph matching problem, it is easy to see that the feasible region $\mathcal{M}$ is $\mathfrak{p}$-closed. According to Theorem 3.1, problem (4.18) is equivalent to the maximum vertex-weight clique problem (MVCP) in an association graph. Hence, it is feasible to apply ACS for approximately solving problem (4.18).

Since image retrieval requires fast solutions, solving two graph matching problems in order to determine a single structural similarity is counterproductive. We propose an approach that approximates the Bischoff-Reuß-Wysotzki measure by solving a single graph matching problem of the same complexity as either of both original matching problems.

Assume that $X$ and $Y$ are attributed graphs representing two given images. Let $X \diamond Y$ and $Y \diamond X$ denote the association graphs for the matching objectives $f(\phi, X, Y)$ and $f(\phi, Y, X)$. The association graphs $X \diamond Y$ and $Y \diamond X$ are defined on the same set of vertices, which can be obtained from one another by relabeling. But $X \diamond Y$ and $Y \diamond X$ may have non-isomorphic edge structures. By symmetry of $s_E$, an edge between two vertices in both association graphs either exists or is missing in at least one of both graphs. We merge $X \diamond Y$ and $Y \diamond X$ to an association graph $Z = (X \diamond Y) \amalg (Y \diamond X)$ by taking the union of the edge sets, where we identify identical edges of $X \diamond Y$ and $Y \diamond X$. Formally, the vertex set of $Z$ is defined by

$$V(Z) = V(X \diamond Y)$$

and its edge set by

$$E(Z) = \big\{(ik, jl) : (ik, jl) \in E(X \diamond Y) \vee (ki, lj) \in E(Y \diamond X)\big\}.$$

The weights assigned to vertices and edges of $Z$ are then determined by $X \diamond Y$, $Y \diamond X$ and the scaling of the Bischoff-Reuß-Wysotzki measure.

The MVCP formulation for $Z$ is not equivalent to, but provides a lower bound of the Bischoff-Reuß-Wysotzki measure. By construction of $Z$, each maximal (vertex-weight) clique $C$ of $Z$ can be decomposed into cliques $C_X$ of $X \diamond Y$ and $C_Y$ of $Y \diamond X$ such that their union is $C$. Since each vertex weight is counted once and the sets $C_X$ and $C_Y$ may overlap, the maximum weight clique of $Z$ is not larger than the Bischoff-Reuß-Wysotzki measure.

***Selected Algorithm***

We evaluated the performance of the following three algorithms on the problem of approximating the Bischoff-Reuß-Wysotzki measure.

| Acronym | Method |
|---|---|
| PMF | Potts Mean-Field Annealing |
| ACS | Attention Control System |
| MAC | Merged Attention Control |

The PMF algorithm and its parameter settings are presented in Appendix A. In contrast to the MFA algorithm applied in Section 4.6.2, PMF used a quenched annealing schedule. As in [31], PMF and ACS approximate the Bischoff-Reuß-Wysotzki measure by solving two graph matching problems (4.18). On the other hand, MAC requires the solution of only one graph matching problem to approximate the same measure. The MAC algorithm runs ACS on the merged association graph. Note that the edge structure of a merged association graph makes PMF inapplicable to the merged problem.

Bischoff et al. [31] applied the Competitive Layer Model proposed by Wersing

[383] and PMF to the matching problem (4.18). Since the PMF algorithm returned better approximations within shorter computation times, we selected PMF to compare it against ACS and MAC.

***Evaluation Procedure***

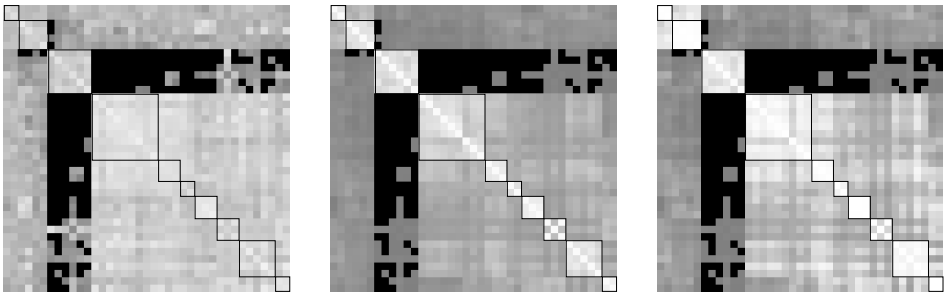
The segmented images to be compared are shown in Figure 4.9. The segmentation procedure is described in [30]. Additional information on the data is provided in Appendix B.2.1.

For each pair of segmented images, the selected algorithms approximated the Bischoff-Reuß-Wysotzki measure. Hence, PMF and ACS solved $1,560$ graph matching problems and MAC solved 780 matching problems.

The algorithms were implemented in Java using `JDK` 1.2. Experiments were executed on a multi-server `Sparc SUNW Ultra-4`.

***Numerical Results***

Figure 4.4 presents pairwise approximations of the Bischoff-Reuß-Wysotzki measure computed by the selected algorithms.



(a) PMF (b) ACS (c) MAC

**Figure 4.10** Shown are 2D visualizations of pairwise Bischoff-Reuß-Wysotzki similarities for (a) PMF, (b) ACS, and (c) MAC. Bright grey levels correspond to large similarity values. Rows and columns of the similarity matrices are ordered by the identity of the images given in Figure 4.9. Black squares in the diagonal outline the inter-similarities of images in the same category.

In absolute terms, the similarity matrices indicate that PMF exhibited the best average performance, MAC ranked second, and ACS performed worst.

We arrive at a completely different result when we assess the solution quality in terms of what humans would perceive as being similar. In our problem formulation, we asked for a system that exhibits highest similarity values between pairs of images

within the same category. Since PMF performed worst on pairs of images within the same category and best on pairs of images from different categories, separation of different categories is smeared. MAC performed best on pairs of images within the same category, and ACS worst on images from different categories. Both approaches provide satisfactory separation of different categories.

It is notable that ACS and MAC also perceive images of e.g. category FARMER I and FARMER II as more similar than an image from one of either FARMER category and an image from any other category. This observation does not apply for PMF.

| | time (*sec*) | | | |
|---|---|---|---|---|
| | min | max | avg | dev |
| PMF | 0.04 | 57.16 | 1.17 | 15.77 |
| ACS | 0.04 | 1.40 | 0.10 | 0.01 |
| MAC | 0.02 | 1.09 | 0.05 | 0.00 |

**Table 4.4** Clock times of selected algorithms for approximating the pairwise Bischoff-Reuß-Wysotzki similarities. Time was measured in *seconds*.

Table 4.4 summarizes the computation times of the selected algorithms. As expected, MAC was, on the average about twice as fast as ACS and 23 times faster than PMF. Since solution quality of MAC and ACS are satisfactory, both algorithms constitute a further step forward in the design of fast image retrieval systems.

### 4.6.4 Similarity of Chemical Compounds

The most prevalent approaches to measure structural similarity of chemical graphs are premised on the maximum common induced subgraph [294]. These techniques calculate similarities that are intuitive and admit easy visualization of the common substructures [59, 349]. Historically, measures based on the maximum common induced subgraph have suffered from its computational intractability.

Because of its good time performance in previous experiments, we evaluated ACS on the maximum common induced subgraph problem for chemical compounds. For this purpose we used the MUTAGENESIS dataset described in Appendix B.2.2. The dataset consists of 230 chemical compounds. Each compound was transformed to an attributed graph. Atoms were represented by vertices, and bonds by edges. We colored each vertex with the element symbol of the corresponding atom. Type and electrical charge of an atom have not been considered. Edges were colored with the type of bond between the corresponding atoms. We considered all possible pairwise comparisons between two distinct structures of the dataset, giving a total of $26,335$ comparisons.

### *Selected Algorithms*

We compared the performance of the ACS with the performance of the following metaheuristics:

| Acronym | Method | Reference |
|---|---|---|
| TAB | Reactive Tabu Search | [21] |
| REP | Replicator Equations | [280] |
| SA | Simulated Annealing | [146] |
| SD | Steepest Descent | [167] |

The selection of algorithms with which to compare the ACS was motivated by the following factors:

1. Since long computation times are one reason for less attention to structural similarity measures in chemistry literature, the selected algorithms should be fast.

2. *Reactive Tabu Search* proposed by Battiti & Tecchiolli [21] is one of the most successful metaheuristics for the maximum clique problem. The algorithm is selected to serve as a benchmark in regard to solution quality.

3. *Replicator Equations* were first applied to the MCP by Pelillo [284]. The replicator dynamics solves the Motzkin-Strauss Formulation of the MCP. One attractive feature of this approach is that it implements hard constraints instead of soft constraints via penalty terms. In addition, a search is performed in a state space of one dimension less than in that of ACS. Details of the algorithm and its context are presented in Appendix A.2.1.
We selected the Replicator algorithm because of its simplicity, effectiveness, and the considerable attention it is given in the graph matching literature.

4. *Simulated Annealing* as suggested by Homer & Peinado [146] combines a simulated annealing and exterior point penalty method using a quenched annealing schedule for both control parameters, temperature and penalty. The algorithm is described in Appendix A.2.2.
The algorithm is selected, because it is fast and ranked among the best heuristics for maximum clique presented at the DIMACS challenge [191]. In addition, simulated annealing has been successfully applied to graphs with up to $10,000$ vertices and outperformed other competing algorithms.

5. *Steepest Descent* proposed by Jagota [167] is selected as a benchmark for what we should at least expect. The algorithm is described in Appendix A.1.3.

### *Evaluation Procedure*

The selected algorithms computed the cardinality of the maximum common induced subgraph for each pair of distinct chemical graphs from the MUTAGENESIS

dataset. For this, an association graph $Z$ is derived from a pair of given chemical graphs $X$ and $Y$ such that the common induced subgraphs of $X$ and $Y$ correspond to the cliques of $Z$.

For Reactive Tabu Search, we used the C++ implementation by [21].[9] The maximal CPU-time dedicated for each comparison executed by Tabu Search was set to 5 seconds. Since the other algorithms were implemented in Java using `JDK` 1.2 and measuring their time performance involved clock time, a fair comparison of time performances is only possible for ACS, REP, SA, and SD. All experiments were executed on a multi-server `Sparc SUNW Ultra-4`.

***Numerical Results***

Figure 4.11 depicts the performance profiles of the selected algorithms. Table 4.5 summarizes the results using absolute values.

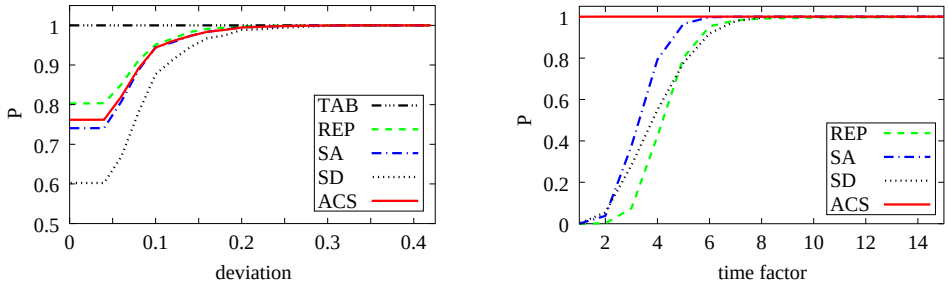
The performance profiles for the clique size metric shows that TAB always found the best solution. To assess the deviation of the average solution quality of TAB from the optimal solution, we randomly sampled a small subset of 100 chemical graph matching problems and solved the sampled problems with an exact algorithm. In all 100 trials, TAB returned the global optimal solution. Hence, we assume that TAB has found the optimal solution in almost all $26,335$ trials.

Solution quality of REP, SA, and ACS is roughly equivalent and superior to SD. As indicated by Figure 4.11(a), in about 75% of all trials, REP, SA, and ACS returned the same clique size as TAB. By our assumption of the performance of TAB, the percentage of optimal solutions found by REP, SA, and ACS is about the same order of magnitude. In average, as shown by Table 4.5, REP, SA, and ACS deviate only about 2% from the best solution.

With respect to time, ACS is the fastest algorithm in all trials. As shown in Table 4.5, REP, SA, and SD are about 4-5 times slower than ACS. Though time measurements are biased towards TAB due to a faster programming language and better time metric, ACS is still about 55 times faster than TAB.

To summarize, ACS is competitive with other fast heuristics regarding to solution quality and exhibits the best trade-off between accuracy and speed.

9. The implementation is available at `http://rtm.science.unitn.it/intertools/clique/` (3/31/2004).




(a) Profiles for size of clique

(b) Profiles for clock time

**Figure 4.11** Results of 26,335 pairwise comparisons of distinct chemical graphs from the MUTAGENESIS dataset. Shown are the performance profiles for (a) clique size and (b) clock time. Performance profile of TAB for the time metric are excluded, because the time performance of TAB is measured by CPU-time.

| | Clique size | | Time (*msec*) | | | | |
|---|---|---|---|---|---|---|---|
| | avg | dev | min | max | avg | fac | total |
| ACS | 13.12 | 0.02 | 1 | 775 | 89.5 | 1.0 | 39’ |
| TAB | 13.45 | 0.00 | 5 000 | 5 000 | 5 000.0 | 55.9 | 36h34’ |
| REP | 13.18 | 0.02 | 4 | 7 750 | 433.8 | 4.8 | 3h10’ |
| SA | 13.12 | 0.02 | 3 | 4 263 | 364.8 | 4.1 | 2h40’ |
| SD | 12.87 | 0.04 | 2 | 5 976 | 469.3 | 5.2 | 3h43’ |

**Table 4.5** Shown are absolute results of the algorithms for comparison of chemical graphs from the MUTAGENESIS dataset. The third column with identifier *dev* shows the deviation $1 - avg/avg^*$ of the average clique size *avg* of an algorithm from the best average clique size $avg^* = 13.45$. Similarly, the seventh column with identifier *fac* shows the ratio $avg/avg^*$ of the average time performance *avg* of an algorithm and the best average time performance $avg^* = 89.5$. The other columns are self-explanatory. Note that TAB is implemented in C++ and all other algorithms in Java. In addition, the times for TAB are CPU-times, whereas the other time data refer to clock time.

## 4.7 Conclusion

This chapter presented a novel technique for solving combinatorial optimization problems using Hopfield networks. To develop this technique, we derived a computationally simple procedure to iteratively adapt the optimal system parameters to the current active subspace.

In experiments, we found three advantages of the proposed system: first, as opposed to almost all other techniques, the system requires no tuning of system parameters; second, the system is fast and scales well with increasing problem size; and third, compared to other solvers, the system exhibits a better speed-accuracy trade-off.

In conjunction with Chapter 3, we now have a manageable framework to solve a broad range of graph matching problems. Together with a solver that requires no tuning of system parameters, the generic approach via the MWCP makes the framework easily accessible to the unskilled practioner.

We conclude this chapter with a threefold outlook. The scene has been set for generic solutions to a wide variety of graph matching problems. It is beyond the scope of this work to devise and explore powerful solutions for the MWCP from other domains including, for example, nonlinear and quadratic programming.

Neural network models exploiting the new technique of selective attention is not confined to solving clique problems. In Chapter 7, we present another selective attention model for distance-based classification problems.

Finally, investigations on large scale problems using ACS applying the principle of controlled forgetting have not been considered here. We conjecture that careful deleting of passive units makes ACS attractive as a fast solver for the MWCP on graphs of order larger than $10,000$.

# 5 Weighted Graph Isomorphism

In 1996, Fortin [96] claimed that neural network solutions to the graph isomorphism problem *cannot be used unmodified in a practical setting*. Though considerable improvements have been made since the above statement was published, Fortin's claim still remains true. Comparing accuracy and execution times reveals that contemporary neural network solutions are not competitive with standard solutions to the graph isomorphism problem.

We devote this chapter the proposition of a two-stage neural network solution that

- is competitive with the most powerful solutions to standard graph isomorphism
- outperforms other approaches to inexact graph isomorphism problems.

The proposed solution is based on a neural refinement procedure to reduce the search space, followed by an energy-minimizing matching process. Experiments on chemical, random, and random weighted graphs with up to $5,000$ vertices are presented and discussed.

## 5.1 Introduction

The graph isomorphism problem (GIP) consists of deciding whether two given graphs are structurally equivalent, i.e. whether there is a bijective mapping from the vertices of one graph to the vertices of the second graph such that the adjacency relationship is preserved. The problem has received considerable attention because of its practical implications and unresolved complexity status.

Applications include e.g. the identification of isomorphic molecular structures in chemistry [2, 211, 297, 390], scene analysis [10], detection of kinematic chains [292], optimal routing of messages in multistage interconnecting networks [69], or the construction and enumeration of combinatorial configurations [64].

Theoretical interest in the GIP originates from the persistent difficulty in characterizing its computational complexity. The GIP is still unsolved in the sense that there is neither an NP-completeness proof, nor has an efficient algorithm with polynomial complexity yet been found. For an exposition on the complexity of the GIP and its relative position within the class NP see [212].

The *inexact graph isomorphism problem* is a straightforward extension of the GIP.

Inexact graph isomorphism asks for structural equivalence accepting small deviations in the vertex and edge attributes. Since inexact isomorphism generalizes the standard GIP, it is of at least the same complexity. Inexact GIP finds applications in computer vision [290, 291].

Despite its practical and theoretical importance, neural networks and related heuristics are not competitive with standard solutions to the GIP. Even the most powerful approaches proposed by Pelillo [280, 281] and Rangarajan et al. [290, 291] require a prohibitive amount of time and are too erroneous on graphs with only 100 vertices. Even if we are willing to accept a small degree of uncertainty, neural solutions to the GIP are useless in a practical setting.

The situation is much better for inexact GMPs. On random weighted graphs, neural network solutions like the *Lagrangian Relaxation Network* [291] and the *Optimizing Network Architecture* [290] are significantly superior than other approaches such as the eigendecomposition approach [363], the polynomial transform approach [8], and the linear programming approach [9]. Nevertheless, the execution times of all these algorithms are still prohibitively slow since they do not exploit the structure of the graphs under consideration.

The aim of this chapter is twofold: first, we want to make neural network solutions competitive to efficient standard solutions for exact GMPs; second, we want to further improve the performance of neural networks for inexact GMPs. For this purpose, we first extend the well-known concept of vertex invariants for exact graph isomorphisms to their inexact counterpart. To assess the effects of vertex invariants, we suggest a novel representation of the search space in terms of perfect matchings of a bipartite graph. Next, we propose a two-stage neural graph isomorphism algorithm for exact and inexact GIPs. In a preprocessing step, a neural refinement procedure classifies the vertices of both graphs under consideration into subsets of structural similar vertices. In a second step, the ACS algorithm performs an isomorphism test exploiting the information from the previous stage. We assessed the effectiveness of the proposed approach on graphs with 100-5000 vertices and on chemical molecules.

The remainder of this chapter is organized as follows: The next Section presents a statement of the problem and introduces basic definitions. In Section 5.3, we provide a representation of the search space in terms of bipartite graphs. Section 5.4 reviews vertex invariants. In Section 5.5, we extend the framework of vertex invariants to inexact graph isomorphism. Section 5.7 presents and discusses experiments, and finally, Section 5.8 summarizes the results and presents an outlook on further research.

## 5.2 Inexact Graph Isomorphism

Throughout this chapter, we assume that $X$ and $Y$ are weighted graphs from $\mathcal{G}_{\mathbb{R}}$ with adjacency matrices $\boldsymbol{X} = (x_{ij})$ and $\boldsymbol{Y} = (y_{ij})$. The null attribute is defined by $\epsilon = 0$. We refer to $X$ as the model and to $Y$ as the data graph. An important concept to define inexact graph isomorphisms is the *noise tolerance threshold* $\tau \geq 0$.

The noise tolerance $\tau$ controls to which extent vertex and edge attributes of the data graphs can be regarded as noisy copies of the vertex and edge attributes of the model graph.

We call $X$ and $Y$ *$\tau$-isomorphic*, if there exists a bijection

$$\phi : V(X) \to V(Y), \quad i \mapsto i^\phi$$

with

$$\|\boldsymbol{x_i} - \boldsymbol{y}_{i^\phi}\|_\infty \leq \tau$$

for all items $\boldsymbol{i} \in I(X)$. Such a mapping $\phi$ is called a *$\tau$-isomorphism*. By $\mathcal{I}^\tau_{X,Y}$ we denote the set of all $\tau$-isomorphisms $\phi : V(X) \to V(Y)$. The *graph $\tau$-isomorphism problem* lies in deciding whether two given graphs are $\tau$-isomorphic. For $\tau = 0$ we obtain the standard definition of graph isomorphism.

Note that the definition of a $\tau$-isomorphism allows mapping between edges with attribute $\boldsymbol{a} \in \mathcal{A}$ of one graph and non-edges of the other graph if $\|\boldsymbol{a}\|_\infty \leq \tau$.

Clearly, the graph $\tau$-isomorphism problem lies in class NP. Since it includes the graph isomorphism problem, graph $\tau$-isomorphism is at least of the same complexity. Therefore, we suspect that any algorithm for graph $\tau$-isomorphism is inefficient unless it is shown that graph $\tau$-isomorphism is in class P.

In the remainder of this section, we introduce some basic definitions and notations for later use in our theoretical considerations.

### *Basic Definitions and Notations*

*Covers and Partitions.*
A family $\mathcal{C} = \{V_1, \ldots, V_k\}$ of nonempty subsets $V_i \subseteq V$ with $V = \bigcup_i V_i$ is called a *cover* of $V$. The elements of a cover are its *clusters*. A *partition* $\mathcal{P}$ of a set $V$ is a cover of $V$ whose members are pairwise disjoint. The elements of a partition are usually called its *cells*. We say a cover $\mathcal{C}$ of $V$ is *finer* than a cover $\mathcal{C}'$ of $V$, written as $\mathcal{C} \prec \mathcal{C}'$, if every cluster of $\mathcal{C}$ is a subset of some cluster of $\mathcal{C}'$. Under these conditions, $\mathcal{C}'$ is *coarser* than $\mathcal{C}$.

*Exact and Inexact Automorphisms.*
A *$\tau$-automorphism* of $X$ is a $\tau$-isomorphism from $X$ to itself. By $\mathrm{Aut}_\tau(X)$, we denote the set of all $\tau$-automorphisms of $X$. Two vertices $i, j \in V(X)$ are *$\tau$-similar*, written as $i \sim_\tau j$, if there exists a $\tau$-automorphism $\phi$ with $i^\phi = j$. The *$\tau$-automorphism cover* $\mathcal{C}(X)$ of $X$ is a cover of $V(X)$ induced by $\sim_\tau$. For $\tau = 0$, $\mathrm{Aut}(X)$ is a permutation group. The relation $\sim$ is an equivalence relation, which induces an *automorphism partition* $\mathcal{P}_X$ of $X$. A cell of $\mathcal{P}_X$ is called *orbit*.

*Bipartite Graphs and Matchings.*
A graph $X$ is called *bipartite* if $V(X)$ admits a partition into two cells $(U, U')$ such that $E(X) \subseteq U \times U' \cup U' \times U$. Hence, vertices in the same cell must not be adjacent.

The bipartite graph $X$ is called *complete* if $E(X) = U \times U' \cup U' \times U$. A complete bipartite graph with partition cells $(U, U')$ is also denoted by $K_{n,n'}$, where $n = |U|$ and $n' = |U'|$ are the number of vertices of its partition cells. A bipartite graph with partition cells $(U, U')$ is *balanced* if $|U| = |U'|$.

A *matching* of a graph $X$ is a subset $M \subseteq E(X)$ such that no two different edges are incident with a common vertex. A *perfect matching* is a matching that covers all vertices of $X$. It is noteworthy that the terms *matching* of a graph and *graph matching* have different meanings. It will be clear from the context which of the terms we refer to so that no confusion should arise.

*Direct Sum of Graphs.*
Let $X$ and $Y$ be graphs of order $|X| = n$ and $|Y| = m$. The *direct sum* $Z = X \oplus Y$ is a graph with vertex set $V(Z) = V(X) \dot{\cup} V(Y)$, edge set $V(E) = E(X) \dot{\cup} E(Y)$, and adjacency matrix

$$\boldsymbol{Z} = \begin{pmatrix} \mathbf{0}_{n,m} & \boldsymbol{X} \\ \boldsymbol{Y} & \mathbf{0}_{m,n} \end{pmatrix},$$

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are the adjacency matrices of $X$ and $Y$.

*Notation for Exact and Inexact Concepts.*
Like $\tau$-isomorphism or $\tau$-automorphism, a $\tau$-concept is called *exact concept* if $\tau = 0$, and *inexact concept* otherwise. For $\tau = 0$, we omit the prefix "$\tau$-" and write isomorphism, automorphism, and so on.

## 5.3 Representation of the Search Space

An appropriate representation of the search space is essential to devise efficient solutions to the given problem and to gain insight into its structure. In this section, we focus on the latter issue and represent the search space in terms of perfect matchings of a bipartite graph. The chosen representation is useful to analyze the effects of vertex invariants.[1]

Assume that $X$ and $Y$ are attributed graphs of order $n$. The intrinsic search space of graph $\tau$-isomorphism is the set $\mathcal{S}_{XY}$ of all $n!$ bijections $\phi : V(X) \to V(Y)$. Any algorithm for solving the graph $\tau$-isomorphism problem of $X$ and $Y$ tries to find a bijection $\phi \in \mathcal{S}_{XY}$ that satisfies the properties of a $\tau$-isomorphism. Reducing the search space aims at identifying potential candidates for $\tau$-isomorphism that satisfy predetermined necessary conditions.

Closely related to $\mathcal{S}_{XY}$ is the set $\mathcal{S}_n$ of all permutations acting on $[1 : n]$.

1. Vertex invariants are introduced in Section 5.4.

Given bijective labelings $\nu_X$ on $V(X)$ and $\nu_Y$ on $V(Y)$, we obtain the following commutative diagram

$$\begin{array}{ccc} V(X) & \xrightarrow{\phi} & V(Y) \\ \downarrow{\scriptstyle \nu_X} & & \downarrow{\scriptstyle \nu_Y} \\ [1:n] & \xrightarrow{\pi} & [1:n] \end{array} \tag{5.1}$$

for all $\phi \in \mathcal{S}_{XY}$. Since all mappings involved in diagram (5.1) are bijective, we can identify the vertex mappings $\phi \in \mathcal{S}_{XY}$ with permutations $\pi \in \mathcal{S}_n$ via $\phi = \nu_Y^{-1} \circ \pi \circ \nu_X$. Hence, the set $\mathcal{S}_n$ is an alternative representation of the intrinsic search space $\mathcal{S}_{XY}$.

A representation of the search space in terms of the set $\mathcal{S}_n$ is attractive because $\mathcal{S}_n$ forms a group under multiplication (i.e. composition of mappings) called *symmetric group*. This in turn links the graph isomorphism problem to the theory of permutation groups, a basic concept in algebra and combinatorics. This branch in mathematics provides powerful tools, which have been successfully applied to graph isomorphism and related problems. But we can no longer apply these techniques to more general graph matching problems including graph $\tau$-isomorphism problems.

For studying the effects of vertex $\tau$-invariants, we replace the search space $S_{XY}$ with the set $\mathcal{S}_{XY}^p \supseteq S_{XY}$ of all partial bijections from $V(X)$ to $V(Y)$. Reducing the search space then aims at discarding partial bijections from $\mathcal{S}_{XY}^p$, which violate necessary conditions to establish $\tau$-isomorphism.

It is useful to represent the set $\mathcal{S}_{XY}^p$ as a complete bipartite graph $K_{n,n}$. The bipartition $\big([1:n],[1:n]\big)$ of $K_{n,n}$ represents the vertices of $X$ and $Y$ via $\nu_X^{-1}$ and $\nu_Y^{-1}$. Edges $(i,j)$ of $K_{n,n}$ represent bijections $\phi : \{i\} \to \{j\}$ of singletons. Matchings of $K_{n,n}$ bijectively correspond to partial bijections from $\mathcal{S}_{XY}^p$ and perfect matchings to total bijections from $\mathcal{S}_{XY}$. Thus, the set of perfect matchings of $K_{n,n}$ is an equivalent representation of the search space $S_{XY}$.

Discarding partial bijections from $\mathcal{S}_{XY}^p$ is equivalent to deleting matchings in $K_{n,n}$. The resulting graph is a bipartite graph with hopefully less cardinality of perfect matchings. Note that discarding edges from an arbitrary bipartite graph does not necessarily decrease the number of perfect matchings.

Determining the complexity of the reduced search space comes down to the problem of enumerating perfect matchings of a bipartite graph. This problem is generally at least as hard as any NP-complete problem [365]. Kasteleyn [203] introduced an elegant technique to enumerate perfect matchings in planar graphs. His approach roughly reduces the problem to the computation of a determinant, and has been extended to a broad class of graphs, the so-called *Pfaffian graphs* [92]. Though it has been shown that Pfaffian bipartite graphs do not comprise the set of all balanced bipartite graphs, the basic concept of Kasteleyn's method, the *permanent* of a matrix, is still useful to bound the number of perfect matchings in a bipartite graph.

The *permanent* of an $(n \times n)$-matrix $\boldsymbol{A} = (a_{ij}) \in \mathbb{R}^{n \times n}$ is defined by

$$\mathrm{per}(\boldsymbol{A}) = \sum_{\phi \in \mathcal{S}_n} \prod_{i=1}^{n} a_{ii^\phi}.$$

Note that the permanent is the determinant without alternating sign. Though the permanent seems to be similar to the determinant, its computation – as opposed to the determinant is hard [66].

Now we want to relate the permanent to the number of perfect matchings of a bipartite graph. Assume that $B$ is a bipartite graph of even order $2n$ with balanced bipartition $\big(U, U'\big)$. The *biadjacency matrix* $\boldsymbol{B} = (b_{ij})$ of $B$ is an $(n \times n)$-matrix of the form[2]

$$b_{ij} = \begin{cases} 1 & : \text{ if } (i,j) \in E(B) \cap (U \times U') \\ 0 & : \text{ otherwise.} \end{cases}$$

Then the permanent of $\boldsymbol{B}$ counts the number of perfect matchings in $B$: each perfect matching corresponds to a bijection from $U$ to $U'$. Such a bijection corresponds to a permutation $\phi \in \mathcal{S}_n$ with

$$b_{11^\phi} = \cdots = b_{nn^\phi} = 1.$$

## 5.4 Exact Vertex Invariants

An exact vertex invariant is a property of vertices that is preserved under any isomorphism. For exact isomorphism problems, the concept of the vertex invariant is a practical technique for reducing the search space. In this section, we briefly review vertex invariants and propose a theoretical framework for analyzing their effects. The aim is to establish a comprehensive foundation for inexact vertex invariants.

Let $\mathcal{G}[V] = \{(X,i) : X \in \mathcal{G}_\mathcal{A},\, i \in V(X)\}$ be the set of all pairs $(X,i)$ consisting of an attributed graph $X$ and a vertex $i$ of $X$. A function $f : \mathcal{G}[V] \to \mathbb{R}^p$ is an *exact vertex invariant* if

$$\phi \in \mathcal{I}_{X,Y} \;\Rightarrow\; f(X,i) = f(Y,i^\phi)$$

for all $i \in V(X)$. We call the vector $f(X,i)$ an *exact invariant of vertex* $i$. For the sake of simplicity, we omit the term *exact* when it is clear from the context that $\tau = 0$. To illustrate the concept of a vertex invariant, let us consider some simple examples.

***Example 5.1***

2. Note that the biadjacency matrix $\boldsymbol{B}$ is not the adjacency matrix $\boldsymbol{A}$ of $B$. The matrix $\boldsymbol{B}$ is a square-matrix representation of the strict upper triangular matrix of $\boldsymbol{A}$. The difference becomes evident by observing that $\boldsymbol{B}$ is an $(n \times n)$-matrix and $\boldsymbol{A}$ a $(2n \times 2n)$-matrix.

Let $X$ be a graph and let $i$ be a vertex of $X$.

1. *Degree of a vertex:* The best known and most frequently used vertex invariant is the degree of a vertex.

2. *Longest path:* The longest path invariant of vertex $i$ is the length of a longest path starting at $i$.

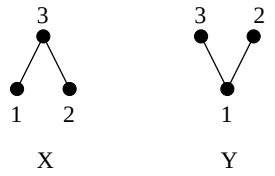
3. *$k$-Paths:* The $k$-path invariant assigns each vertex the number of vertices reachable along a path of length $k$.

4. *$k$-cliques:* The $k$-cliques invariant of vertex $i$ is the number of different cliques of size $k$ that contain $i$.

5. *Distance sequence:* The distance sequence invariant of vertex $i$ is an $n$-dimensional vector $\boldsymbol{p}$ where $p_k$ is the $k$-path invariant of $i$ for all $k \in [1:n]$.

Note that 1-4 are examples of one-dimensional vertex invariants. The distance sequence is an example of an $n$-dimensional vertex invariant. Further higher dimensional vertex invariants can be constructed by combining several one-dimensional vertex invariants.

Since vertex invariants are preserved under isomorphisms, only vertices with the same invariant can be mapped onto each other. We can therefore restrict our permutations from $\mathcal{S}_n$ to those that correspond to invariant preserving bijections from $\mathcal{S}_{XY}$. To illustrate the effects of vertex invariants, consider the graphs $X$ and $Y$ depicted in Figure 5.1. Assume that we use the degree as a vertex invariant. Since both graphs are of order $n = 3$, the original search space $\mathcal{S}_n$ is of order $n! = 6$. Any isomorphism between $X$ and $Y$ maps vertices of the same degree onto each other. Hence, the search space reduces to 2 permutations, each of which is an isomorphism. Both permutations are shown in Figure 5.1.




permutations

| X – Y | X – Y |
|---|---|
| 1 – 3 | 1 – 2 |
| 2 – 2 | 2 – 3 |
| 3 – 1 | 3 – 1 |

**Figure 5.1** Example of isomorphic graphs $X$ and $Y$. The degree invariant reduces the search space from 6 to 2 permutations. In this particular example, both permutations, which are shown in the table at the right, are also isomorphisms from $X$ to $Y$.

The following remarks are important.

- Consider two vertices with the same invariant. Then there is not necessarily an isomorphism mapping both vertices onto one another. Figure 5.2 provides an example. Both graphs $X$ and $Y$ have three vertices with degree 1, two vertices with degree 2, and one vertex with degree 3. Hence, there exists a bijection between $V(X)$ and $V(Y)$, mapping vertices of the same degree onto each other. Since $X$ and $Y$ are not isomorphic, none of these bijections is an isomorphism.

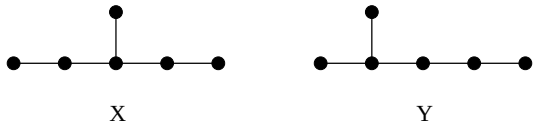



**Figure 5.2** Example of non-isomorphic graphs $X$ and $Y$ for which bijective mappings $\phi : V(X) \to V(Y)$ that preserve the degree of the vertices exist.

- The use of vertex invariants is considered to be a black art, because vertex invariants not always result in any decrease of the search space. A simple example are regular graphs of the same order and the same degree.[3] Since each vertex of a regular graph has the same degree, the degree invariance does not lead to a reduced search space. The art of using vertex invariants consists in composing appropriate invariants for a particular class of graphs, which are simple to compute.
- The previous item requires invariants, which are simple to compute. The reason is that many isomorphism problems can be solved directly in less time than it takes to determine more powerful invariants such as, for example, the $k$-cliques invariant.

Next, we determine to which extent vertex invariants reduce the search space. Though it is straightforward to asses the effects of exact invariants, we follow a different approach. The framework presented here serves to set the foundation for analyzing inexact vertex invariants.

A vertex invariant $f$ induces a balanced bipartite graph $B = X \star_f Y$ with bipartition $\big(V(X), V(Y)\big)$ and edge set

$$E(B) = \Big\{(i,j) \in V(X) \times V(Y) \,:\, f(X,i) = f(X,j)\Big\}.$$

Each edge in $B$ connects a vertex of $X$ with a vertex of $Y$ if and only if both vertices have equal invariant. We color the vertices $i$ of $B$ by

$$f_i = \begin{cases} f(X,i) & : \text{ if } i \in V(X) \\ f(Y,i) & : \text{ if } i \in V(Y). \end{cases}$$

Then the colors partition the vertex set $V(B)$ into a set $\mathcal{P}_f(X,Y)$ of equivalence classes with

$$i,j \in U \;\Leftrightarrow\; f_i = f_j$$

for all *cells* $U \in \mathcal{P}_f(X,Y)$. We call $B$ a *bipartite association* of $X$ and $Y$ induced

3. A regular graph is a graph $X$ with $\delta(X) = \Delta(X)$. Regular graphs of the same order and degree are not necessarily isomorphic. For example, cubic graphs of order $n = 18$ have $41,301$ isomorphism classes [211].

by the vertex invariant $f$. The associated partition $\mathcal{P}_f(X,Y)$ of $V(B)$ is called *invariant partition.*

A necessary condition for $X$ and $Y$ to be isomorphic is that any bipartite association of $X$ and $Y$ can be decomposed into disjoint balanced bipartite graphs induced by the cells of its invariant partition.

***Proposition 5.1***
*Let $X$ and $Y$ be attributed graphs of order $n$, and let $B = X \star_f Y$ be a bipartite association induced by a vertex invariant $f$. If $X$ and $Y$ are isomorphic, then*

*(BG1)* $\displaystyle B = \bigoplus_{U \in \mathcal{P}_f(X,Y)} B[U]$

*(BG2)* $U \in \mathcal{P}_f(X,Y) \Rightarrow U \simeq K_{m,m}$, *where* $\displaystyle m = \frac{|U|}{2}$

***Proof*** Let $\mathcal{P} = \mathcal{P}_f(X,Y)$. We first show (BG1).

(BG1): Since $\mathcal{P}$ is a partition of the vertex set $V(B)$, we have

$$V(B) = \dot{\bigcup_{U \in \mathcal{P}}} U.$$

It is sufficient to show that no edge connects vertices from different cells. Let $i \in U$ and $j \in U'$ be vertices from distinct cells $U, U' \in \mathcal{P}$. Then $f_i \neq f_j$ and therefore, $(i,j) \notin E(B)$.

(BG2): Consider the cell $U \in \mathcal{P}$. Let $\pi_X : U \to V(X)$ and $\pi_Y : U \to V(Y)$ be the projections of $U$ to the partition cells of $B[U]$. Since $X$ and $Y$ are isomorphic and $f$ is a vertex invariant, we have $|\pi_X(U)| = |\pi_Y(U)|$. Thus, $B[U]$ is a balanced bipartite graph with bipartition $\big(\pi_X(U), \pi_Y(U)\big)$. Moreover, $B[U]$ is complete bipartite, because all vertices of $B[U]$ have the same color. Finally, set $m = \pi_X(U)$. Then we have $|U| = 2m$ and $B[U] \simeq K_{m,m}$. ■

According to (BG1), the induced subgraphs $B[U]$ induced by the cells $U \in \mathcal{P}_f(X,Y)$ are the components of $B$. In addition, from (BG2), it follows that each component $B[U]$ of $B$ is a complete bipartite graph isomorphic to $K_{m,m}$.

Given the partition $\mathcal{P}_f(X,Y)$ of the bipartite association $B$, we can precisely specify the complexity of the search space.

***Proposition 5.2***
*Let $B = X \star_f Y$ be a bipartite association of $X$ and $Y$ satisfying (BG1) and (BG2). Then*

$$\operatorname{per}(\boldsymbol{B}) = \prod_{U \in \mathcal{P}_f(X,Y)} \left(\frac{|U|}{2}\right)!\,, \tag{5.2}$$

*where $\boldsymbol{B}$ is the biadjacency matrix of $B$.*

***Proof*** Let $\mathcal{P} = \mathcal{P}_f(X,Y)$, and let $\mathcal{M}(Z)$ denote the set of perfect matchings of a given graph $Z$. According to (BG2), each component of $B$ is an induced subgraph

isomorphic to a complete bipartite graph $K_{m,m}$. Thus, each component has $m!$ perfect matchings. Due to (BG1), a perfect matching $M$ of $B$ is of the form

$$M = \bigcup_{U \in \mathcal{P}} M_U,$$

where $M_U = M \cap E(B[U])$ is a perfect matching of $B[U]$. The set $\mathcal{M}(B)$ is of the form

$$\mathcal{M}(B) = \left\{ \bigcup_{U \in \mathcal{P}} M_U \,:\, M_U \in \mathcal{M}(B[U]) \right\}.$$

Hence, there are

$$\prod_{U \in \mathcal{P}_f(X,Y)} |M(B[U])| = \prod_{U \in \mathcal{P}_f(X,Y)} \left(\frac{|U|}{2}\right)!$$

possible ways to combine perfect matchings of the components to a perfect matching of $B$. The assertion follows from $\mathrm{per}(\boldsymbol{B}) = |\mathcal{M}(B)|$. ■

Proposition 5.2 is interesting for two reasons: first, it tells us that a vertex invariant $f$ reduces the complexity of the search space $S_n$ from $n!$ to the permanent $\mathrm{per}(\boldsymbol{B})$ of the bipartite association $B = X \star_f Y$ determined by $f$; second, equation (5.2) indicates how to reduce the search space $S_n$.

Since an invariant partition decomposes the vertex set of a bipartite association $B$ into disjoint subsets, the permanent $\mathrm{per}(\boldsymbol{B})$ becomes smaller the finer an invariant partition is. The granularity of an invariant partition depends on the chosen vertex invariant. Thus, reducing the search space using vertex invariants aims at refining the invariant partition. Refinement of an invariant partition is bounded by the automorphism partition of $X$ and $Y$.

***Proposition 5.3***
*Let $B = X \star_f Y$ be a bipartite association of $X$ and $Y$ satisfying (BG1) and (BG2). Then*

$$\mathrm{per}(\boldsymbol{B}) \geq \prod_{U \in \mathrm{Aut}(X)} |U|! = \prod_{U \in \mathrm{Aut}(Y)} |U|!\,,$$

*where $\boldsymbol{B}$ is the biadjacency matrix of $B$.*

Before presenting a proof, the following comments might be helpful:

- Observe that the cells $U$ in Proposition 5.3 are subsets of $V(X)$ and $V(Y)$, respectively, whereas $U$ in Proposition 5.2 refers to subsets of $V(B)$. This explains cancellation of the factor $1/2$ in Proposition 5.3.
- The first product in Proposition 5.3 runs over all cells of the automorphism partition of $X$ and the second product is taken over all cells of the automorphism

partition of $Y$. Thus, Proposition 5.3 states that isomorphic graphs yield the same products. In the proof we show a stronger relationship between the automorphism partitions of isomorphic graph.

Now let us turn to the proof of Proposition 5.3.

***Proof*** Let $\mathcal{P}_f(X,Y)$ be the invariant partition of $V(B)$. Projections of cells $U \in \mathcal{P}_f(X,Y)$ to the partition cells $V(X)$ and $V(Y)$ of $B$ induce partitions $\mathcal{P}_f(X)$ of $V(X)$ and $\mathcal{P}_f(Y)$ of $V(Y)$. The partitions $\mathcal{P}_f(X)$ and $\mathcal{P}_f(Y)$ have the following properties:

1. $\big|\mathcal{P}_f(X)\big| = \big|\mathcal{P}_f(Y)\big|$
2. $i,j \in U \;\Rightarrow f(X,i) = f(Y,i)$ for all $U \in \mathcal{P}_f(X)$
3. for all $U \in \mathcal{P}_f(X)$ there is a $U' \in \mathcal{P}_f(Y)$ with $f_U = f_{U'}$,

where $f_U = f_{U'}$ denotes the unique color of the vertices from the cells $U \in \mathcal{P}_f(X)$ and $U' \in \mathcal{P}_f(Y)$. Conversely, any pair of partitions $\mathcal{P}_X$ of $V(X)$ and $\mathcal{P}_Y$ of $V(Y)$ satisfying (1)-(3) for some vertex invariant $f$ induces an invariant partition $\mathcal{P}_f(X,Y)$ of $V(B)$. Hence, we have

$$\operatorname{per}(\boldsymbol{B}) = \prod_{U \in \mathcal{P}_f(X)} |U|! = \prod_{U \in \mathcal{P}_f(Y)} |U|!.$$

Since any vertex invariant $f$ is constant on the orbits of $\mathrm{Aut}(X)$ and $\mathrm{Aut}(Y)$, the automorphism partitions $\mathcal{P}(X)$ and $\mathcal{P}(Y)$ are finer than $\mathcal{P}_f(X)$ and $\mathcal{P}_f(Y)$, respectively. Let $g$ be a vertex invariant that classifies vertices of a graph according to its automorphism partition. Then the automorphism partitions $\mathcal{P}(X)$ and $\mathcal{P}(Y)$ satisfy (1)-(3) and induce an invariant partition $\mathcal{P}_g(X,Y)$ on vertices of the bipartite association $X \star_g Y$. From the equivalence of invariant partitions $\mathcal{P}_f(X,Y)$ and the induced pairs $\big(\mathcal{P}_f(X), \mathcal{P}_f(Y)\big)$, it follows that $\mathcal{P}_g(X,Y)$ is the finest invariant partition. This proves the assertion. ■

## 5.5 Inexact Vertex Invariants

In this section, we extend the concept of vertex invariants to inexact vertex invariants for graph $\tau$-isomorphism problems. To analyze the effects of inexact vertex invariants, we generalize the framework of bipartite associations and study their permanents.

A function

$$f : \mathcal{G}[V] \to \mathbb{R}^p, \quad (X,i) \mapsto \big(f_1(X,i), \ldots, f_p(X,i)\big)$$

is a (p-dimensional) *vertex $\tau$-invariant*, if

$$\phi \in \mathcal{I}^{\tau}_{X,Y} \;\Rightarrow\; \big\|f(X,i) - f(Y,i^{\phi})\big\|_{\infty} \leq \tau$$

for all $i \in V(X)$. We call the vector $f(X,i)$ *$\tau$-invariant of $i$*. For $\tau = 0$ we obtain the standard notion of an exact vertex invariant.

The next Proposition 5.4 provides some examples of one-dimensional vertex $\tau$-invariants.

***Proposition 5.4***
*Let $X$ be a normalized graph from $\mathcal{G}_{\mathbb{U}}$ with adjacency matrix $\boldsymbol{X} = (x_{ij})$, let $i$ be a vertex of $X$, and let $\tau$ be a noise tolerance threshold. Then the following functions are vertex $\tau$-invariants:*

1. $f(X,i) = \dfrac{1}{n}\deg_{\mathrm{w}}(i)$
2. $f(X,i) = \|\boldsymbol{x}_i\|_\infty$

***Proof*** Let $Y$ be another normalized graph from $\mathcal{G}_{\mathbb{U}}$ with adjacency matrix $\boldsymbol{Y} = (y_{ij})$. Suppose that $X$ and $Y$ are $\tau$-isomorphic. Let $\phi \in \mathcal{I}^{\tau}_{X,Y}$ be a $\tau$-isomorphism, and let $r$ be a vertex of $Y$ with $r = i^{\phi}$.
***1.*** We have

$$\left|f(X,i) - f(Y,r)\right| = \frac{1}{n}\left|\deg_{\mathrm{w}}(i) - \deg_{\mathrm{w}}(r)\right| = \frac{1}{n}\left|\sum_{\substack{j=1\\ j\neq i}}^{n} x_{ij} - \sum_{\substack{s=1\\ s\neq r}}^{n} y_{rs}\right|$$

Since $\phi$ is a bijection we may rewrite the last equation to

$$\left|f(X,i) - f(Y,r)\right| = \frac{1}{n}\left|\sum_{\substack{j=1\\ j\neq i}}^{n} x_{ij} - y_{rj^{\phi}}\right|.$$

Applying the triangle inequality gives

$$\left|f(X,i) - f(Y,r)\right| \leq \frac{1}{n}\sum_{\substack{j=1\\ j\neq i}}^{n}\left|x_{ij} - y_{rj^{\phi}}\right|.$$

From the definition of a $\tau$-isomorphism follows

$$\left|x_{ij} - y_{rj^{\phi}}\right| = \left\|x_{ij} - y_{rj^{\phi}}\right\| \leq \tau.$$

Hence, we find

$$\left\|f(X,i) - f(Y,r)\right\|_{\infty} = \left|f(X,i) - f(Y,r)\right| \leq \tau.$$

This proves the first assertion.

***2.*** Let $\bar{x} = f(X,i)$ and $\bar{y} = f(Y,r)$. Then there are vertices $j \in V(X)$ and $s \in V(Y)$ such that $x_{ij} = \bar{x}$ and $y_{rs} = \bar{y}$. Since $\phi$ is bijective, we can find vertices $k \in V(X)$ and $t \in V(Y)$ with $k^{\phi} = s$ and $j^{\phi} = t$. From the definition of a $\tau$-isomorphism, it

follows that

$$|\bar{x} - y_{rt}| = |x_{ij} - y_{rt}| = \left|x_{ij} - y_{i^\phi j^\phi}\right| = \left\|x_{ij} - y_{i^\phi j^\phi}\right\| \leq \tau \tag{5.3}$$

and

$$|x_{ik} - \bar{y}| = |x_{ik} - y_{rs}| = |x_{ik} - y_{i^\phi k^\phi}| = \|x_{ik} - y_{i^\phi k^\phi}\| \leq \tau. \tag{5.4}$$

Our goal is to show

$$\left|f(X,i) - f(Y,r)\right| = |\bar{x} - \bar{y}| \leq \tau.$$

We distinguish three cases to this end:

*Case 1:* Assume that

$$(\bar{x} - \bar{y})(\bar{x} - y_{rt}) = 0.$$

This implies $\bar{x} = \bar{y}$ or $\bar{x} = y_{rt}$. The case $\bar{x} = \bar{y}$ is trivial. Suppose that $\bar{x} = y_{rt}$. From

$$x_{ik} \leq \bar{x} = y_{rt} \leq \bar{y}$$

together with (5.4) follows

$$|\bar{x} - \bar{y}| \leq |x_{ik} - \bar{y}| \leq \tau.$$

*Case 2:* Now assume that

$$(\bar{x} - \bar{y})(\bar{x} - y_{rt}) < 0.$$

From $y_{rt} \leq \bar{y}$ follows $y_{rt} < \bar{x} < \bar{y}$. In addition, we have $x_{ik} \leq \bar{x} < \bar{y}$. Hence, with (5.4) we obtain

$$|\bar{x} - \bar{y}| \leq |x_{ik} - \bar{y}| \leq \tau.$$

*Case 3:* Finally, assume that

$$(\bar{x} - \bar{y})(\bar{x} - y_{rt}) > 0.$$

We consider two further cases: (a) both terms are positive or (b) both terms are negative. If (a) holds, then $y_{rt} \leq \bar{y} < \bar{x}$ and therefore, by applying (5.3) we obtain

$$|\bar{x} - \bar{y}| \leq |\bar{x} - y_{rt}| \leq \tau.$$

If (b) holds, we have

$$x_{ik} \leq \bar{x} < y_{rt} \leq \bar{y}.$$

By using (5.4) we arrive at

$$|\bar{x} - \bar{y}| \leq |x_{ik} - \bar{y}| \leq \tau.$$

■

As for exact invariants, we want to assess to which extent a vertex $\tau$-invariant reduces the search space. For this, we extend the concept of bipartite association and replace associated invariant partitions by invariant covers.

A vertex $\tau$-invariant $f$ induces a *bipartite $\tau$-association* $B = X \star_f Y$. The bipartite graph $B$ is balanced with bipartition $\big(V(X), V(Y)\big)$ and edge set

$$E(B) = \Big\{(i,j) \in V(X) \times V(Y) \,:\, \big\|f(X,i) - f(Y,j)\big\|_\infty \leq \tau\Big\}.$$

Two vertices of $B$ are connected by an edge if and only if their $\tau$-invariants do not differ by more than $\tau$ with respect to the maximum norm. We color the vertices of $X$, $Y$, and $B$ by

$$f_i = \begin{cases} f(X,i) & : \text{ if } i \in V(X) \\ f(Y,i) & : \text{ if } i \in V(Y). \end{cases}$$

The colors partition the vertex sets $V(X)$, $V(Y)$, and $V(B)$ into covers $\mathcal{C}_f(X)$, $\mathcal{C}_f(y)$, and $\mathcal{C}_f(X,Y)$, called *invariant covers*. For a cluster $U$ of any of the invariant clusters, we find that

$$i,j \in U \;\Leftrightarrow\; \big\|f_i - f_j\big\|_\infty.$$

We say a subset of vertices $U$ is *invariant with respect to $f$* if $U$ is a subset of a cluster.

Suppose that $X$ and $Y$ are isomorphic. What makes analysis of vertex $\tau$-invariant more complicated than in the exact counterpart is that invariant covers do not behave as well as invariant partitions. Projections of clusters $U \in \mathcal{C}_f(X,Y)$ to the partition cells $V(X)$ and $V(Y)$ of $B$ are not necessarily clusters of $\mathcal{C}_f(X)$ and $\mathcal{C}_f(Y)$. As a consequence, an invariant cover $\mathcal{C}_f(X,Y)$ does not necessarily satisfy (BG1) and (BG2) of Proposition 5.1. Hence, we can not use Proposition 5.2 to assess the complexity of the reduced search space. But what we can do is estimate the complexity of the search space by using a famous result, which was conjectured by Minc [263] and proven by Brègman [41].

***Theorem 5.1 (Minc-Brègman Theorem)***
*Let $\boldsymbol{A} \in \mathbb{B}^{n\times n}$ be a matrix where row $i$ sums to $r_i$ for all $i \in [1\!:\!n]$. Then*

$$\operatorname{per}(\boldsymbol{A}) \leq \prod_{i=1}^{n} (r_i!)^{\frac{1}{r_i}}. \tag{5.5}$$

***Proof*** [41]. ■

We can directly apply the Minc-Brègman Theorem to bound the complexity of the reduced search space.

***Corollary 5.1***
*Let $X$ and $Y$ be attributed graphs of order $n$, and let $B = X \star_f Y$ be a bipartite $\tau$-association induced by vertex invariant $f$. Then*

$$\operatorname{per}(\boldsymbol{B}) \leq \prod_{i=1}^{n} \big(\deg(i)!\big)^{\frac{1}{\deg(i)}}, \tag{5.6}$$

*where $\boldsymbol{B}$ is the biadjacency matrix of $B$.*

From inequality (5.6), it follows that the best we can do to reduce the search space is reduce the upper bound of the permanent $\operatorname{per}(\boldsymbol{B})$ by removing irrelevant edges. To provide a technique to remove irrelevant edges, we proceed as follows: first, we show in Proposition 5.5 that finer invariant covers of $V(X)$ and $V(Y)$ result in a finer invariant cover of $V(B)$. Next, Proposition 5.6 states that finer invariant covers have fewer edges. Finally, in Proposition 5.7, we show that the $\tau$-automorphism cover is the finest invariant cover.

***Proposition 5.5***
*Suppose that $X \star_f Y$ and $X \star_g Y$ are bipartite $\tau$-associations of $X$ and $Y$ induced by vertex $\tau$-invariants $f$ and $g$, respectively. From $\mathcal{C}_f(X) \prec \mathcal{C}_g(X)$ and $\mathcal{C}_f(Y) \prec \mathcal{C}_g(Y)$ follows $\mathcal{C}_f(X,Y) \prec \mathcal{C}_g(X,Y)$.*

***Proof*** Let $C \in \mathcal{C}_f(X,Y)$ be a cluster. We show that there is a cluster $C' \in \mathcal{C}_g(X,Y)$ with $C \subseteq C'$. Let $C_{\pi(X)}$ and $C_{\pi(Y)}$ be the projections of $C$ to the partition cells $V(X)$ and $V(Y)$ of $X \star_f Y$. Since $C$ is invariant with respect to $f$, the clusters $C_{\pi(X)}$ and $C_{\pi(Y)}$ are also invariant with respect to $f$. Hence, there are clusters $C_X \in \mathcal{C}_f(X)$ and $C_Y \in \mathcal{C}_f(Y)$ with $C_{\pi(X)} \subseteq C_X$ and $C_{\pi(Y)} \subseteq C_Y$. From $\mathcal{C}_f(X) \prec \mathcal{C}_g(X)$ and $\mathcal{C}_f(Y) \prec \mathcal{C}_g(Y)$, it follows that there are clusters $C'_X \in \mathcal{C}_g(X)$ and $C'_Y \in \mathcal{C}_g(Y)$ with $C_X \subseteq C'_X$ and $C_Y \subseteq C'_Y$. Hence, we have $C_{\pi(X)} \subseteq C'_X$ and $C_{\pi(Y)} \subseteq C'_Y$. This implies that $C$ is invariant with respect to $g$. Therefore, $C$ is either a cluster or a subset of a cluster of $\mathcal{C}_g(X,Y)$. This proves the assertion. ■

***Proposition 5.6***
*Let $B_f = X \star_f Y$ and $B_g = X \star_g Y$ be bipartite $\tau$-associations of $X$ and $Y$ induced by vertex $\tau$-invariants $f$ and $g$, respectively. From $\mathcal{C}_f(X,Y) \prec \mathcal{C}_g(X,Y)$ follows $E(B_f) \leq E(B_g)$.*

***Proof*** By definition of bipartite $\tau$-associations, we may identify the vertex sets of $B_f$ and $B_g$. Therefore, we may write $V = V(B_f) = V(B_g)$. Now let $i \in V$ be a vertex. It is sufficient to show that $N_f(i) \leq N_g(i)$, where $N_f(i)$ and $N_g(i)$ are the neighbors of $i$ in $B_f$ and $B_g$, respectively. Since $N_f^+(i) = N_f(i) \cup \{i\}$ is invariant with respect to $f$, there is a cluster $C \in \mathcal{C}_f(X,Y)$ with $N_f^+(i) \subseteq C$. By assumption,

$\mathcal{C}_f(X,Y)$ is finer than $\mathcal{C}_g(X,Y)$. Hence, there is a cluster $C' \in \mathcal{C}_g(X,Y)$ with $C \subseteq C'$. From $N_f^+(i) \subseteq C \subseteq C'$ follows $N_f^+(i) \subseteq N_g^+(i)$. This implies the assertion. ■

***Proposition 5.7***
*Let $X$ be a graph, and let $f$ be a vertex $\tau$-invariant. Then*

$$\mathcal{C}(X) \prec \mathcal{C}_f(X),$$

*where $\mathcal{C}(X)$ is the $\tau$-automorphism cover of $X$.*

***Proof*** Let $C \in \mathcal{C}(X)$ be a cluster of the $\tau$-automorphism cover. We show that there is a cluster $C' \in \mathcal{C}_f(X)$ with $C \subseteq C'$. Let $i,j \in C$. Then there is a $\tau$-automorphism $\phi$ with $i^\phi = j$. This implies $\|f(X,i) - f(X,j)\|_\infty \leq \tau$. Since $f$ is a vertex $\tau$-invariant, there is a cluster $C' \in \mathcal{C}_f(X)$ with $i,j \in C'$. Hence, we have $\mathcal{C}_G \prec \mathcal{C}_f(G)$. ■

Propositions 5.5, 5.6 and 5.7 tell us what we can do to reduce the search space for inexact graph isomorphism problems. Given two graphs $X$ and $Y$, we use vertex $\tau$-invariants to approximate the $\tau$-automorphism covers $\mathcal{C}(X)$ and $\mathcal{C}(Y)$. Finer invariant covers of $V(X)$ and $V(Y)$ induce a finer invariant cover of the vertices of the bipartite $\tau$-association. In turn, finer invariant covers of vertices of bipartite $\tau$-associations reduce the number of edges. According to the Minc-Brègman Theorem, fewer edges may lower the bound of the permanent as given in (5.6). A lower bound hopefully indicates a smaller permanent and therefore a less complex search space. The $\tau$-automorphism cover bounds to the extent we can reduce the search space using vertex $\tau$-invariants.

## 5.6 A Neural Graph Isomorphism Algorithm

The aim of this section is to propose a two-stage neural network approach to solve exact and inexact graph isomorphism problems. The first stage is a neural vertex classification procedure based on vertex invariants. In a second stage, the network tests for isomorphism.

Algorithm 4 outlines the basic form of a Neural Graph Isomorphism (NGI) algorithm. Here, NGI calls ACS (Attention Control System) to establish isomorphism or non-isomorphism. By $C$ we denote the clique returned by ACS.

**Algorithm 4** (NGI — Neural Graph Isomorphism Algorithm)

**Input:**
- $X$ – normalized graph of order $n$
- $Y$ – normalized graph of order $n$
- $\tau$ – noise tolerance threshold

**Procedure:**

*Neural Refinement Procedure*

approximate $\tau$-automorphism cover $\mathcal{C}(X)$

approximate $\tau$-automorphism cover $\mathcal{C}(Y)$

*Neural Isomorphism Test Procedure*

construct a $\tau$-association graph $X \diamond_\tau Y$

$C = \mathsf{ACS}(X \diamond_\tau Y)$

**Output:** YES if $|C| = n$, NO otherwise

Before describing the neural refinement and isomorphism test procedure of NGI, we introduce some notation to simplify technicalities.

All neural networks involved in NGI are Hopfield networks associated with a graph. Networks for approximating the $\tau$-automorphism covers $\mathcal{C}(X)$ and $\mathcal{C}(Y)$ are associated with $X$ and $Y$, respectively. The network for testing isomorphism is associated with an association graph $X \diamond_\tau Y$. For a given graph $Z$, a Hopfield network $\mathfrak{H}_Z$ associated with $Z$ consists of $|Z|$ fully connected units. For the sake of simplicity, we identify the units of $\mathfrak{H}_Z$ with the vertices of $Z$. The dynamics of $\mathfrak{H}_Z$ is of the form (4.1). For the sake of readability, we restate the update rule of $\mathfrak{H}_Z$. Using the same notations as in Section 4.2 and 4.3, we have

$$\begin{aligned} \boldsymbol{u}_{t+1} &= (1-d)\boldsymbol{u}_t + \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{h} \\ \boldsymbol{x}_{t+1} &= g\big(\boldsymbol{u}_{t+1}\big). \end{aligned}$$

### 5.6.1 A Neural Refinement Procedure

At any time $t$, the internal state $\boldsymbol{u}_t$ of a Hopfield model $\mathfrak{H}_Z$ associated with a graph $Z$ gives rise to a function

$$f_t : \mathcal{G}[V] \to \mathbb{R}, \quad (Z, i) \mapsto u_i(t). \tag{5.7}$$

We are interested in the conditions under which $f_t$ is a vertex $\tau$-invariant.

#### *Exact Graph Isomorphism*

Consider a Hopfield model $\mathfrak{H}_Z$ associated with a normalized graph $Z$. Theorem 5.2 specifies the weights, external inputs, and initial activation of $\mathfrak{H}_Z$ for which $f_t$ is an exact vertex invariant.

***Theorem 5.2***
*Let $Z = (V, E, \boldsymbol{Z})$ be a normalized graph with adjacency matrix $\boldsymbol{Z} = (z_{ij})$, let $\mathfrak{H}_Z$ be a Hopfield model associated with $Z$, and let $\vartheta : \mathbb{R} \to \mathbb{R}$ be a real-valued function. Assume that for all $i, j \in V$, the following properties hold:*

1. $w_{ij} = \vartheta(z_{ij})$,
2. $h_i = h_j$,
3. $u_i(0) = \vartheta(z_{ii})$.

*Then*

$$i \sim j \quad \Rightarrow \quad u_i(t) = u_j(t)$$

*for all $i, j \in V$ and all $t \geq 0$.*

***Proof*** Let $i, j \in V$ be vertices with $i \sim j$ and $\phi \in \mathrm{Aut}(Z)$ with $j = i^\phi$. For $t = 0$ the assertion follows from $u_i(0) = \vartheta(z_{ii}) = \vartheta(z_{jj}) = u_j(0)$. Now assume that $u_i(t) = u_j(t)$ holds for some $t > 0$. Since $u_i(t) = u_j(t)$, we have $x_i(t) = x_j(t)$. Furthermore, any automorphism preserves adjacency relations. Hence,

$$w_{ik} = \vartheta(z_{ik}) = \vartheta(z_{jk^\phi}) = w_{jk^\phi}.$$

Then by induction, we have

$$\begin{aligned} u_i(t+1) &= (1-d)u_i(t) + \sum_{k \neq i} w_{ik} x_k(t) + h_i \\ &= (1-d)u_j(t) + \sum_{k^\phi \neq j} w_{jk^\phi} x_{k^\phi}(t) + h_j \\ &= u_j(t+1). \end{aligned}$$

■

A Hopfield network $\mathfrak{H}_Z$ satisfying properties (1)-(3) of Theorem 5.2 is a refinement procedure that approximates the automorphism partition of $Z$. At each time step $t$, the internal state $\boldsymbol{u}_t$ induces a partition $\mathcal{P}_t(X)$ of $V(Z)$. Two vertices $i$ and $j$ are members of the same cell from $\mathcal{P}_t(X)$ if, and only if, $u_i(t) = u_j(t)$. The initial activation induces an initial partition $\mathcal{P}_0(X)$, which is iteratively refined according to update rule (4.1). Refinement terminates when the current partition $\mathcal{P}_t(X)$ is not finer than the previous partition $\mathcal{P}_{t-1}(G)$. Algorithm 5 summarizes the refinement procedure.

In the simplest form, we can derive a partition $\mathcal{P}_t(X)$ from $\boldsymbol{u}_t$ by collecting vertices $i$ in the same cell if, and only if, their corresponding units have identical activation. To obtain finer partitions, we may proceed as follows: Let $\boldsymbol{u}_t$ be the current activation of $\mathfrak{H}_Z$, and let $\mathcal{P}_{t-1}(X) = \{C_1, \ldots, C_p\}$ be the partition at time $t-1$. We partition each cell $C_k \in \mathcal{P}_{t-1}(X)$ into sub-cells $C_{k1}, \ldots, C_{kq_k}$ such that $i, j \in C_{kl}$ if, and only if, $u_i(t) = u_j(t)$ for all $l \in [1 : q_k]$. The partition $\mathcal{P}_t(X)$ at

**Algorithm 5** (NVR — Neural Vertex Refinement Algorithm )

**Input:**
$Z$ – normalized graph of order $n$

**Initialization:**
construct $\mathfrak{H}_Z$ satisfying (1)-(3) of Theorem 5.2
set $t = 0$

**Procedure:**
**repeat**
increment $t$
update $\mathfrak{H}_Z$ according to the rule (4.1)
derive partition $\mathcal{P}_t(X)$ from $\boldsymbol{u}_t$
**until** $\mathcal{P}_t(X) \not\prec \mathcal{P}_{t-1}(X)$

**Output:** $\mathcal{P}_{t-1}(X)$

time $t$ is then of the form

$$\mathcal{P}_t(X) = \{C_{11}, \ldots, C_{1q_1}, \ldots, C_{p1}, \ldots, C_{pq_p}\}.$$

In this way, refinement terminates when $\mathcal{P}_t(X) = \mathcal{P}_{t-1}(X)$.

We can express the difference between both refinement procedures in terms of vertex invariants. At time $t$, simple refinement uses $f_t$ as a vertex invariant, whereas the alternative form applies the $t$-dimensional invariant $\boldsymbol{f}_t = (f_0, \ldots, f_t)$ to derive the partition $\mathcal{P}_t(X)$.

It remains to show that the neural refinement procedure of $\mathfrak{H}_Z$ terminates.

***Theorem 5.3***
*Let $\mathfrak{H}_Z$ be a Hopfield network associated with a graph $Z$. Then Algorithm 5 terminates within finite time.*

***Proof*** Since $Z$ is finite, there is only a finite number of distinct partitions of $V(Z)$. In addition, the relation $\prec$ defines a partial order on the set of vertex partitions. Hence, there exists a $t_f$ such that $\mathcal{P}_{t_f}(Z)$ is not finer than $\mathcal{P}_{t_f-1}(Z)$. ■

The next example illustrates how a neural refinement procedure operates.

***Example 5.2***
Consider the graph $Z$ given in Figure 5.3. The graph consists of $|Z| = 6$ vertices. Thus, the cardinality of the search space $\mathcal{S}_6$ is $6! = 720$. We reduce the search space using the refinement procedure

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t + \boldsymbol{W}\boldsymbol{x}_t, \tag{5.8}$$

where $\boldsymbol{W} = (w_{ij})$ is of the form

$$w_{ij} = \begin{cases} 1 & : \text{ if } (i,j) \in E(Z) \\ 0 & : \text{ otherwise} \end{cases}$$

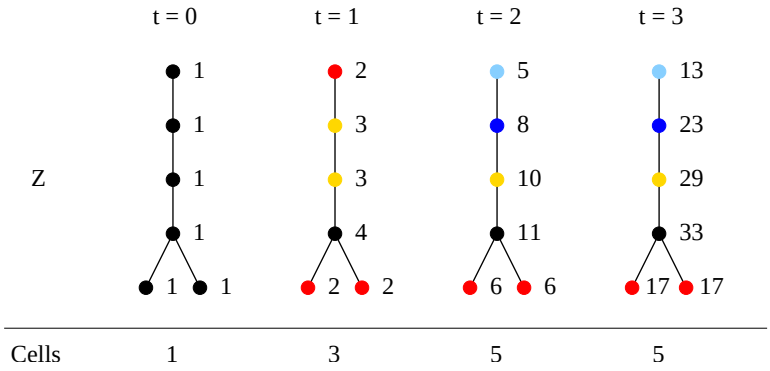



**Figure 5.3** Neural refinement procedure. The quantities attached to the vertices show the output state of the corresponding units. Vertices with the same state belong to the same cell. To highlight their membership to a cell, vertices from the same cell are drawn with the same color. Initially, the activation of each unit is set to 1. Thus, at $t = 0$, the partition $P_0(Z)$ consists of the single cell $V(Z)$. States are updated according to rule (5.8). The refinement procedure terminates at time step $t = 3$ when the number of cells does not increase.

for all $i, j \in V(Z)$. The output is related to the activation via the identical transfer function, i.e. $\boldsymbol{x}_t = \boldsymbol{u}_t$. According to (5.8), the state of a unit $i$ is updated by adding its own state and the states of all its neighbors. This procedure is a slight modification of Morgan's procedure [269].

The network $\mathfrak{H}_Z$ is initialized by $x_i(0) = 1$ for all $i \in V(Z)$. Hence, at $t = 0$ the partition is of the form $\mathcal{P}_0(Z) = \{V(Z)\}$. At the next time step $t = 1$, the state of $\mathfrak{H}_Z$ is $x_i(1) = \deg(i) + 1$, determining the same partition as the degree invariant. We obtain the partition $\mathcal{P}_1(Z) = \{C_1, C_2, C_3\}$, where $C_d$ collects all vertices of degree $d$ (and state $d+1$) for $d \in [1:3]$. Since we have *three* vertices of degree 1, *two* vertices of degree 2, and *one* vertex of degree 3, the search space is reduced to

$$\prod_{i=1}^{3} |C_i|! = 3! \cdot 2! \cdot 1! = 12$$

elements.

Further application of (5.8) until the number of cells does not increase results in a partition $\mathcal{P}_2(Z) = \mathcal{P}_3(Z)$ consisting of five cells. As shown in Figure 5.3, $\mathcal{P}_3(Z)$ has four cells containing *one* vertex and one cell containing *two* vertices. Hence, after termination the search space is reduced to 2 elements.

### *Inexact Graph Isomorphism*

Again, we assume that $\mathfrak{H}_Z$ is a Hopfield network associated with a normalized graph $Z$. As in the exact case, we are interested in the conditions under which the internal states of $\mathfrak{H}_Z$ are vertex $\tau$-invariants.

***Theorem 5.4***
*Let $Z = (V, E, \boldsymbol{Z})$ be a normalized graph with adjacency matrix $\boldsymbol{Z} = (z_{ij})$, let $\mathfrak{H}_Z$ be a Hopfield model associated with $Z$, and let $g : \mathbb{R} \to \mathbb{U}$ denote the transfer function of all units of $\mathfrak{H}_Z$. Assume that $\mathfrak{H}_Z$ satisfies the following properties for all $i, j \in V$:*

1. $d = 1$
2. $h_i = h_j$
3. *The weights $w_{ij}$ of $\mathfrak{H}_Z$ are of the form*

$$w_{ij} = \begin{cases} \dfrac{z_{ij}}{3(n-1)} & : \quad \text{if } (i,j) \in E \\ 0 & : \quad \text{if } (i,j) \notin \overline{E} \end{cases}$$

4. *The function $g$ is Lipschitz continuous with Lipschitz constant $L \leq 1$*
5. $u_i(0) = z_{ii}$

*Then*

$$i \sim_\tau j \quad \Rightarrow \quad |u_i(t) - u_j(t)| \leq \tau$$

*for all $i, j \in V(G)$ and all $t \geq 0$.*

***Proof*** Without loss of generality, we assume that the external input is zero, i.e. $\boldsymbol{h} = \boldsymbol{0}_n$. Let $i, j \in V$ be two vertices with $i \sim_\tau j$ and $\phi \in \mathrm{Aut}(Z)$ with $i^\phi = j$. Clearly, the assertion holds for $t = 0$, since

$$|u_i(0) - u_j(0)| = |z_{ii} - z_{jj}| \leq \tau$$

by property (5). Now assume that $|u_i(t) - u_j(t)| \leq \tau$ for some $t \geq 0$. By induction we have

$$\begin{aligned} |u_i(t+1) - u_j(t+1)| &= \left| \sum_{k \neq i} w_{ik} x_k(t) - \sum_{k^\phi \neq j} w_{jk^\phi} x_{k^\phi}(t) \right| \\ &= \frac{1}{3(n-1)} \left| \sum_{k \neq i} z_{ik} x_k(t) - \sum_{k^\phi \neq j} z_{jk^\phi} x_{k^\phi}(t) \right| \\ &= \frac{1}{3(n-1)} \left| \sum_{k \neq i} z_{ik} x_k(t) - \sum_{k \neq i} (z_{ik} + \tau_{ik})(x_k(t) + \tau_{kk}) \right| \end{aligned}$$

with $\tau_{ik} = z_{i^\phi k^\phi} - z_{ik}$ and $\tau_{kk} = x_{k^\phi}(t) - x_k(t)$. We have $|\tau_{ik}| = |z_{i^\phi k^\phi} - z_{ik}| \leq \tau$. Furthermore, from $|u_i(t) - u_j(t)| \leq \tau$ by assumption, together with

$$|x_i(t) - x_j(t)| \leq |u_i(t) - u_j(t)|$$

by property (4), follows $|\tau_{kk}| \leq \tau$. Hence,

$$\begin{aligned}
|u_i(t+1) - u_j(t+1)| &= \frac{1}{3(n-1)} \left| \sum_{k \neq i} \tau_{kk} z_{ik} + \sum_{k \neq i} \tau_{ik} x_k(t) + \sum_{k \neq i} \tau_{ik} \tau_{kk} \right| \\
&\leq \frac{\tau}{3(n-1)} \left( \underbrace{\sum_{k \neq i} z_{ik} + \sum_{k \neq i} |x_k(t)|}_{\leq 2(n-1)} + \tau(n-1) \right) \\
&\leq \frac{\tau}{3(n-1)} \Big( \underbrace{2(n-1) + \tau(n-1)}_{\leq 3(n-1)} \Big) \\
&= \tau.
\end{aligned}$$

The inequality from the second to the third line uses $|x_k(t)| \leq 1$. This proves the assertion. ■

Note that condition (3) of Theorem 5.4 averages out the errors $|z_{ij} - z_{kl}|$ when mapping all edges incident to vertex $i$ to the edges incident to vertex $k$.

A Hopfield network $\mathfrak{H}_Z$ that satisfies properties (1)-(5) of Theorem 5.4 approximates the $\tau$-automorphism cover of $Z$. At time $t$, the internal state $\boldsymbol{u}_t$ induces an invariant cover $\mathcal{C}_t(Z)$ of $V(Z)$. Each cluster $C$ of $C_t(Z)$ collects all vertices $i, j \in V$ with $|u_i(t) - u_j(t)| \leq \tau$. Starting with an initial invariant cover $\mathcal{C}_0(Z)$ induced by $\boldsymbol{u}_0$, the refinement mechanism is essentially the same as in the exact counterpart. Hence, we may adapt Algorithm 5 to the conditions of inexact graph isomorphism. Instead of partitions $\mathcal{P}_t(Z)$ of $V$, we derive invariant covers $\mathcal{C}_t(Z)$ at each time step $t$. The refinement algorithm terminates when the current invariant cover is not finer than the previous one. As in the exact case, we may use either $f_t$ or $\boldsymbol{f}_t = (f_0, \ldots, f_t)$ as vertex $\tau$-invariant at time step $t$. Convergence of the inexact refinement procedure follows from the same argumentation as in the proof of Theorem 5.3.

### 5.6.2 A Neural Isomorphism Test Procedure

Since exact and inexact graph isomorphism problems are $\mathfrak{p}$-graph matching problems, we can apply Theorem 3.1 to transform graph $\tau$-isomorphism to a maximum clique search in an association graph. We exploit information about the partitions and invariant covers from the refinement procedure to reduce the order of the association graph.

Let $X$ and $Y$ be normalized graphs, and let $f$ be a vertex $\tau$-invariant. Here, $f$ is the invariant that induces the partition and invariant cover, respectively, of the refinement procedure. As usual, let $\boldsymbol{X} = (x_{ij})$ and $\boldsymbol{Y} = (y_{ij})$ be the attributed adjacency matrices. The compatibility values for items $\boldsymbol{i}$ of $X$ and $\boldsymbol{j}$ of $Y$ are of the

form

$$\kappa_{\boldsymbol{ij}} = \begin{cases} s_V(x_{\boldsymbol{i}}, y_{\boldsymbol{j}}) & : \quad \boldsymbol{i} \in V(X), \boldsymbol{j} \in V(Y) \\ s_E(x_{\boldsymbol{i}}, y_{\boldsymbol{j}}) & : \quad \boldsymbol{i} \in V(X)^{[2]}, \boldsymbol{j} \in V(Y)^{[2]} \end{cases},$$

where $s_V$ and $s_E$ are functions that measure the similarity of vertex and non-vertex items. Vertex similarity is defined by

$$s_V(x_{ii}, y_{jj}) = \begin{cases} 1 & : \quad \big\|f(X,i) - f(Y,j)\big\| \leq \tau \\ 0 & : \quad \text{otherwise} \end{cases}$$

for all $i \in V(X)$ and $j \in V(Y)$. The similarity measure for non-vertex items is of the form

$$s_E(x_{\boldsymbol{i}}, y_{\boldsymbol{j}}) = \begin{cases} 1 & : \quad \big|x_{\boldsymbol{i}} - y_{\boldsymbol{j}}\big| \leq \tau \\ 0 & : \quad \text{otherwise} \end{cases}$$

for all $\boldsymbol{i} \in V(X)^{[2]}$ and $\boldsymbol{j} \in V(Y)^{[2]}$.

We express the graph $\tau$-isomorphism problem as an optimization problem

$$\begin{aligned} &\text{maximize} && f\big(\phi, X, Y\big) = \sum_{i \in D(\phi)} \kappa_{\boldsymbol{ii}^{\phi}} \\ &\text{subject to} && \phi \in \mathcal{M}_{X,Y}. \end{aligned} \tag{5.9}$$

The graphs $X$ and $Y$ are isomorphic if, and only if, there is a morphism $\phi \in \mathcal{M}_{X,Y}$ with $f\big(\phi, X, Y\big) = n^2$. In this case, $\phi$ consistently maps all $n^2$ items of $X$ to the $n^2$ items of $Y$. Hence, $\phi$ is a $\tau$-isomorphism.

Problem (5.9) is a $\mathfrak{p}$-graph matching problem and can therefore be transformed to a maximum clique problem in an association graph $X \diamond Y$. The vertex and edge set of an association graph $X \diamond Y$ are of the form

$$V(X \diamond Y) = \Big\{(i,j) \in V(X) \times V(Y) \ : \ \big\|f(X,i) - f(Y,j)\big\| \leq \tau\Big\}$$

$$E(X \diamond Y) = \Big\{(ik, jl) \in V(X \diamond Y)^{[2]} \ : \ \big|x_{ij} - y_{kl}\big| \leq \tau, \ i \neq j, k \neq l\Big\}.$$

Thus, testing $X$ and $Y$ for isomorphism is equivalent to finding a maximum clique with $n$ vertices in $X \diamond Y$.

Using vertex $\tau$-invariants may reduce the order of $X \diamond Y$ such that undesirable maximal cliques disappear, whereas all maximum cliques are preserved. Note that construction of the association graph already reduces the search space. Vertices for which $s_V(x_{ii}, y_{jj}) = 0$ cannot be mapped onto each other by a $\tau$-isomorphism and are therefore excluded.

## 5.7 Experimental Results

We evaluated the performance of NGI on random graphs, chemical molecules, and random weighted graphs.

### 5.7.1 Random Binary Graphs

In our first test series, we assessed the performance of NGI applied to the graph isomorphism problem of random graphs. Random graphs are one standard test set for graph isomorphism problems in the structural pattern recognition and neural network community [95, 280, 290, 291, 332].

#### *Setting of NGI*

We used $f_t$ from (5.7) as vertex invariant. In addition, for sparse graphs we used the number $c_i$ of connected components $Z_i$ of order $i \in \{1, 2, 3\}$ as graph invariant. A component $Z_i$ of order $i$ is a path of length $i-1$ if $i \in \{1, 2, 3\}$. Since isomorphic graphs have the same number of components, it is sufficient to compare the numbers $n_i$. If the graphs under consideration pass that test, the components $Z_i$ can be discarded and isomorphism testing continues on the reduced graphs consisting of the remaining components.

#### *Selected Algorithms*

We compared the performance of NGI with the performance of the following graph isomorphism algorithms:

| Acronym | Method | Implementation | Reference |
|---|---|---|---|
| REP | Replicator Equations | Java | [280] |
| ULL | Ullman's algorithm | C++ | [362] |
| VF | VF algorithm | C++ | [63] |

For random graphs with edge probability $p \geq 0.1$, the REP approach proposed by Pelillo [280] is the most powerful method that has been reported in neural network literature. For those graphs, REP outperformed the Lagrangian relaxation network [291], the optimizing network architecture [290], and Simic's constrained networks approach [332]. The algorithm is described in Appendix A.2.1. We used the original parameters presented in [280], which are $\tau = 10^{-17}$, $\kappa = 10$, and $\boldsymbol{x}_0 = \boldsymbol{e}_n/n$. In his approach, Pelillo transformed the GIP to a clique search problem in a reduced association graph using the vertex degree invariant.

ULL and VF are both exact algorithms that guarantee to return a correct solution. We selected ULL for comparison with NGI, because it is still one of the most commonly used algorithms for graph and subgraph isomorphism problems.

VF was chosen, as it is one of the fastest graph isomorphism algorithm available. On the problems considered here, the performance of VF is, in average, roughly equivalent to the performance of *Nauty* proposed by McKay [255].[4]

***Evaluation Procedure***

We conducted three test series. In all experiments, we considered pairs of graphs $(X, Y)$, where $X$ is an $n$-vertex random graph with edge probability $p$ and $Y$ is a randomly permuted copy of $X$. Since $X$ and $Y$ are isomorphic if, and only if, the complements $\overline{X}$ and $\overline{Y}$ are isomorphic, it is sufficient to consider edge probabilities $p \leq 0.5$.

1. *Experiment 1: Comparison of NGI and REP*
In our first experiment, we compared the performance of NGI with the performance of REP. The chosen parameters were

$$n \in \{100\}$$
$$p \in \{0.01, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}.$$

For each $n$ and each $p$, we generated 100 pairs of random graphs.

2. *Experiment 2: Comparison of NGI, ULL, and* VF
The second test series compared NGI against ULL and VF. We chose the following parameters[5]:

$$n \in [100 : 100 : 1000]$$
$$p \in [0.05, 0.5].$$

For each $n$, we generated 100 pairs of random graphs, where $p$ was sampled from a uniform distribution over $[0.05, 0.5]$. Since ULL requires connected input graphs, the parameter $p$ is bounded below by 0.05.

3. *Experiment 3: Comparison of NGI and* VF *for Graphs or Order* $\geq 1000$
In our last experiment, we assessed the performance of NGI for random graphs of order $n \geq 1000$. The chosen parameters were

$$n \in [1000 : 500 : 5000]$$
$$p \in ]0.0, 0.5].$$

For each $n$, we generated $1,000$ pairs of random graphs, where $p$ was sampled as in the second experiment.

---

4. In the literature, Nauty is considered to be the fastest isomorphism algorithm today. We preferred VF to Nauty, because it is much easier to handle and simpler to integrate in our test environment.
5. Recall that $[n : s : m]$ denotes the set of all elements from $n$ to $m$ in steps of $s$.

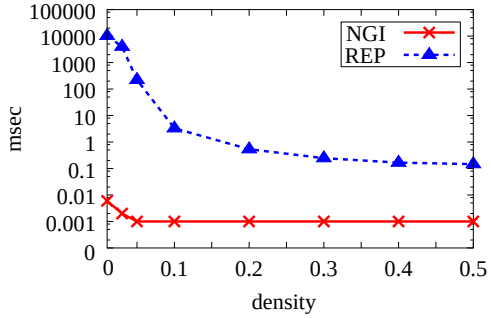



**Figure 5.4** Average clock times of REP and NGI for isomorphism tests on 100-vertex random graphs with varying edge probability. Time was measured in *msec*.

| | REP | | NGI | | |
|---|---|---|---|---|---|
| $p$ | avg | std | avg | std | fac |
| 0.01 | *10,253.000* | 5,995.90 | 0.006 | 0.01 | 1,525,700.0 |
| 0.03 | 3,994.900 | 1,802.60 | 0.002 | — | 1,637,300.0 |
| 0.05 | 222.510 | 251.40 | 0.001 | — | 176,600.0 |
| 0.10 | 3.291 | 1.13 | 0.001 | — | 2,697.4 |
| 0.20 | 0.534 | 0.12 | 0.001 | — | 392.4 |
| 0.30 | 0.245 | 0.05 | 0.001 | — | 191.5 |
| 0.40 | 0.167 | 0.03 | 0.001 | — | 133.3 |
| 0.50 | 0.146 | 0.03 | 0.001 | — | 104.4 |

**Table 5.1** Average clock times (*sec*) and standard deviation of REP and NGI for isomorphism tests on 100-vertex random graphs with varying edge probability. The last column with identifier `fac` shows the factor by which NGI is faster than REP. For $p = 0.01$, we forced REP to terminate prematurely after $10,000$ iterations in all trials. Note that standard deviation of NGI for $p \geq 0.03$ is less than 0.01.

For direct comparison of execution times, we implemented NGI and REP in Java using `JDK` 1.2. For ULL and VF we used the C++ implementations by [63, 95].[6] All experiments were executed on a multi-server `Sparc SUNW Ultra-4`.

### *Numerical Results*

In all three experiments, NGI always returned a correct solution.

*Experiment 1: Comparison of NGI and REP*

6. The implementations are available at `http://amalfi.dis.unina.it/graph/` (03/31/2004).

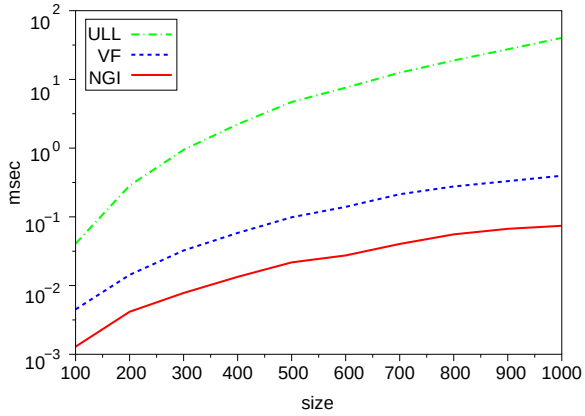



**Figure 5.5** Average clock times of ULL, VF, and NGI for isomorphism tests on random graphs of varying size. Time was measured in *msec*. Note that ULL and VF have been implemented in C++, and NGI in Java.

For $p = 0.01$, we prematurely terminated REP after $10,000$ iterations. For $p \geq 0.03$, REP always returned a correct solution. To obtain a general overview, Figure 5.4 depicts the average execution times of NGI and REP. Table 5.1 presents the results in a more detailed form.

From the results, we see that NGI was about 100 up to $1.5 \cdot 10^6$ times faster than REP. The time performance of REP degraded with decreasing edge probability, because the degree invariant becomes useless for very sparse graphs. This observation and the superior results of NGI indicate that structural preprocessing can have considerably more impact on the performance than sophisticated and powerful energy minimization methods like the Lagrangian relaxation network [291] or the optimizing network architecture [290].

*Experiment 2: Comparison of NGI, ULL, and* VF

As in the first experiment, Figure 5.5 serves to provide a general overview of the time performance of NGI, ULL, and *VF*. More detailed results are summarized in Table 5.2.

Although a fair time comparison is not possible because of different implementation languages, we see that the Java implementation of NGI is significantly faster than the C++ implementations of ULL and *VF*. The results

- confirm that previous approaches from the neural network community like REP are not even competitive with standard methods like ULL, and
- show that neural network solutions to the GIP can be made competitive with powerful methods like VF.

*Experiment 3: Evaluation of NGI for Graphs or Order* $\geq 1000$

Table 5.3 summarizes the results of our last test series. Since we obtain a similar

| | ULL | | VF | | NGI | |
|---|---|---|---|---|---|---|
| $n$ | avg | std | avg | std | avg | std |
| 100 | 0.041 | 0.008 | 0.005 | 0.005 | 0.001 | 0.001 |
| 200 | 0.285 | 0.038 | 0.014 | 0.007 | 0.004 | 0.002 |
| 300 | 0.946 | 0.105 | 0.032 | 0.013 | 0.008 | 0.002 |
| 400 | 2.223 | 0.140 | 0.058 | 0.024 | 0.013 | 0.004 |
| 500 | 4.696 | 0.330 | 0.099 | 0.044 | 0.022 | 0.013 |
| 600 | 7.587 | 0.375 | 0.140 | 0.067 | 0.027 | 0.003 |
| 700 | 12.536 | 0.534 | 0.213 | 0.097 | 0.040 | 0.008 |
| 800 | 18.886 | 0.928 | 0.277 | 0.154 | 0.056 | 0.043 |
| 900 | 27.397 | 1.339 | 0.331 | 0.155 | 0.067 | 0.033 |
| 1000 | 40.201 | 1.849 | 0.396 | 0.180 | 0.074 | 0.013 |

**Table 5.2** Average clock times (*sec*) and standard deviation of ULL, VF, and NGI for isomorphism tests on random graphs with varying size. Note that ULL and VF have been implemented in C++, and NGI in Java.

picture as in the second experiment, we did not include a plot for a general overview. The results indicate that NGI scales better than VF with increasing problem size.

### 5.7.2 Chemical Compounds

The aim of this experiment is to study the effects of neural refinement procedures when the graphs to compare have a high degree of symmetry. For this, we applied NGI to the problem of identifying chemical compounds from the MUTAGENESIS dataset.[7] The compounds were transformed to chemical graphs in the same way as in Section 4.6.4. Again, we only used element symbols as attributes for vertices and bond types as attributes for edges.

#### *Setting of NGI*

The refinement procedure of NGI used $\boldsymbol{f}_t = (f_0, \ldots, f_t)$ as vertex invariant (see p. 131). In addition, non-edges $(i,j) \in \overline{E}(X)$ and $(k,l) \in \overline{E}(Y)$ are considered to be compatible if $d_{ij} = d_{kl}$, where $d_{rs}$ denotes the distance between vertices $r$ and $s$. Note that the distance is an edge invariant, which can also be obtained by evolving a neural network.

7. Appendix B.2.2 describes the MUTAGENESIS dataset.

| | VF | | NGI | | |
|---|---|---|---|---|---|
| $n$ | avg | std | avg | std | fac |
| 1000 | 0.46 | 0.25 | 0.07 | 0.01 | 6.4 |
| 1500 | 1.22 | 0.72 | 0.16 | 0.06 | 7.5 |
| 2000 | 1.99 | 1.02 | 0.28 | 0.05 | 7.0 |
| 2500 | 3.65 | 2.44 | 0.43 | 0.05 | 8.5 |
| 3000 | 5.37 | 3.35 | 0.64 | 0.09 | 8.4 |
| 3500 | 8.92 | 4.99 | 0.87 | 0.09 | 10.3 |
| 4000 | 10.76 | 6.53 | 1.13 | 0.11 | 9.5 |
| 4500 | 16.33 | 10.47 | 1.48 | 0.17 | 11.0 |
| 5000 | 18.76 | 12.33 | 1.83 | 0.19 | 10.2 |

**Table 5.3** Average clock times (*sec*) and standard deviation of VF and NGI for isomorphism tests on random graphs of varying order. The last column with identifier `fac` shows the factor by which NGI is faster than VF. Note that a fair comparison is not possible, because VF has been implemented in C++, and NGI in Java.

### *Selected Algorithms*

We compared the performance of NGI with the performance of the following algorithms:

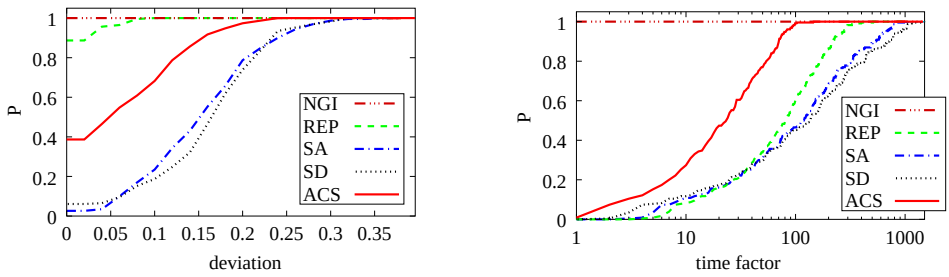
| Acronym | Method | Reference |
|---|---|---|
| REP | Replicator Equations | [280] |
| SA | Simulated Annealing | [146] |
| SD | Steepest Descent | [167] |
| ACS | Attention Control System | Sec. 4.5.2 |

The algorithms were selected for the same reasons as in Section 4.6.4. A pseudo-code for the algorithms is presented in Appendix A. We excluded Tabu Search, because the correct solution of each isomorphism test was known in advance.

### *Evaluation Procedure*

For each chemical graph $X$, we generated a randomly permuted copy $Y$. The selected algorithms computed the cardinality of the maximum common induced subgraph $X$ and $Y$ via maximum clique search in an association graph $Z$. An algorithm solved the isomorphism problem when the size of the clique found was equal to the order of $X$ and $Y$.

The algorithms were implemented in Java using `JDK` 1.2. Experiments were executed on a multi-server `Sparc SUNW Ultra-4`.




(a) Profiles for size of clique

(b) Profiles for clock time

**Figure 5.6** Results of 230 isomorphism tests of chemical graphs from the MUTAGENESIS dataset. Shown are the performance profiles for (a) clique size and (b) clock time.

#### *Numerical Results*

Figure 5.6 presents the performance profiles of the selected algorithms for clique size and clock time. From the plots it is evident that NGI clearly outperformed the other algorithms in regard to solution quality and time. As shown in Figure 5.6(a) and Table 5.4, NGI found a correct solution in all 230 trials and is an average of 30 times faster than ACS, and an average of more than 100 times faster than the other algorithms. From Figure 5.6(b) we see that ACS is a good choice as a subroutine in NGI, because ACS is significantly faster than REP, SA, and SD. The results underpin the effectiveness of refinement procedures for the GIP.

Regarding solution quality, it is notable that REP, SA, and ACS are no longer roughly equivalent as in the experiments of Section 4.6.4. The plot in Figure 5.6(a) shows that REP is clearly superior to the other simple heuristics SA, SD, and ACS. The performance of SA degraded to the poor performance of SD in such that ACS is ranked second.

Although a fair comparison is not possible, it should be mentioned that `NGI` is about 50 times faster than the algorithm proposed by [2] applied on synthetical protein molecules of about the same size.

### 5.7.3 Random Weighted Graphs

In our last test series, we applied NGI to the $\tau$-isomorphism problem of random weighted graphs. We chose random weighted graphs as test graphs to enable comparison with other algorithms.

#### *Selected Algorithms*

For comparison, we considered the *Lagrangian Relaxation Network* (LRN) proposed by [291] and the *Optimizing Network Architecture* (ONA) proposed by [290].

| | | time (*msec*) | |
|---|---|---|---|
| | error | avg | fac |
| NGI | 0.00 | 3.2 | 1.0 |
| REP | 0.11 | 338.6 | 105.8 |
| SA | 0.97 | 692.6 | 216.4 |
| SD | 0.94 | 896.9 | 280.3 |
| ACS | 0.61 | 102.6 | 32.1 |

**Table 5.4** Shown are the error and average time behavior of the algorithm for the isomorphism problem of chemical graphs from the MUTAGENESIS dataset. The column with identifier *error* shows the error rate, the column with identifier *avg* shows the average computation time measured in *msec*, and the column with identifier *fac* shows the ratio $avg/avg^*$ of the average time performance *avg* of an algorithm and the best average time performance $avg^* = 3.2$.

Both algorithms have performed best on inexact graph isomorphism problems and outperformed the eigendecomposition approach [363], the polynomial transform approach [8], and the linear programming approach [9].

***Evaluation Procedure***

For each $\tau$-isomorphism test we generated pairs of graphs $(X, Y)$ as follows: First, we created a random graph $X$ with $n$ vertices and edge probability $p$. Vertices and edges were colored with weights drawn from a uniform distribution over $]0, 1]$. To construct $Y$, we randomly permuted $X$ and added random noise to the vertex and edge weights of $Y$. Noise was sampled from a uniform distribution over the interval $[-\tau, +\tau]$. The chosen parameters were $n = 100, 500, 1000$, $p = 0.25, 0.5, 0.75, 1.0$, and $\tau = \alpha/n$ with $\alpha = 1.0$, 0.5, 0.25, 0.1. We generated 100 examples for each pair $(n, \alpha)$ giving a total of 6 000 isomorphism tests for NGI. LRN and ONA have been applied by Rangarajan et al. [291, 290] only on complete random weighted graphs of order 100 with varying noise levels.

NGI was implemented in Java using `JDK` 1.2. Experiments were executed on a multi-server `Sparc SUNW Ultra-4` with a `CPU` clock rate of 900.0 MHz and a memory clock rate of 150.0 MHz. As reported in [291, 290], LRN and ONA were both executed on Silicon Graphics Indigo workstations with R4000 and R4400 processors. According to Silicon Graphics Computer Systems, the CPU clock rate of R4000 and R4400 processors are 100 and 150 Mhz, respectively.

***Numerical Results***

NGI returned a correct solution on all 6000 trials. Table 5.5 shows the average computation times. We see that `NGI` performed best on sparse graphs with a low noise level and worse on dense graphs with a high noise level. For various noise

| | | noise factor $\alpha$ | | | | |
|---|---|---|---|---|---|---|
| $n$ | $p$ | 1.0 | 0.75 | 0.5 | 0.25 | 0.1 |
| 100 | | 6 | 5 | 4 | 4 | 3 |
| 500 | 0.25 | 119 | 115 | 112 | 108 | 92 |
| 1000 | | 549 | 537 | 531 | 519 | 448 |
| 100 | | 6 | 5 | 5 | 4 | 4 |
| 500 | 0.50 | 135 | 129 | 126 | 121 | 107 |
| 1000 | | 663 | 642 | 629 | 620 | 562 |
| 100 | | 8 | 6 | 5 | 5 | 4 |
| 500 | 0.75 | 151 | 142 | 134 | 131 | 119 |
| 1000 | | 675 | 716 | 721 | 716 | 675 |
| 100 | | 10 | 7 | 6 | 5 | 4 |
| 500 | 1.00 | 173 | 156 | 146 | 140 | 129 |
| 1000 | | 879 | 841 | 818 | 800 | 774 |

**Table 5.5** Results of $\tau$-isomorphism tests on random weighted graphs. Shown are the average clock times in *msec* required by `NGI`.

levels, LRN also returned a correct solution for the graph $\tau$-isomorphism problem on graphs of order 100 in all trials. The same holds for ONA, provided the noise level is sufficiently low.

Since LRN and ONA were implemented in a different programming language and have been executed on slower machines, comparison of execution times with NGI is intricate. Nevertheless, if the ratio of CPU clock rates of two machines differs significantly from the ratio of execution times of two algorithms running on the respective machines, we can at least make qualitative statements on the time performance of both algorithms.

The execution time of LRN for matching graphs of order 100 was, on average, about 20-30 minutes on a SGI workstation. ONA required an average of about 80 seconds on the same type of machine. In contrast, the average execution times of NGI to match 100-vertex graphs were in the range of 0.003-0.01 seconds on a `Sparc SUNW Ultra-4` machine. It is notable that NGI for graphs of order 1000 is still about 80 times faster than ONA for graphs of order 100.

Although the algorithms ran on different machines, we may conclude that NGI is significantly faster than LRN and ONA for the following reasons: first, the CPU clock rate of a `Sparc SUNW Ultra-4` is only about 6-9 times faster than the CPU clock rate of an SGI workstation with R4000 and R4400 processors; second, the average execution time of NGI on graphs of order 100 is roughly $10^4$-$10^7$ times faster than the execution times of LRN and ONA on problem instances of the same size.

To summarize, if noise was sufficiently low so that the structure of a graph is not disrupted, NGI outperformed LRN and ONA. For increasing noise level, the graph $\tau$-isomorphism problem gradually turned into a general graph matching problem.

Hence, the refinement procedure of NGI became gradually ineffective, resulting in a degraded performance.

## 5.8 Conclusion

In this chapter, we presented a two-stage neural network solution to the graph $\tau$-isomorphism problem. In the first stage a neural refinement procedure classifies vertices into clusters (cells) using vertex $\tau$-invariants. The second stage performs isomorphism testing through a clique search in a reduced association graph. We justified this approach by studying the permanents of the bipartite association derived from both graphs under consideration.

Experiments indicate that NGI is (i) superior than neural and other energy minimizing methods, and (ii) competitive with the most powerful solutions to the GIP. Two direct implications of these results are that neural refinement is (i) capable of discovering structural properties of vertices within a graph, and (ii) of greater impact than sophisticated energy minimization methods.

The results suggest pursuing research in the following directions: (i) enhance NGI with more sophisticated vertex invariants, (ii) incorporate edge and graph invariants into NGI, and (iii) test the improved NGI on harder problems.

# 6 Neural Anytime Computation

In this chapter, we present real-time neural networks for combinatorial optimization problems that can be interrupted at any point in time and provide meaningful results. In addition, the quality of results improves with increasing computation time. Such systems are essential for successful operation in domains where it is computationally infeasible or economically undesirable to complete deliberation. The proposed approach is based on ideas from anytime computing, which is a well-known approach of real-time systems in Artificial Intelligence.

## 6.1 Introduction

This section is devoted to the construction of real-time systems, a fundamental problem in computer science and artificial intelligence. A real-time system is a system that is subject to constraint in time. In almost all cases, the computation required to return an optimal solution to a complex task will degrade the system's overall utility. A successful system therefore trades off solution quality for computation time [33, 68, 154, 226, 333, 375, 398].

An important approach of real-time systems in Artificial Intelligence are *anytime algorithms* [33]. An anytime algorithm is an algorithm that provides approximate answers to computationally difficult problems so that

1. an answer is available at any time in the execution of the algorithm and
2. the quality of the answer improves with increasing execution time.

Anytime algorithms can be categorized into *interruptible* and *contract* algorithms [308]. Interruptible algorithms return meaningful results even when unexpectedly interrupted. Contract algorithms guarantee to produce a result within an allocated time that must be specified in advance. If interrupted before the allocated deadline, a contract algorithm may yield useless results. In many applications, interruptible algorithms are more appropriate, but also more difficult to construct [400].

Continuing development of anytime algorithms since the end of the 1980s has led to successful applications in diverse areas such as the evaluation of belief networks [155, 382], planning and scheduling [34, 399], database query processing [327], constraint satisfaction [98, 376], and real-time control systems [29].

It is notable that there have been almost no attempts to incorporate anytime

characteristics in the design of neural networks for solving combinatorial optimization problems. In [103], Gallone & Charpillet proposed a contract Mean-Field algorithm for scheduling problems. In their work, contract time was predetermined by varying the initial temperature of a fixed annealing schedule. Research on interruptible neural computation is lacking. This is notable, because connectionist modelers emphasize *robustness* as one attractive feature of neural networks. Robustness in the neural network community, however, is commonly referred to as fault tolerance with respect to distortions in input data or damaged parts of the neural architecture. Time-dependent fault tolerance in the sense that neural networks can be unexpectedly interrupted at any time while still producing meaningful results has not yet been considered in the context of combinatorial optimization. Thus, the central question of this chapter is

*Can neural networks operate robustly in real-time environments?*

The answer is yes, and evidence will be provided in this and the next chapter. After reviewing anytime algorithms in the next section, we present a framework for constructing anytime Hopfield models for combinatorial optimization problems in Section 6.3. Experiments are presented and discussed in Section 6.4. Finally, this chapter concludes with Section 6.5.

## 6.2 Anytime Algorithms

An *anytime algorithm* is an iterative algorithm that can be interrupted and provide an approximate result at any time. Usually the quality of the result improves gradually with increasing computation time up to an optimal solution. This section briefly reviews basic elements of anytime algorithms. For a detailed discussion on anytime algorithms see [398].

The term *anytime algorithm* was coined by Dean & Boddy [71, 72] in the late 1980s in their work on time-dependent planning. During the same period, similar models of anytime computing were developed by Horvitz [152, 153] and Liu et al. [234]. Precursors of algorithms with anytime characteristics have been studied in diverse areas including numerical approximation, heuristic search, dynamic programming, Monte Carlo algorithms, and database query handling [107]. The common spirit of these different approaches is the perception that the time required to compute an optimal solution will typically reduce the overall *utility* of an algorithm. Therefore, construction of systems that trade solution quality for computation time is conducive to their overall utility.

Originally, Dean & Boddy [71, 72] introduced the term *anytime algorithm* for a class of algorithms with the following properties:

1. *Interruptibility.* The algorithm can be terminated at any time and will still return some result.

2. *Well-Behaved Improvement.* The quality of results returned improve in some well-behaved manner as a function of time.

3. *Preemptability.* The algorithm can be suspended and resumed with minimal overhead.

Further properties of anytime algorithms have been found desirable to meet the necessities of diverse resource-bounded applications. Following [400], desired properties of anytime algorithms from the standpoint of meta-level control include:

4. *Measurable Quality.* The quality of an approximate result can be precisely determined. An example of a measurable quality is the distance between the approximate result and the correct result.

5. *Recognizable Quality.* The quality of an approximate result can be determined at run time, i.e. within constant time.

6. *Monotonicity.* The quality of the result is a nondecreasing function of the time. If quality is recognizable, we can achieve monotonicity by returning the best result generated so far, rather than the last generated result.

7. *Consistency.* The quality of the result correlates with computation time and input quality. Usually, quality of an algorithm's results for a given amount of time is scattered given various input instances. Consistency means that variance is sufficiently narrow so that quality prediction can be performed.

8. *Diminishing Returns.* The improvement of solution quality diminishes over computation time.

Note that properties (4)-(8) can be used in any combination to specify property (2). The properties of an anytime algorithm for a specific problem domain are usually summarized in a *performance profile.*

The three main components of an anytime algorithm are an *iterative algorithmic map*, a well-behaved *quality measure*, and a *performance profile.* To describe the three components, consider the following optimization problem

$$\begin{array}{ll} \text{maximize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X}, \end{array} \tag{6.1}$$

where $\mathcal{X}$ denotes the solution space. To illustrate some features of anytime algorithms, we occasionally assume that problem (6.1) refers to a MCP.

### *Anytime Algorithmic Maps*

Initiated at $\boldsymbol{x}_0 \in \mathcal{X}$, an iterative algorithmic map $\mathsf{A}$ generates a sequence $(\boldsymbol{x}_t)$ of states defined by

$$\boldsymbol{x}_{t+1} = \mathsf{A}(\boldsymbol{x}_t).$$

Note that an algorithmic map $\mathsf{A}$ is more generally a *state-to-set mapping*. Although $\mathsf{A}(\boldsymbol{x}_t)$ is a subset of $\mathcal{X}$, algorithm $\mathsf{A}$ generates a sequence in the following manner: given an initial point $\boldsymbol{x}_0$, the algorithm generates sequences through the iteration

$$\boldsymbol{x}_{t+1} \in \mathsf{A}(\boldsymbol{x}_t).$$

We refer to [239] for more details.

Incorporation of anytime characteristics into the algorithmic map $\mathsf{A}$ requires only slight modifications for many standard techniques such as *stochastic iterative sampling* [222] or local search algorithms like *hill-climbing* [307], *tabu search* [114, 115], and *simulated annealing* [1, 207, 366]. Other types of algorithms that can be converted to anytime computation procedures include (but are not limited to) numerical approximation algorithms (e.g. for Taylor series approximation) [73, 74] and dynamic programming [23].

Algorithm 6 outlines a simple anytime version of the algorithmic map $\mathsf{A}$. The subset $\mathcal{F} \subseteq \mathcal{X}$ of feasible solutions depends on the problem at hand. For the MCP, we may, for example, choose $\mathcal{F}$ as the set of cliques. By $\boldsymbol{x}$ we denote the state generated by the algorithmic map $\mathsf{A}$, and $\boldsymbol{y} \in \mathcal{F}$ is the best feasible solution found so far. For a given quality measure $Q$, the *improvement condition* is a function

$$IC : \mathcal{X} \times \mathcal{F} \to \{\text{true}, \text{false}\}, \quad (\boldsymbol{x}, \boldsymbol{y}) \mapsto \boldsymbol{x} \in \mathcal{F} \wedge Q(\boldsymbol{x}) > Q(\boldsymbol{y})$$

that returns true if, and only if, $\boldsymbol{x}$ is feasible and of better quality than $\boldsymbol{y}$. Note that in order to determine $IC(\boldsymbol{x}, \boldsymbol{y})$, it is more convenient in some cases to evaluate a simpler expression equivalent to $\boldsymbol{x} \in \mathcal{F} \wedge Q(\boldsymbol{x}) > Q(\boldsymbol{y})$.

**Algorithm 6** (Anytime A)

**Input:**
- $IC$ – improvement criterion
- $\mathcal{X}$ – solution space
- $\mathcal{F}$ – subset of feasible solutions

**Initialization:**

**if** $\boldsymbol{x} \notin \mathcal{F}$ **then** choose $\boldsymbol{x} \in \mathcal{F}$

$\boldsymbol{y} =$ register$(\boldsymbol{x})$

**Procedure:**

**repeat**

  $\boldsymbol{x} = \mathsf{A}(\boldsymbol{x})$

  **if** $IC(\boldsymbol{x}, \boldsymbol{y})$ **then** $\boldsymbol{y} =$ register$(\boldsymbol{x})$

  **if** signal **then** break

**until** convergence

**Output:** $\boldsymbol{y}$

The anytime $\mathsf{A}$ algorithm starts by quickly generating a feasible solution from scratch and registering that solution. Registration makes a result available when the algorithm terminates or is interrupted. Next, the algorithm repeatedly tries to

perform an improvement step by generating a new state $\boldsymbol{x}_{t+1} = \mathsf{A}(\boldsymbol{x}_t)$ and then checking the improvement condition $IC(\boldsymbol{x}_{t+1}, \boldsymbol{y})$. If $\boldsymbol{x}_{t+1}$ satisfies the improvement condition with respect to $\boldsymbol{y}$, the new state $\boldsymbol{x}_{t+1}$ is registered and replaces the existing solution $\boldsymbol{y}$. The algorithm terminates after convergence or once it receives an interruption signal. An anytime algorithm handles interrupts by scanning interruption signals at regular intervals. In the particular pseudo-code of Algorithm 6, interruption signals are scanned cyclically after each iteration.

### *Quality Measures*

Quality measures are used to monitor the progress of an anytime algorithm in problem solving, to characterize its performance, and to allocate computational resources effectively. Here, a quality measure for an anytime algorithm $\mathsf{A}$ is a function of the form

$$Q : \mathcal{F} \times \mathbb{R}_+ \to \mathbb{R}, \quad (\boldsymbol{x}, t) \mapsto Q(\boldsymbol{x}, t).$$

The value $Q_t = Q(\boldsymbol{x}_t, t)$ is called *intermediate result* of $\mathsf{A}$ at time $t$. An intermediate result provides a preliminary estimate of the unknown optimal solution given the allocated time. A meaningful quality measure should measure some aspect of the algorithm's output that improves over time, at least on the average.

The overall utility of an algorithm for a given problem domain hinges on an appropriate design of the quality measure. Since different types of algorithms have completely different ways to approach an approximate or optimal solution, quality measures must be designed to meet the characteristics of the problem domain and the algorithms they evaluate.

### *Performance Profiles*

Performance profiles enable an efficient meta-level control of anytime algorithms. A performance profile characterizes the expected solution quality as a function of computation time. A compact representation of performance information is the *expected performance profile* (EPP) [33, 152]. Given a quality measure $Q$, the EPP of an algorithm $\mathsf{A}$ is a function

$$E : \mathbb{R}_+ \to \mathbb{R}, \quad t \mapsto E[Q_t],$$

which maps computation time to the expected quality of the result. EPPs are useful when the variance of the quality distribution is small at any point in time. For large variances of the quality distribution, a *performance interval profile* (PIP) [398] is more suitable than EPPs. A PIP of an algorithm $\mathsf{A}$ with quality measure $Q$ is a function

$$I : \mathbb{R}_+ \to \mathbb{R} \times \mathbb{R}, \quad t \mapsto [L_t, U_t],$$

where

$$L_t \leq Q(\boldsymbol{x}, t) \leq U_t$$

for all $\boldsymbol{x} \in \mathcal{F}$ and $t \in \mathbb{R}_+$.

To generate a performance profile, we sample a representative set of input instances and record the solution quality of A at certain points in time for each input instance. The set of solution qualities over all sampled input instances and points in time is the *quality map* of A. We use the quality map to construct a performance profile of the anytime algorithm. An example of a fictitious performance profile is presented in Figure 6.1.

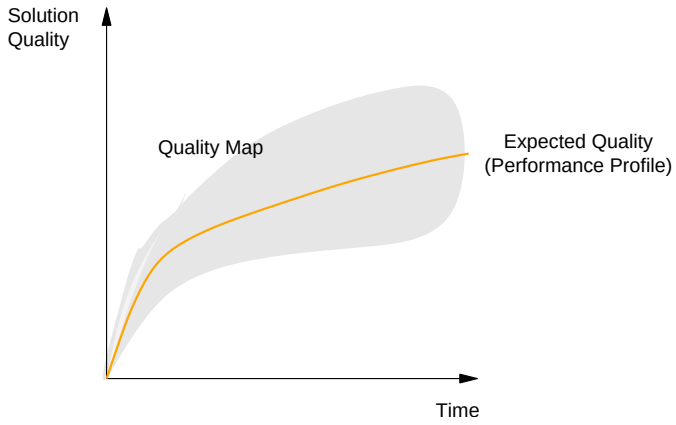



**Figure 6.1** Example of a performance profile. The grey shaded cloud represents the quality map. The performance profile as an estimation of the expected quality is depicted by a solid orange line.

### 6.2.1 An Example

We conclude this introductory review on anytime algorithms with an example that summarizes the distinguishing features of anytime computation. Although this example is simplistic and presents an idealized picture, it is still useful to highlight the characteristic benefits of anytime algorithms.

Consider Figure 6.2 that illustrates utility maximization performed by anytime computation. Here, utility is defined as a function that balances solution quality and computation costs. In this particular example, the utility function is of the simple form

$$\texttt{utility} = \texttt{solution quality} - \texttt{time cost}.$$

The plot depicts the solution quality as a function of time for four types of

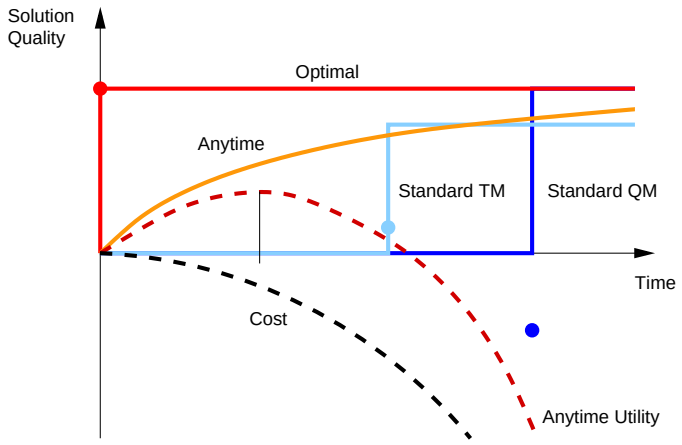



**Figure 6.2** Solution quality and utility of *optimal*, *exact*, *approximate*, and *anytime algorithms*. Solid lines show quality functions and filled circles indicate the utility of an algorithm after termination. The color *red* refers to optimal algorithms, *blue* to exact algorithms that maximize quality (Standard QM), *light blue* to approximate algorithms that minimize time (Standard TM), and *orange* to anytime algorithms. Dashed lines show the cost function (*black*) and the utility function of anytime algorithms (*dark red*).

algorithms and their utility after termination. In addition, a cost function describes the expected loss of utility as a function of time in the absence of a solution.

An *optimal algorithm* returns an optimal solution in no time. The quality function of an optimal algorithm is a step function that rises to maximal quality at $t = 0$. Maximal utility of an optimal algorithm is achieved at $t = 0$ and coincides with maximal quality.

*Standard algorithms* either maximize solution quality (*Standard QM*) or minimize computation time (*Standard TM*). Standard QM algorithms are exact algorithms that guarantee to return an optimal solution after a certain time. Standard TM algorithms are referred to as approximate algorithms that produce an acceptable solution as fast as possible. Certainly, Hopfield models for solving combinatorial optimization problems belong to the class of standard TM algorithms that minimize computation time.

The quality function of a Standard QM algorithm is a step function with minimum quality during computation, which rises to maximal quality after termination. Unfortunately, by that time the solution returned by a Standard QM algorithm is of little use. The algorithm returns a result at a point in time when the cost is very high. Therefore, the utility of a Standard QM algorithm after termination is negative. This shortcoming of Standard QM algorithms motivated researchers to develop approximate algorithms, i.e. Standard TM algorithms. As indicated by the plot, Standard TM algorithms may provide satisfactory results, but still they have unsatisfactory small overall utility.

As opposed to standard algorithms, an anytime algorithm generates suboptimal results at any time when solution quality increases with time. Combining quality and cost, a time allocation can be determined that maximizes the utility of an anytime algorithm.

## 6.3 Anytime Hopfield Models

The aim of this section is to present an anytime Hopfield model for solving combinatorial optimization problems.

Perhaps one reason why anytime computing has not been considered in neural network design for combinatorial optimization problems is that neural network methods spend most of their overall computation time approaching a feasible solution by evolving through an infeasible region. This compels us to slightly modify the traditional definition of a quality measure and to limit the use of anytime computing as outlined in Algorithm 6. Then everything comes down to devising suitable quality measures that map the states of an anytime Hopfield model to meaningful intermediary results of the unknown optimal solution.

Let $\mathfrak{H}$ be a Hopfield model for solving problem (6.1). Our first modification relaxes the constraint on the domain of a quality measure. A quality measure for $\mathfrak{H}$ is a function of the form

$$Q : \mathcal{X} \times \mathbb{R}_+ \to \mathbb{R}, \quad (\boldsymbol{x}, t) \mapsto Q(\boldsymbol{x}, t).$$

In contrast to the standard definition of a quality measure, a quality measure for $\mathfrak{H}$ admits both feasible and infeasible states as input. An example of a quality measure for $\mathfrak{H}$ is the Euclidean distance of $\boldsymbol{x} \in \mathcal{X}$ and the closest optimal solution $\boldsymbol{x}^* \in \mathcal{F}$. Note that a quality reflecting the distance of an approximate solution from $\mathcal{X}$ and an optimal solution from $\mathcal{F}$ is independent of time and measurable, but not recognizable.

The second modification restricts the use of anytime Hopfield models to problems in which we are interested in the constrained objective value $f(\boldsymbol{x}^*)$ of an optimal solution $\boldsymbol{x}^*$, rather than in the solution $\boldsymbol{x}^*$ itself. This restriction is a consequence of the convergence characteristics of Hopfield models. Algorithm 7 presents a basic form of an anytime $\mathfrak{H}$ algorithm. For convenience, we write $Q_t$ instead of $Q(\boldsymbol{x}_t, t)$.

In the remainder of this section, we propose a family of quality measures for the MCP that meets the specific needs of the convergence behavior of Hopfield models.

### *Quality Measures*

Since Hopfield models exhibit a black box behavior, it is difficult to construct non-trivial quality measures that at least approach the optimal quality from below with increasing computation time. What we can do is formulate heuristic quality

**Algorithm 7** (Anytime $\mathfrak{H}$)

**Input:**
$Q$ – Hopfield quality measure
$\mathcal{X}$ – solution space

**Initialization:**
set $t = 0$
set $\boldsymbol{x}_t \in \mathcal{X}$
$Q^* = \mathsf{register}(Q_t)$

**Procedure:**
**repeat**
$\boldsymbol{x}_{t+1} = \mathfrak{H}(\boldsymbol{x}_t)$
**if** $Q_{t+1} > Q^*$ **then** $Q^* = \mathsf{register}(Q_{t+1})$
$t = t + 1$
**if** signal **then** break
**until** convergence

**Output:** $Q^*$

measures that approximate a desired property of a quality measure.

To propose a family of quality measures for Hopfield models, we assume that problem (6.1) refers to a MCP of some given graph $Z$. Let $\mathfrak{H}_Z$ be a Hopfield model associated with $Z$. For a given state $\boldsymbol{x}$ of $\mathfrak{H}_Z$, we define the sets $V_a(\boldsymbol{x})$ of *active vertices* and $\overline{E}_a(\boldsymbol{x})$ of *active non-edges* by

$$V_a(\boldsymbol{x}) = \big\{i \in V(Z) \,:\, x_i > 0\big\}$$
$$\overline{E}_a(\boldsymbol{x}) = \big\{(i,j) \in \overline{E}(Z) \,:\, i,j \in V_a(\boldsymbol{x})\big\}.$$

Note that we identify vertices, edges, and non-edges of $Z$ with their corresponding units, excitatory, and inhibitory connections of $\mathfrak{H}_Z$. A family of quality measures for $\mathfrak{H}_Z$ with problem dependent control parameter $\gamma_t$ is of the form

$$Q(\boldsymbol{x}, t) = \max\Big(0, \big|V_a(\boldsymbol{x})\big| - \gamma_t \big|\overline{E}_a(\boldsymbol{x})\big|\Big). \tag{6.2}$$

The parameter $\gamma_t$ controls to which extent we penalize active non-edges. A proper choice of $\gamma_t$ is crucial for maximizing the utility of $\mathfrak{H}$. Too small values for $\gamma_t$ may result in intermediate results $Q_t$ that exceed the quality $Q(\boldsymbol{x}^*, t)$ of an optimal solution $\boldsymbol{x}^*$. If we choose $\gamma_t$ too large, the performance profile for anytime $\mathfrak{H}$ may degenerate to a step function as for standard $\mathfrak{H}$.

## 6.4 Experiments

The experiments serve to illustrate some features of anytime $\mathfrak{H}$ procedures and to indicate that neural networks for combinatorial optimization problems are potential

capable of performing robustly in real-time environments. Results are presented with an emphasis on conceptual issues rather than experimental exhaustion.

In all experiments, we plugged the Attention Control System (ACS) presented in Section 4.5.2 as our chosen Hopfield model $\mathfrak{H}$ into Algorithm 7.

### 6.4.1 Anytime ACS for the Maximum Clique Problem

In our first experiment, we applied the anytime ACS algorithm to the MCP of 100 random graphs $Z$ of order $n = 2500$ and edge probability $p = 0.75$. Construction of random graphs is presented in Appendix B. We applied (6.2) as a quality measure with control parameter

$$\gamma_t = \begin{cases} 0.0065 & : \quad 0 \leq t < 3 \\ 0.07 & : \quad 3 \leq t < 5 \\ 0.2 & : \quad 5 \leq t < 8 \\ 0.2 + 0.07(t-7) & : \quad 8 \leq t \end{cases}$$

#### Numerical Results

Figure 6.3 shows the expected performance profiles for anytime ACS and standard ACS.

In all trials, the quality measure (6.2) as applied in Algorithm 7 satisfied the following properties for all $t \geq 0$:

1. $0 \leq Q_t$
2. $Q_t \leq Q_{t+1}$
3. $Q_t \leq \omega(Z)$,

where $\omega(Z)$ is the cardinality of a maximum clique of $Z$. The first property follows from the definition of $Q$ and the second property from the improvement condition in Algorithm 7. The third property cannot be proven for the general case. In this test series the third property was empirically satisfied. This can be seen in the plot by observing that each black dot referring to a solution quality at a certain time step for some input graph lies below or on the line of maximum quality 1.0. In addition, $Q$ is diminishing on average for $t \geq 13$ with decreasing variance of the distribution of solution qualities.

By satisfying properties (1)-(3) in all trials, the anytime ACS always provided a result of guaranteed quality in terms of a lower bound of the optimal solution at any point in time. Together with the property of diminishing returns for $t \geq 13$, the performance profile for the anytime ACS can be used for efficient meta-control such as allocating execution times in advance in order to maximize the expected utility.

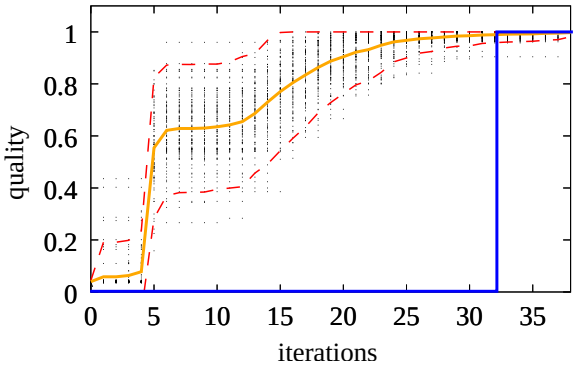



**Figure 6.3** Expected performance profile for anytime ACS (orange) and standard ACS (blue) applied to the MCP on 100 random graphs of order $2,500$ and edge probability $0.75$. The performance profiles show the average solution quality relative to the maximum solution quality. The red dashed lines indicate the confidence interval around the performance profile for anytime ACS at confidence level $\alpha = 0.05$. Black dots depict the distribution of solution qualities $Q(\boldsymbol{x}, t)$ for anytime ACS as a function of time $t$ over all 100 input instances.

### 6.4.2 A Simple Classification Task

In our second experiment, we applied anytime ACS to a simple two-category classification problem. We have chosen (6.2) as a quality measure with

$$\gamma_t = \begin{cases} 0.017 & : \quad 0 \leq t < 3 \\ 0.01 & : \quad 3 \leq t < 5 \\ 0.15 & : \quad 5 \leq t < 8 \\ 0.15 + 0.02(t-7) & : \quad 8 \leq t \end{cases}.$$

For each classification task, we first randomly generated two models $Y_1$ and $Y_2$ representing category $C_1$ and $C_2$, respectively. Both models were random graphs of order $n = 25$. The chosen edge probabilities were $p_1 = 0.3$ for $Y_1$ and $p_2 = 0.7$ for $Y_2$. Next, we selected a category $C_i$ and created a noisy copy $X$ of $Y_i$ according to the following procedure:

1. Create a copy $X$ of $Y_i$.
2. Delete each vertex of $X$ with 5% probability.
3. Insert for each vertex of $X$ a new vertex with 5% probability.
4. Connect newly created vertices to all other vertices with probability $p_i$.
5. Flip edges (non-edges) to non-edges (edges) with 2.5% probability.
6. Randomly permute $X$.

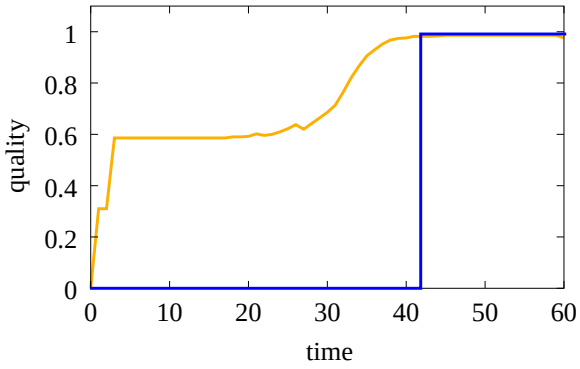



**Figure 6.4** Expected performance profile for anytime ACS (orange) and standard ACS (blue) applied to a two-category classification problem. The performance profile for both algorithms shows the average classification accuracy as a function of time.

We created 250 pairs $(Y_1, Y_2)$ of models and 125 noisy copies of each of both models. We generated new models in each trial to make the results independent of a specific pair of models. We used a nearest neighbor classifier based on the maximum common induced subgraph.

Figure 6.4 summarizes the results. From the plot we see that the classification accuracy of an anytime ACS after a few iterations is slightly better than a random guess. Until time $t = 30$, accuracy of an anytime ACS remains roughly constant. Improvement of accuracy when $t$ is in the range $[30 : 40]$ is shown in Table 6.1. According to the table, the chosen quality measure approximates the order preserving property

$$s(X, Y) > s(X, Y') \Rightarrow Q_t > Q'_t$$

at the final stage of the relaxation process of ACS. Hereby, $s(,)$ denotes a similarity measure defined by the size of a maximum common induced subgraph. The quality measures $Q_t$ and $Q'_t$ refer to problem $s(X, Y)$ and $s(X, Y')$, respectively.

This simple example shows that an anytime ACS can be used to design an agent of *bounded rationality* [333] for decision problems, which aims at finding the least-cost or best-return decision, rather than optimizing or evaluating all possible alternatives.

| t | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_t$ | 0.66 | 0.69 | 0.71 | 0.76 | 0.82 | 0.87 | 0.91 | 0.93 | 0.95 | 0.97 | 0.97 |

**Table 6.1** Average classification accuracy of anytime ACS for $t \in [30 : 40]$.

## 6.5 Conclusion

In this chapter we presented anytime Hopfield models, which can be interrupted at any time to provide a meaningful answer. Because of the particular convergence characteristics of Hopfield networks, anytime computing is limited to problems in which one is interested in maximizing an objective value of a solution rather than in the solution itself. We proposed a family of quality measures for the MCP to monitor the progress of quality improvement of an anytime Hopfield model and to construct performance profiles. In initial experiments, we indicated that neural networks for combinatorial optimization problems can be converted to anytime networks that trade solution quality for computation time.

The scene has been set for neural anytime computation in the domain of combinatorial optimization. It is beyond the scope of this work to explore anytime characteristics of neural energy minimizers in detail. To establish neural anytime computation as an alternative, extensive research on basic issues is necessary, including investigations on anytime behavior of different Hopfield models, construction of suitable quality measures and meaningful performance profiles, as well as extensive application to real-world problems. But instead of purely adopting methods from classical anytime computation, further research on neural anytime computation should aim at developing new anytime systems that more naturally fit to the paradigm of neural networks. An example is a winner-takes-all classifier for structures presented in the next chapter, which interrupts unfavorable subsystems in a self-organizing manner.

# 7 Structural Winner-Takes-All Classifiers

Nearest neighbor (NN) classifiers are the most natural and common approach in structural pattern recognition for solving multi-category problems. A template-matching procedure computes the structural proximities between a given input and all the models representing the categories. The nearest neighbor rule then assigns the input to the category represented by the nearest model. To improve the utility of an NN classifier, this chapter proposes a novel winner-takes-all (WTA) classifier. The system combines techniques from anytime computing with the principle *elimination of competition*. The basic idea is to dynamically allocate computing resources to promising input-model pairs in a self-organizing manner. This approach results in a fast classifier suitable for recognition problems with pairwise dissimilar models.

## 7.1 Introduction

Assume that we are given a set of $k$ *model* graphs

$$\mathcal{Y} = \{Y_1, \ldots, Y_k\} \subseteq \mathcal{G}_\mathcal{A}$$

together with a set of categories

$$\mathcal{C} = \{C_1, \ldots, C_k\},$$

where model $Y_i$ represents category $C_i$ for all $i \in [1:k]$. The goal is to assign an input graph $X \in \mathcal{G}_\mathcal{A}$ to one of the categories of $\mathcal{C}$ on the basis of the given preclassified models from $\mathcal{Y}$.

Because an appropriate probability model for attributed graphs is usually unknown, the most natural and common choice of a classifier is based on the nearest neighbor (NN) rule with respect to some proximity measure [30, 131, 130, 242, 310, 311, 330]. Given a structural similarity measure $s$, an NN classifier first computes or approximates the $k$ similarities $s(X, Y_1), \ldots, s(X, Y_k)$ between the input graph $X$ and all the models from $\mathcal{Y}$. This step is referred to as *template* or *prototype matching*. Next, a *maximum selector* identifies the largest similarity

$$s(X, Y_{i^*}) = \max_{i \in [1:k]} s(X, Y_i).$$

Finally, the NN rule assigns $X$ to category $C_{i^*}$ corresponding to $s(X, Y_{i^*})$.

The computational bottleneck of NN classifiers is that $k$ different graph matching problems need to be solved to classify an input $X$. Since the graph matching problem is intractable, NN classifiers applying standard graph matching algorithms[1] are too time consuming, even for moderate-scaled problems. As argued in Chapter 6, utility can be improved by replacing exact and approximate matching algorithms by anytime algorithms. It is, however, unclear when to interrupt computation to maximize the utility of an anytime NN classifier. Another cause of inefficiency is that both NN classifiers, standard and anytime, treat all matching pairs $(X, Y_i)$ on an equal footing, regardless how the matching process improves with computation time. In terms of selective attentional systems, standard and anytime NN classifiers *do not* preferentially select promising alternatives for further processing by filtering out unwanted information.

In this chapter, we propose a *winner-takes-all* (WTA) classifier for graphs within a pure connectionist framework. The key idea is to implement a mechanism of selective attention by combining concepts from anytime computation with the principle *elimination of competition*. The WTA classifier is composed of two separate layers. The bottom layer consists of $k$ subnetworks $\mathfrak{H}_i$, each of which is a Hopfield model to approximate the structural similarity $s(X, Y_i)$. The top layer is an inhibitory winner-takes-all network for maximum selection. During computation the subnetworks in the bottom layer pass evidence of their intermediate states to the top layer. Given the evidence provided by the subnetworks, the competitive WTA mechanism in the top layer focuses on promising and interrupts unfavorable subnetworks until one subnetwork $\mathfrak{H}_{i^*}$ wins the competition. The input is then assigned to the category corresponding to the winner of the competition.

This chapter is organized as follows: Section 7.2 contrasts traditional approaches of NN classifiers with the proposed WTA classifier using the Pandemonium model. In Section 7.3, we review inhibitory WTA networks for maximum selection. Section 7.4 proposes the structural WTA classifier. We present and discuss experiments in Section 7.5, before concluding the chapter with a summary and an outlook on further research.

## 7.2 The Pandemonium and the WTA Pandemonium Model

To point out the main difference between standard and anytime NN classifiers on the one hand and WTA classifiers on the other hand, we adopt a position based on the *Pandemonium model* [321].

The Pandemonium model proposed by Selfridge [321] consists of four separate layers. Each layer is composed of *demons* specialized for specific tasks. The bottom

1. As in Section 6.2.1, standard algorithms are either exact or approximate algorithms.

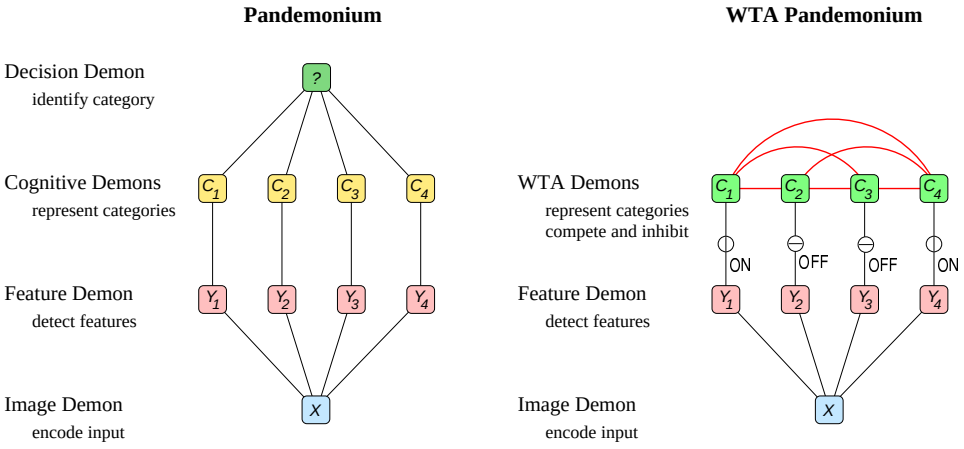



**Figure 7.1** Classical Pandemonium model (*left*) and WTA Pandemonium model (*right*) adopted for a 4-category problem in a structural domain. Communication between demons from lower layers to upper layers is indicated by solid black lines. Each feature demon is connected to a uniquely determined cognitive demon. A connected pair of a feature and cognitive demon represent a model-category pair. Note that in a more general setting, feature demons may communicate their results to more than one cognitive demon. The red lines represent the inhibitory nature of communication among WTA demons. The on-off switches in the WTA Pandemonium model indicate the principle *elimination of competition*. An on-switch means that the feature and WTA demons at the end points of the corresponding connection participate in the competition. Otherwise, if the switch is turned off, the corresponding feature and WTA demons are mute.

layer consists of *image demons* that convert the raw data to an internal representation that higher level demons can process. The second layer is composed of *feature demons* that scan the output of the image demons. When they detect certain features, they scream. The louder a feature demon screams, the more confident the demon is that it detected the feature. The third layer is composed of *cognitive demons* that represent the different categories. Cognitive demons weight the evidence provided from the screams of the feature demons and scream the amount of evidence up to the *decision demon* in the top layer. The decision demon listens to the loudest scream from the cognitive demons, and then decides what category is being presented.

Standard and anytime NN classifiers fit into the Pandemonium model. Given a structural object, an image demon converts the raw structure to an attributed graph $X$. Feature demons are implemented by standard or anytime graph matching algorithms. The features to detect are structural correspondences of $X$ with the models from $\mathcal{Y}$. Cognitive demons weight the results provided by the feature demons in terms of the underlying similarity or quality measure. The decision demon finally assigns the input graph $X$ to the category corresponding to the nearest model

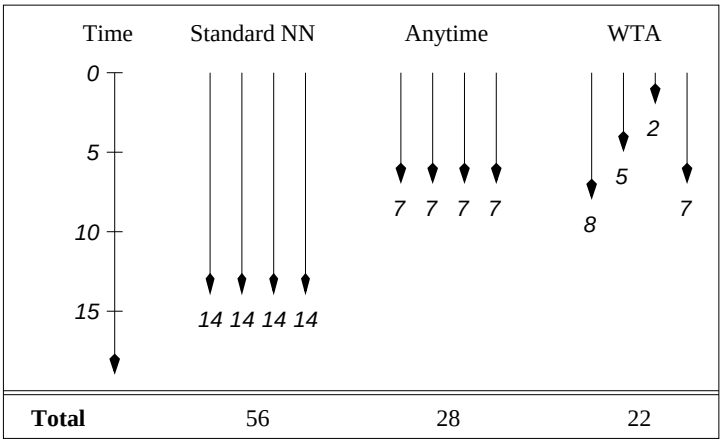



**Figure 7.2** Distribution of computation times for *standard NN*, *anytime NN*, and *WTA classifier*. Shown are the termination times of four feature demons. A feature demon terminates after convergence or interruption. Pandemonium classifiers allocate computational resources equally. WTA Pandemonium classifiers allocate computational resources dynamically in a competitive manner.

or highest quality. From the perspective of the Pandemonium model, the main difference between standard and anytime NN classifiers is only the point in time when the decision demon is able to make a decision. In the standard case, a decision can be made only after convergence of all graph matching algorithms. In anytime computing, the decision demon can decide at any time.

The WTA Pandemonium model differs from the Pandemonium model in the following way: decision making is based on self-organization rather than on a supervising decision demon. To implement self-organization, the cognitive demons directly compete among each other. The louder a cognitive demon screams, the more it intimidates (*inhibits*) its cognitive competitors. An inhibited cognitive demon becomes silent and prompts its feature demons to suspend screaming. The cognitive demon that wins the competition assigns the input to the category it represents. Thus, cognitive demons in the WTA Pandemonium are both cognitive and decisive. We occasionally refer to cognitive demons in the WTA Pandemonium model as *WTA demons*. Figure 7.1 presents a simplified schematic view of the Pandemonium model and WTA Pandemonium model in the context of structural template matching.

In contrast to the classical Pandemonium model, the WTA Pandemonium model dynamically allocates computational resources in a competitive manner. Figure 7.2 illustrates this feature, for which we assume that classification accuracy of each classifier is approximately at the same level. The main contingent of computational resources is consumed by the feature demons, each of which solves a graph matching problem. Standard and anytime NN classifier treat the feature demons equally, regardless how loudly or quietly they scream. The feature demons in the classical

Pandemonium model may either complete their computations or are all interrupted at the same time. In contrast, WTA demons compete among each other, whereby demons that scream louder result in interruption of more moderate demons. The mechanism dynamically allocates computational resources for louder demons and deprives inhibited demons. In our idealized example, the utility is maximized for WTA Pandemoniums, because it requires less total computation time for decision making than the classical Pandemonium models. Our example is idealized for two reasons. First, it is unclear when to interrupt anytime NN classifiers to obtain satisfactory results; second, formulating appropriate quality measures such that a competitive decision does not result in a substantial loss of classification accuracy is a non-trivial task.

## 7.3 Winner-Takes-All Networks for Maximum Selection

This section reviews inhibitory WTA networks for maximum selection, which constitute the foundation for designing WTA classifiers.

Assume that we are given a set of $k$ real-valued numbers

$$v_1, \ldots, v_k \in \mathbb{R}$$

with unique maximum

$$v_{i^*} = \max_{i \in [1:k]} v_i.$$

One connectionist architecture to identify the maximum value $v_{i^*}$ is an inhibitory *winner-takes-all* network called MAXNET [231]. A MAXNET is composed of $k$ pairwise inhibitory connected units. Discrete-time dynamics is of the form

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t + \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{h}_t \tag{7.1a}$$

$$\boldsymbol{x}_{t+1} = \lfloor \boldsymbol{u}_{t+1} \rfloor_0, \tag{7.1b}$$

where $\boldsymbol{u}_t$ is the activation vector and $\boldsymbol{x}_t$ the output state of MAXNET. The weight matrix $\boldsymbol{W} = (w_{ij})$ is given by

$$w_{ij} = \begin{cases} w_I & : \ i \neq j \\ 0 & : \ i = j \end{cases},$$

where the quantity $w_I < 0$ is the inhibition of MAXNET. The vector $\boldsymbol{h}_t$ represents an externally applied input. For $u \in \mathbb{R}$, the function

$$\lfloor u \rfloor_0 = \begin{cases} u & : \ u > 0 \\ 0 & : \ u \leq 0 \end{cases}$$

denotes the *linear threshold function*. The linear threshold function is an un-

bounded, piecewise linear function with lower saturation 0. As usual, $\lfloor \boldsymbol{u} \rfloor_0$ is defined componentwise on $\boldsymbol{u} \in \mathbb{R}^k$, i.e.

$$\lfloor \boldsymbol{u} \rfloor_0 = \big(\lfloor u_1 \rfloor_0, \ldots, \lfloor u_k \rfloor_0\big)^\mathsf{T}.$$

The lack of an upper saturation is necessary to exclude spurious ambiguous states [132, 231, 299, 383]. As a consequence, we have the following *MAXNET Convergence Theorem.*

***Theorem 7.1 (MAXNET Convergence Theorem)***
*Let* $\boldsymbol{u}_0 = \big(u_1(0), \ldots, u_k(0)\big)^\mathsf{T}$ *be the initial activation of MAXNET. Assume that the following conditions are satisfied:*

1. *The maximum* $u_{i^*}(0) = \max\limits_{i \in [1:k]} u_i(0)$ *is positive and uniquely determined.*
2. *The inhibition is in the range* $0 < \big|w_I\big| < \dfrac{1}{k-1}$.
3. *The external input is of the form* $\boldsymbol{h}_t = \boldsymbol{0}_k$ *for all* $t \geq 0$.

*Then the dynamics (7.1) of MAXNET converges after a finite number of iterations to a stable fixed point* $\boldsymbol{x}$ *of the form*

$$\Theta(x_i) = \begin{cases} 1 & : \quad \text{if } i = i^* \\ 0 & : \quad \text{otherwise} \end{cases}$$

*for all* $i \in [1:k]$.

***Proof*** [217]. ■

Note that we use the threshold function $\Theta$ to characterize the stable states of MAXNET.[2]

If the sufficient conditions (1)-(3) of Theorem 7.1 are satisfied, MAXNET converges to a state where the unit with highest initial activation has positive output and all other units are inhibited. This property makes MAXNET amenable to maximum selection of a given $k$-tuple $\boldsymbol{v} = (v_1, \ldots, v_k)^\mathsf{T}$ of input values. Algorithm 8 describes the MAXNET algorithm. After convergence to a stable state, the algorithm returns the index $i^*$ referring to the maximum value of $\boldsymbol{v}$. We call unit $i^*$ the *winner* of the competition.

2. Recall from Section 4.2 that the threshold function $\Theta$ is given by

$$\Theta(x) = \begin{cases} 1 & : \quad x > 0 \\ 0 & : \quad x \leq 0 \end{cases}$$

for $x \in \mathbb{R}$. In addition, $\Theta(\boldsymbol{x})$ denotes the shortcut for $\big(\Theta(x_1), \ldots, \Theta(x_k)\big)^\mathsf{T}$ for $\boldsymbol{x} \in \mathbb{R}^k$.

**Algorithm 8** (MAXNET Algorithm)

**Input:**

$\boldsymbol{v}$ – vector of $k$ real valued numbers with unique maximum
$w_I$ – inhibition
$\boldsymbol{h}$ – external input

**Initialization:**

set $\boldsymbol{u}_0 = \boldsymbol{v}$

**Procedure:**

**repeat**

$\boldsymbol{u} = \boldsymbol{u} + \boldsymbol{W}\boldsymbol{x} + \boldsymbol{h}$

$\boldsymbol{x} = \lfloor \boldsymbol{u} \rfloor_0$

**until** $\boldsymbol{e}_k^{\mathsf{T}} \Theta(\boldsymbol{x}) = 1$

**Output:** $i^*$ with $x_{i^*} > 0$

Sufficient conditions for convergence of MAXNET after a finite number of iteration steps, where the external input $\boldsymbol{h}_t = \boldsymbol{h}$ is constant over time with $\boldsymbol{h} \neq \boldsymbol{0}_k$ are shown in [184]. Similar and other techniques to select a maximum can be found in [88, 150, 214, 232, 395, 394]. In [249], maximum selection is generalized to $K$-WTA networks that identify the largest $K$ of $k$ real numbers. For further results on inhibitory WTA networks we refer to [184, 85, 217, 247, 347].

## 7.4 Structural Winner-Takes-All Classifiers

Let $s$ be a structural similarity measure on $\mathcal{G}_\mathcal{A}$. Suppose that

$$\mathcal{Y} = \{Y_1, \ldots, Y_k\} \subseteq \mathcal{G}_\mathcal{A}$$

is a set of $k$ model graphs representing the corresponding categories

$$\mathcal{C} = \{C_1, \ldots, C_k\}.$$

The task of a structural WTA classifier is to assign a given data graph $X \in \mathcal{G}_\mathcal{A}$ to one of the $k$ categories corresponding to the largest similarity

$$s(X, Y_{i^*}) = \max_{i \in [1:k]} s(X, Y_i).$$

This section proposes a self-organizing WTA classifier that combines concepts from anytime computing with the principle *elimination of competition* to improve performance over classical Pandemonium NN classifiers.

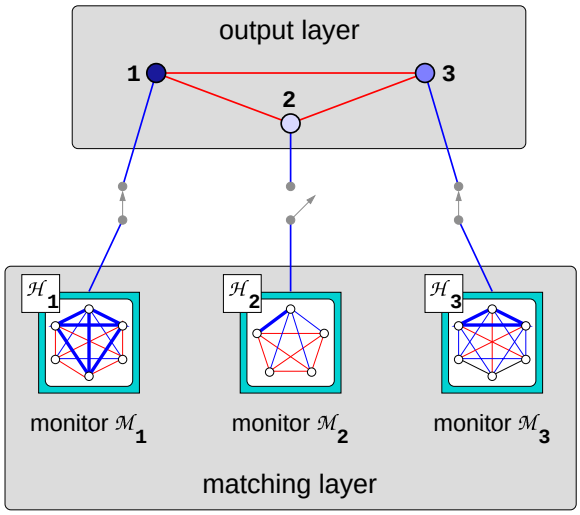



**Figure 7.3** Architecture of a structural WTA classifier for three categories. Red lines refer to inhibitory connections and blue lines to excitatory connections. Darker shading of the output units represent higher activation. In each subnetwork of the matching layer, a maximum clique of the corresponding association graph is highlighted.

### 7.4.1 Architecture

To describe the architecture, we follow a different approach than suggested by the WTA Pandemonium model in Section 7.2. A structural WTA classifier is composed of a *decision* or *output layer*, a *matching layer*, and a *coupling* of both layers. Figure 7.3 depicts an example of a structural WTA classifier. In the following, we describe the three basic components of the system.

1. *Decision layer*: The decision layer is a MAXNET architecture consisting of $k$ output units, where output unit $i$ represents category $C_i$.

2. *Matching layer*: A matching layer is composed of $k$ subnetworks and $k$ monitors.

   (a) *Subnetworks*: Each subnetwork $\mathfrak{H}_i$ is an anytime Hopfield network designed to approximately solve $s(X, Y_i)$ by a clique search in a derived association graph $X \diamond Y_i$. Note that the topology of the subnetworks in the matching layer varies dynamically with the data graph $X$ applied to the system.

   (b) *Monitors*: Conceptually, a monitor $\mathfrak{M}_i$ for $\mathfrak{H}_i$ is a device that observes the progress of $\mathfrak{H}_i$ by evaluating its output states in terms of a quality measure

$$Q_i : \mathbb{U}^{n_i} \times \mathbb{R}_+ \to \mathbb{R}, \quad (\boldsymbol{x}, t) \mapsto Q_i(\boldsymbol{x}, t),$$

where $n_i$ is the order of the association graph $X \diamond Y_i$. The quality $Q_i(\boldsymbol{x}, t)$ of $\mathfrak{H}_i$ can be viewed as an intermediate result at time $t$ for the unknown but true similarity $s(X, Y_i)$.

3. *Coupling*: Matching and decision layer are coupled via feed-forward connections. Each monitor $\mathfrak{M}_i$ is connected by a feed-forward link to output unit $i$. These links relate a subnetwork $\mathfrak{H}_i$ with its representative output unit $i$. We call a pair $\mathfrak{A}_i = (\mathfrak{H}_i, i)$ an *alternative*.
A connection between monitor $\mathfrak{M}_i$ and output unit $i$ either *persists* or is *released*. Figure 7.3 indicates the possible states of a connection by a symbolic switch. If the connection for alternative $\mathfrak{A}_i$ persists, we say $\mathfrak{A}_i$ is *enabled*. Similarly, an alternative $\mathfrak{A}_i$ is called *disabled* if the corresponding connection is released.

### 7.4.2 The Algorithm

The structural WTA algorithm proceeds first by *initializing* the activations of all units in the matching and decision layer. In addition, the connections between subnetworks and output units are established. Once the network has been properly initialized, the decision process of a structural WTA classifier involves intertwined execution of four basic steps, *matching*, *monitoring*, *competition*, and *focusing*, until one output unit wins the competition.

1. *Initialization.*
   - *Decision Layer*: The initial activation $u_i(0)$ of each output unit $i$ is set to an identical positive value $u_i(0) = a > 0$ for all $i \in [1\!:\!k]$.
   - *Matching Layer*: Units of the subnetworks $\mathfrak{H}_i$ are initialized in the same way as a decoupled network $\mathfrak{H}_i$ for approximately solving the structural similarity $s(X, Y_i)$.
   - *Coupling*: All alternatives $\mathfrak{A}_i = (\mathfrak{H}_i, i)$ are enabled. By $\mathcal{E} = [1\!:\!k]$ we denote the set of indices of enabled alternatives.

2. *Matching.* For all $i \in \mathcal{E}$, simultaneously update all subnetworks $\mathfrak{H}_i$ for one iteration step according to

$$\boldsymbol{u}_{t+1}^i = (1-d)\boldsymbol{u}_t^i + \boldsymbol{W}^i \boldsymbol{x}_t^i + \boldsymbol{h}^i \tag{7.2a}$$

$$\boldsymbol{x}_{t+1}^i = (1-\lambda)\boldsymbol{x}_t^i + \lambda g\big(\boldsymbol{u}_{t+1}^i\big). \tag{7.2b}$$

Update rule (7.2) is discussed in Section 4.2.

3. *Monitoring.* Compute the quality

$$Q_i(t) = \begin{cases} Q_i\left(\boldsymbol{x}_t^i, t\right) & : \quad i \in \mathcal{E} \\ 0 & : \quad i \notin \mathcal{E} \end{cases}$$

of state $\boldsymbol{x}_t^i$ provided by $\mathfrak{H}_i$ at time $t$ for all $i \in [1\!:\!k]$. Propagate the quality $Q_i(t)$ to output unit $i$.

4. *Competition.* Update MAXNET in the decision layer for one iteration step according to

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t + \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{q}_t \tag{7.3a}$$

$$\boldsymbol{x}_{t+1} = \lfloor \boldsymbol{u}_{t+1} \rfloor_0, \tag{7.3b}$$

where the vector $\boldsymbol{q}_t$ collects the intermediate results $Q_i(t)$ received from the matching layer. The dynamics of MAXNET is discussed in Section 7.3.

5. *Focusing.* Set

$$\mathcal{E} = \{i \in [1\!:\!k] \,:\, x_i(t+1) > 0\},$$

where $x_i(t+1)$ is the updated output state of the $i$-th unit in the decision layer. Alternatives $\mathfrak{A}_i = (\mathfrak{H}_i, i)$ with $x_i(t) > 0$ but $x_i(t+1) \leq 0$ lost the competition and are disabled. Disabling eliminates alternatives $\mathfrak{A}_i$ from further competition and interrupts the computation of their subnetworks $\mathfrak{H}_i$.

The procedure terminates when one output unit wins the competition or all subnetworks have converged to a stable state. In the latter case, we select an output unit $i^*$ as the winner of the competition if the solution found by subnetwork $\mathfrak{H}_{i^*}$ is maximal over the set of solutions returned by all the subnetworks in the matching layer.

In the following, we discuss monitoring and focusing. Both components are essential to the design of efficient and accurate structural WTA classifiers.

### 7.4.3 Monitoring

Monitoring plays a central role in a structural WTA classifier, because decision making is based on information provided by monitors. A monitor $\mathfrak{M}_i$ observes the progress of a subnetwork $\mathfrak{H}_i$ in the matching layer by computing quality values $Q_i(t)$ and propagating them to output unit $i$ at certain points in time. Constructing monitors therefore involves two design decisions: first, the particular form of a quality measure, and second, the time intervals at which a monitor observes a subnetwork.

#### *Quality Measures*

To enable fast and accurate decision making, we need appropriate quality measures. A desired property of quality measures is *discriminability*. Given an input graph $X$, we call a set $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ of quality measures *discriminable* for $X$ at $t$ if

$$X \in \mathcal{C}_{i^*} \;\Rightarrow\; Q_{i^*}(t) = \max_{i \in [1:k]} Q_i(t). \tag{7.4}$$

Note that discriminability of quality measures depends on the input and on the time. It is unclear how to formulate a set of quality measures that satisfies property

(7.4) for all inputs $X$ at some point of time $t_X$. Therefore, the best we can do is formulate problem-dependent quality measures that approximate property (7.4). But even if we know that a set $\mathcal{Q}$ of quality measures is discriminable, the black box behavior of Hopfield models provides no information as to which time the set $\mathcal{Q}$ is discriminable for a given input. Following the idiom "*it is easier to hit a fly with a swatter than with a needle*", we place the responsibility for decision making on MAXNET. For decision making, MAXNET considers quality values over some period of time (swatter), rather than quality values at a single point in time (needle) as accomplished by anytime NN classifiers.

To propose a family of quality measures, it is sufficient to consider a single subnetwork. This allows us to drop index $i$ that refers to components related to category $\mathcal{C}_i$. Let $Z = X \diamond Y$ be an association graph of model $Y$ and input $X$. By $\mathfrak{H}_Z$ we denote a Hopfield model associated with $Z$. As in Section 6.3, we define

$$\begin{aligned} V_a(\boldsymbol{x}) &= \big\{i \in V(Z) \,:\, x_i > 0\big\} \\ E_a(\boldsymbol{x}) &= \big\{(i,j) \in E(Z) \,:\, i,j \in V_a(\boldsymbol{x})\big\} \\ \overline{E}_a(\boldsymbol{x}) &= \big\{(i,j) \in \overline{E}(Z) \,:\, i,j \in V_a(\boldsymbol{x})\big\}. \end{aligned}$$

as active set of vertices, edges, and non-edges for a given output state $\boldsymbol{x}$. As usual, we identify vertices, edges, and non-edges of $Z$ with their corresponding units, excitatory, and inhibitory connections in $\mathfrak{H}_Z$. We suggest the following family of quality measures for $\mathfrak{H}_Z$

$$\begin{aligned} Q(\boldsymbol{x},t) = &\sum_{i \in V_a(\boldsymbol{x})} Q_V\big(x_i,t\big) + \sum_{(i,j) \in E_a(\boldsymbol{x})} Q_E\big(x_i,x_j,t\big) \\ &+ \sum_{(i,j) \in \overline{E}_a(\boldsymbol{x})} Q_{\overline{E}}\big(x_i,x_j,t\big) + \gamma_t, \end{aligned} \tag{7.5}$$

where $Q_V$, $Q_E$, and $Q_{\overline{E}}$ are problem dependent functions that depend on the outputs of the units and on time. The functions $Q_V$ and $Q_E$ reward active vertices and edges; the function $Q_{\overline{E}}$ penalizes active non-edges. The parameter $\gamma_t$ is an offset that prevents premature elimination from competition and resolves ambiguities. An alternative approach is to scale the quality measures appropriately.

### *Synchronization of Monitors*

Synchronization of monitors refers to specifying the time intervals at which a monitor observes a subnetwork. Synchronizing monitors is motivated by the following factors:

1. Exhaustive monitoring of the subnetworks at each time step consumes computing resources, particularly if determining the quality of a result is of non-trivial complexity.

2. Hopfield models associated to graphs of different size and structure may have different relaxation times. Since longer relaxation times usually cause slower

progress, competition may be biased towards networks with faster relaxation times.

Referring to the first issue, we can introduce a time gap during which the subnetworks evolve without being monitored. To meet the second issue, we can specify different time gaps for different relaxation times where the time gap increases with the relaxation time of a subnetwork.

MAXNET in the output layer may either evolve synchronously or asynchronously with the monitors. In synchronous mode, MAXNET is idle until all monitors in the matching layer have sent the quality values to the corresponding output units. In asynchronous mode, MAXNET continues to evolve regardless whether a new quality value has been observed or not. The external input of an output unit is then the most recent quality value of the corresponding subnetwork.

### 7.4.4 Focusing

In contrast to standard approaches where resources are equally distributed, the distinguishing feature of focusing is that resources are dynamically allocated in a competitive manner according to quality performances. Attention is shifted towards favorable subnetworks, and unfavorable subnetworks are filtered out. A subnetwork $\mathfrak{H}_i$ is considered to be favorable for a given input $X$ if the intermediate results indicate an above average match between $X$ and $Y_i$ averaged over all enabled alternatives. If a subnetwork is considered to be favorable, it may continue its computation. Unfavorable subnetworks are interrupted and the alternative is excluded from further competition. Since disabling reduces the number of alternatives, we may optionally increase the inhibition of MAXNET to adapt the pressure of competition with the number of enabled competitors. For example, we may choose the absolute value $|w_I|$ of the inhibition parameter close to $\lambda = 1/\big(|\mathcal{E}| - 1\big)$, where $|\mathcal{E}|$ is the number of enabled alternatives. According to Theorem 7.1, the quantity $\lambda$ is the time-dependent upper bound for $|w_I|$, which ensures a well-behaved convergence.

As the Attention Control System (ACS), focusing of structural WTA classifiers admits an interpretation in terms of selective attention. The main difference between ACS and structural WTA classifiers is that inhibition of MAXNET not only identifies relevant stimuli, but also shifts attention to that stimuli. We refer to Section 4.5.2 for a more detailed discussion on selective attention.

## 7.5 Experiments

This section serves to assess the performance of the WTA classifier. Section 7.5.1 presents an empirical sensitivity analysis using random attributed graphs. We examined how the WTA classifier performed under controlled variation of experimental and operational parameters. In Sections 7.5.2 and 7.5.3, we evaluated the performance of the WTA algorithm on synthetic character recognition and image

classification.

In all experiments, we applied the ACS algorithm proposed in Section 4.5.2 as a matching procedure for the WTA and NN classifier. The algorithms were implemented in Java using `JDK` 1.2. All experiments were run on a multi-server `Sparc SUNW Ultra-4`.

### 7.5.1 Sensitivity Study

The aim of this section is to evaluate how the WTA classifier performs under controlled variation of

1. the pairwise similarities $s(Y_i, Y_j)$ among all the models $Y_i, Y_j \in \mathcal{Y}$,
2. the inhibition parameter $w_I$ of MAXNET,
3. the number $k = |\mathcal{Y}|$ of models, and
4. the size $n$ of models

Variation of structural corruption and noise imposed on the input graphs is covered in the experiment on synthetic character recognition presented in Section 7.5.2.

In all four experiments, we assumed the following basic test suite. The default parameters of the test suite are summarized in Table 7.5.1.

#### *Test Suite*

In all experiments, we considered random attributed graphs with edge probability $p = 0.5$ and discrete attribute set $\mathcal{A} = \{0, 1, 2\}$, where $\epsilon = 0$ is the null attribute. We used an additional attribute $a = 2$ to introduce more diversity into the graphs. To generate the data for a single classification task, we applied the following procedure:

1. Generate a base graph $B$ of order $|B| = n$. The base graph served as the starting point for deriving the models.
2. Create $k$ models $Y_i \in \mathcal{Y}$ by applying a corruption model on the base graph $B$ with noise probability $P_{\mathcal{Y}}$. For low (high) values of $P_{\mathcal{Y}}$, the corruption model produces a set $\mathcal{Y}$ of models with high (low) pairwise similarities. The corruption model is described below.
3. Randomly select a category $C_i$ from $\mathcal{C}$.
4. Generate an input graph $X$ of the selected category $C_i$ by applying the same corruption model as in step 2 on the model $Y_i$ with noise probability $P_{\mathcal{X}}$.

In each trial, we generated a new base graph and new models to ensure that the results are independent of a particular set of models.

Given a graph $Z$ and a noise probability $P$, the following corruption model generates a perturbed copy of $Z$:

- Recolor each item $\boldsymbol{i} \in I(Z)$ with probability $P$:
  1. Draw a random number $\rho$ from a uniform distribution over $\mathbb{U}$.

| Default Setting | |
|---|---|
| $n = 50$ | order of base graph $B$ |
| $k = 5$ | number of models |
| $P_{\mathcal{Y}} = 5\%$ | noise probability for models |
| $P_{\mathcal{X}} = 1\%$ | noise probability for input graphs |
| $w_I = -\frac{1}{\lvert\mathcal{E}\rvert}$ | inhibition of MAXNET |

**Table 7.1** Default parameters of the basic test suite for the sensitivity study in Section 7.5.1. Note that the inhibition $w_I$ is adapted to the number $|\mathcal{E}|$ of enabled alternatives.

2. If $\rho < P$, randomly select an attribute $a$ from $\mathcal{A}$.
3. Recolor item $\boldsymbol{i}$ with attribute $a$.

The above corruption model introduces noise in the attributes and structural variation in the number of vertices and edges. This is achieved by random insertion, deletion, and attribute substitution of edges and vertices. We imposed the following restriction: no vertices were inserted when generating the models from the base graph $B$. Thus, the expected order of the models was $\left(1 - \frac{1}{3}P_{\mathcal{Y}}\right) n$, where $n = |B|$. To generate an input graph $X$ as a corrupted version of a model $Y_i$, the order of $X$ was bounded from above by $n$.

***Setting of the Classifiers***

Let $X$ be an input and $Y$ be a model graph from $\mathcal{G}_{\mathcal{A}}$. For decision making, we selected the following similarity and quality measure:

1. *Similarity measure for the NN classifier*: The chosen similarity measure for the NN rule is the order of the maximum common induced subgraph of two given graphs.

2. *Quality measure for the WTA classifier*: Let $Z$ be an association graph of $X$ and $Y$, and let $\mathfrak{H}_Z$ be the Hopfield model associated with $Z$. For the WTA classifier, we used the quality measure (7.4) with component functions

$$\begin{aligned} Q_V(x_i, t) &= x_i \\ Q_E(x_i, x_j, t) &= 0 \\ Q_{\overline{E}}(x_i, x_j, t) &= -\frac{1}{2}(x_i + x_j) \\ \gamma_t &= 0.1. \end{aligned}$$

Note that $x_i$ refers to the output of unit $i$ in $\mathfrak{H}_Z$. It is easy to see that the compound quality measure

$$Q = Q_V + Q_E + Q_{\overline{E}} + \gamma_t$$

is directly related to the similarity measure of the NN classifier. After convergence, $Q - \gamma_t$ is the size of a maximal clique in $Z$ found by $\mathfrak{H}_Z$. Since maximal cliques in $Z$ are in one-to-one correspondence with the maximal common induced subgraphs of $X$ and $Y$, the relationship between the chosen similarity and quality measure is evident.

***Evaluation Procedure***

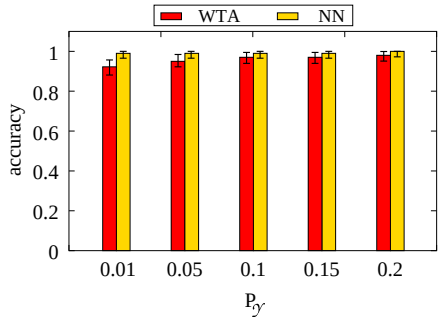
In all four experiments, we varied a single parameter as follows:

- *Experiment 1: Varying Pairwise Similarity of the Models*
  To generate models with different expected pairwise similarities, we varied the noise probability $P_{\mathcal{Y}}$. The chosen values were $P_{\mathcal{Y}} = \{1\%, 5\%, 10\%, 15\%, 20\%\}$. For each value $P_{\mathcal{Y}}$, we generated 500 test instances, giving a total of $2,500$ trials.
- *Experiment 2: Varying the Inhibition Parameter of MAXNET*
  The chosen values for the inhibition parameter $w_I = -\lambda/k = \lambda/5$ were determined by selecting $\lambda \in \{0.1\} \cup [0.25 : 0.25 : 2.5]$. In contrast to all other experiments, we kept the inhibition $w_I$ fixed to the value $-\lambda/k$ throughout a run, rather than adapting it to the number of enabled alternatives (see Table 7.5.1). We generated 500 test instances for each value of $\lambda$, giving a total of $5,500$ trials. Since an NN classifier has no inhibition parameter, we only examined the behavior of the WTA classifier.
- *Experiment 3: Varying the Number of Models*
  The chosen numbers of models were $k \in [2 : 8]$. For each number $k$, we generated 500 test instances, giving a total of $3,500$ trials.
- *Experiment 4: Varying the Order of Models*
  We varied the expected order of the models via the order of the base graph. The selected orders $n$ of the base graph were $n \in [10 : 10 : 50]$. For each $n$, we generated 500 test instances, giving a total of $2,500$ trials.

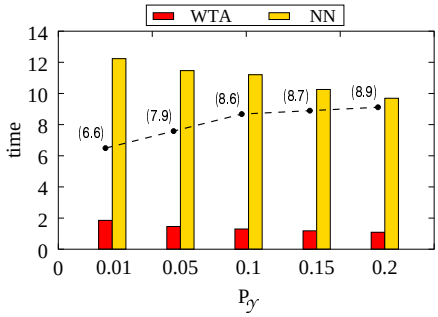
***Numerical Results***

Figures 7.4–7.7 summarize the results. In all four experiments, we made the following observations:

- On average, the NN classifier more accurately predicts the correct category than the WTA classifier.
- As expected, the WTA classifier is significantly faster than the NN classifier.
- Computation time of the WTA and NN classifier deviated roughly 15-20% and 10%, respectively, from the average. This result indicates that elimination of competition leads to a less predictable run time behavior than convergence of subnetworks to a stable state.

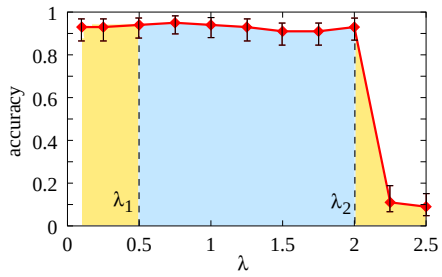



(a) Estimated classification accuracy

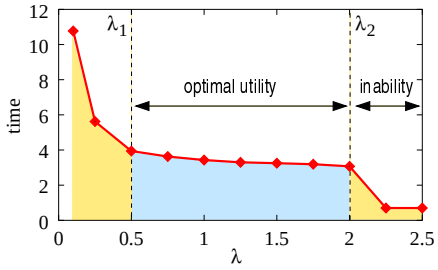



(b) Average computation time

**Figure 7.4** Results of *Experiment 1* for varying noise probability $P_y$. Low values of $P_y$ correspond to high inter similarity of the models. *Subfigure (a)*: estimated classification accuracy. Capped bars indicate the confidence intervals at level $\alpha = 0.05$. *Subfigure (b)*: average computation time measured in *sec*. The dashed line shows the average factor by which the WTA classifier was faster than the NN classifier.

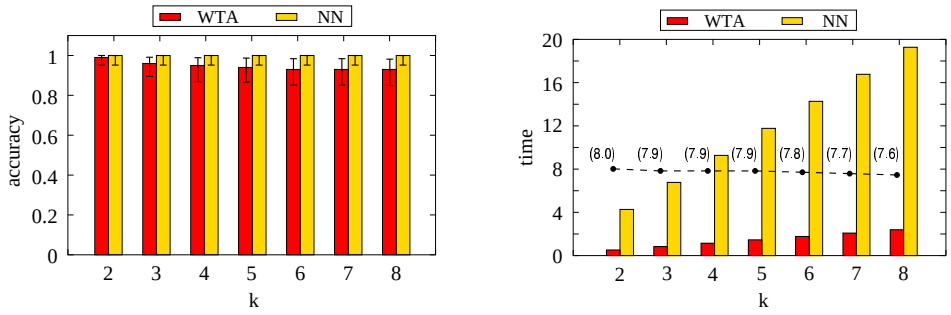



(a) Estimated classification accuracy



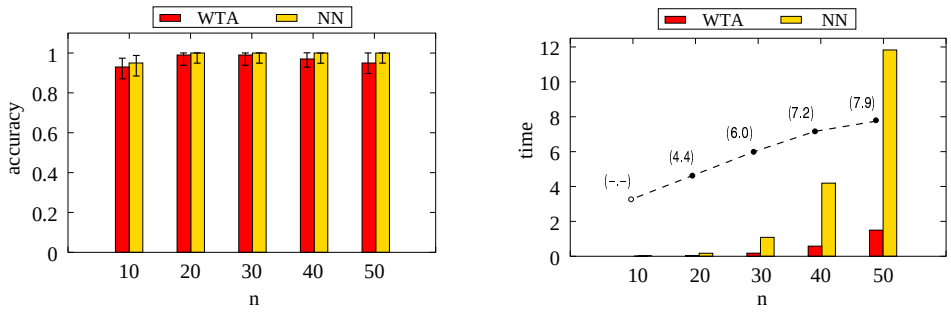

(b) Average computation time

**Figure 7.5** Results of *Experiment 2* for varying inhibition parameter $w_I$. Variation of $w_I$ is controlled by $\lambda$. Shown are the estimated classification accuracy (a) and the average computation time (b) measured in *sec*. Capped bars indicate the confidence intervals at level $\alpha = 0.05$. The light blue shaded region shows the range of optimal utility.




(a) Estimated classification accuracy (b) Average computation time

**Figure 7.6** Results of *Experiment 3* for varying number $k$ of models. *Subfigure (a)*: estimated classification accuracy. Capped bars indicate the confidence intervals at level $\alpha = 0.05$. *Subfigure (b)*: average computation time measured in *sec*. The dashed line shows the average factor by which the WTA classifier was faster than the NN classifier.




(a) Estimated classification accuracy (b) Average computation time

**Figure 7.7** Results of *Experiment 4* for varying order $n$ of the models. *Subfigure (a)*: estimated classification accuracy. Capped bars indicate the confidence intervals at level $\alpha = 0.05$. *Subfigure (b)*: average computation time measured in *sec*. The dashed line shows the average factor by which the WTA classifier was faster than the NN classifier. For $n = 10$, average times were less than system accuracy (1 *msec*) and therefore extrapolated.

Next, we discuss the particularities that occured in the individual experiments:

- *Experiment 1: Varying Pairwise Similarity of the Models*

For increasing pairwise dissimilarity of the models, classification accuracy of the WTA classifier approached the performance of the NN classifier. At the same time, the factor by which WTA was faster than NN increased.

- *Experiment 2: Varying the Inhibition Parameter of MAXNET*

There were two critical values $\lambda_1 < \lambda_2$ for the parameter $w_I$. The critical values determined the range of optimal utility of the WTA classifier. Utility was maximized if the absolute value $|w_I|$ of the inhibition was chosen from the range $[\lambda_1/5, \lambda_2/5]$. In this case, classification accuracy was at a constant but high level. In addition, computation time was low and slightly decreased for increasing $|w_I|$. For $|w_I| < \lambda_1/5$, inhibition was too low and computation time rapidly increased with decreasing $|w_I|$ without gain in classification accuracy. At the other extreme, when $|w_I| > \lambda_2/5$, the inhibition was too strong and the WTA classifier rapidly terminated in almost all cases to a state where all output units of MAXNET were inhibited. Hence, in those cases the WTA classifier was unable to make a decision.

Note that $|w_I| = \lambda_2/5$ is larger than the upper bound provided by Theorem 7.1. Nevertheless, the critical value $\lambda_2$ typically decreases with increasing number $k$ of models. On the other hand, the smaller critical value $\lambda_1$ is independent from $k$. Hence, we may expect that the range of optimal utility diminishes with increasing $k$. In addition, for increasing values of $k$, the principle elimination of competition becomes more and more ineffective. As a consequence, run time performance of the WTA classifier declines.

- *Experiment 3: Varying the Number of Models*

As predicted in Experiment 2, the WTA classifier scaled inferior with the number of models than the NN classifier did with respect to both accuracy and run time. Since we adapted the inhibition to the number of enabled alternatives during run time, the decline of time performance is kept within an acceptable limit.

- *Experiment 4: Varying the Order of Models*

Although the differences in accuracy are not statistically significant, they may indicate that prediction of the WTA classifier is best for a problem dependent size of the underlying graphs. For graphs too small, the perturbance model might yield highly similar graphs. This explains the larger error of both classifiers for models of order $n = 10$. For graphs too large, differences in the quality measure may disappear at initial stages of the computation due to the large normalization factor and the linear threshold function. The WTA classifier may therefore favor the subnetwork with the fastest relaxation time. From the perspective of run time, the results show that the WTA classifier scales better with the size of the models than the NN classifier.

The conclusion we draw from the results is that the WTA classifier performed best for a small number of pairwise dissimilar models. To cope efficiently with a

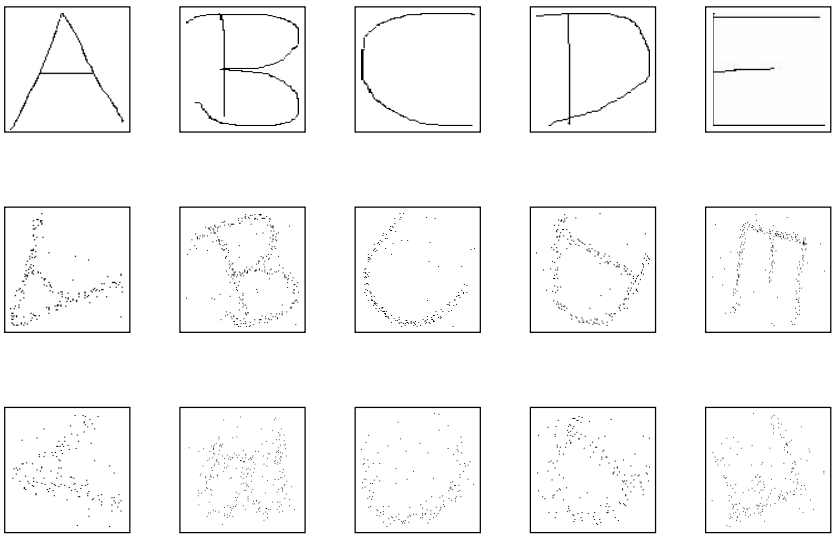


**Figure 7.8** Shown are examples of handwritten characters and corrupted data. The corruption model randomly rotates the image, inserts spurious points, deletes points, and adds Gaussian noise with zero mean and standard deviation $\sigma$. Row 1: model characters. Row 2: sample data characters corrupted with standard deviation $\sigma = 4$. Row 3: sample data characters corrupted with standard deviation $\sigma = 8$.

large number of models, we suggest construction of hierarchically organized WTA classifiers, in which each classifier considers only a small subset of all models.

### 7.5.2 Synthetic Character Recognition

In order to test the WTA algorithm in a more applied setting, we examined the problem of synthetic character recognition based on finding correspondences between model and data. This problem frequently arises in computer vision [127, 291]. The goal is to investigate the robustness of the WTA classifier against corruption in the input data.

#### *Problem Formulation*

Consider five models of handwritten characters $\mathcal{Y} = \{$'A', 'B', 'C', 'D', 'E'$\}$ as shown in the first row of Figure 7.8. The models were drawn using an X windows interface. The black pixels of each model were expressed as a set of points in the 2D plane.

The recognition problem consists of assigning a corrupted input image $X$ to one of the five model characters. A corrupted version $X$ of a given model $Y \in \mathcal{Y}$ was generated according to the following procedure:

1. Copy $Y$ to $X$.

| Order of Models | | | | |
|---|---|---|---|---|
| A | B | C | D | E |
| 335 | 486 | 280 | 311 | 409 |

**Table 7.2** Shown is the order of the graphs representing the model characters.

2. Randomly rotate $X$ around its barycenter. The barycenter of an image is the mean of its points in the 2D plane.

3. Insert a new spurious point for each point with 10% probability. A new point is drawn from a uniform distribution over the dimensions of the image.

4. Delete each point of $X$ with 10% probability. Spurious points generated in the previous step are excluded from deletion.

5. Add Gaussian noise with a zero mean and standard deviation $\sigma$ from the coordinates of each point of $X$.

The second and third rows of Figure 7.8 show examples of corrupted data images generated by the above procedure for $\sigma = 4$ and $\sigma = 8$, respectively.

***Graph-based Representation***

Let $Z$ be an image with point set $\mathcal{P}_Z = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_p\}$. Each element $\boldsymbol{z}_i$ represents a black pixel by its coordinates in the 2D plane. We transformed the point set $\mathcal{P}_Z$ to a complete weighted graph $G_Z$ – invariant to rotation, translation, and scaling.

The vertices of $G_Z$ represented the points from $\mathcal{P}_Z$. To determine vertex and edge weights, we first computed the Euclidean distances

$$d_Z(i,j) = \begin{cases} \|\boldsymbol{z}_i - \mu_Z\| & : \quad i = j \\ \|\boldsymbol{z}_i - \boldsymbol{z}_j\| & : \quad i \neq j \end{cases}, \qquad \forall\, i,j \in [1:p],$$

where $\mu_Z$ denotes the mean of $\mathcal{P}_Z$ (or the barycenter of the image). The distance $d_Z$ is invariant to rotation and translation of $Z$. To obtain scale invariance and make the representation robust against noise, we first ordered the distances $d_Z(i,j)$. Let $r$ be the rank of distance $d_Z(i,j)$. Then the weight assigned to item $(i,j) \in I(G_Z)$ was the normalized rank $2r/(n(n-1))$, where $r \in [1 : n(n-1)/2]$. Since $d_Z(i,j) = d_Z(j,i)$, the adjacency matrix of $G_Z$ is symmetric.

Table 7.2 shows the order of the graphs representing the model characters. Since the probability of deleting and inserting points was equal, the expected order of corrupted graphs derived from model $Y$ coincided with the order of $Y$.

In the following, we identify the notation for the image $Z$, the point set $\mathcal{P}_Z$, and its graph-based representation $G_Z$. As a common notation, we adopt the notation for the image. From the context it will be clear which representation form is addressed.

***Evaluation Procedure***

The chosen values for the standard deviation to perturb the position of the points were $\sigma \in [1:10]$. For each pair $(Y_i, \sigma)$, we created 100 corrupted data characters giving a total of $5,000$ classification tasks.

***Setting of the Classifiers***

Let $X$ and $Y$ be complete weighted graphs. For decision making, we selected the following similarity and quality measure.

1. *Similarity measure for the NN classifier*: Consider the compatibility values

$$\kappa_{\boldsymbol{ij}} = \begin{cases} \lfloor 1 - \alpha_V |x_{\boldsymbol{i}} - y_{\boldsymbol{j}}| \rfloor_0 & : \quad \boldsymbol{i} \in V(X), \boldsymbol{j} \in V(Y) \\ \lfloor 1 - \alpha_E |x_{\boldsymbol{i}} - y_{\boldsymbol{j}}| \rfloor_0 & : \quad \boldsymbol{i} \in E(X), \boldsymbol{j} \in E(Y) \end{cases}$$

for all items $\boldsymbol{i} \in I(X)$ and $\boldsymbol{j} \in I(Y)$. Note that $X$ and $Y$ are complete. Hence, the compatibilities between all non-vertex items are specified.
The parameters $\alpha_V$ and $\alpha_E$ control which vertex and edge correspondences shall contribute to the structural similarity measure. We have chosen $\alpha_V = 1000$ and $\alpha_E = 100$. The selected similarity measure is determined by the following optimization problem

$$\begin{aligned} &\text{maximize} && f(\phi, X, Y) = \frac{1}{\max\left(|X|, |Y|\right)} \sum_{\boldsymbol{i} \in I(X)} \kappa_{\boldsymbol{i}\boldsymbol{i}^{\phi}} \\ &\text{subject to} && \phi \in \mathcal{M}, \end{aligned} \tag{7.6}$$

where $\mathcal{M}$ is the subset of all partial monomorphisms from $X$ to $Y$. Since $\mathcal{M}$ is $\mathfrak{p}$-closed, we can transform problem (7.6) to an equivalent MWCP in an association graph (see Theorem 3.1).

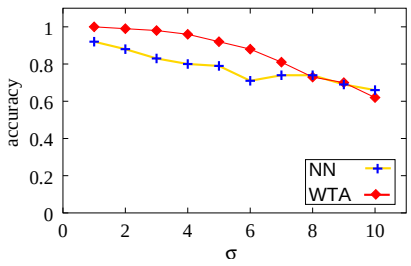
2. *Quality measure for the WTA classifier*: Let $Z$ be an association graph of $X$ and $Y$ for problem (7.6), and let $\boldsymbol{Z} = (z_{ij})$ be the adjacency matrix of $Z$. We used the quality measure (7.4) with component functions

$$\begin{aligned} Q_V(x_i, t) &= x_i \\ Q_E(x_i, x_j, t) &= z_{ij} \\ Q_{\overline{E}}(x_i, x_j, t) &= -\frac{1}{2}(x_i + x_j) \\ \gamma_t &= 0.1. \end{aligned}$$

Again, $x_i$ refers to the output of unit $i$ in $\mathfrak{H}_Z$. The compound quality $Q$ was of the form

$$Q = \left\lfloor \frac{1}{|Z|^2}(Q_V + Q_E + Q_{\overline{E}}) \right\rfloor_0 + \gamma_t,$$

where we included a normalization factor. In addition, the linear threshold




(a)

| | WTA | | NN | |
|---|---|---|---|---|
| $\sigma$ | $\alpha$ | $[\alpha_1, \alpha_2]$ | $\alpha$ | $[\alpha_1, \alpha_2]$ |
| 1 | 1.00 | [0.96, 1.00] | 0.92 | [0.85, 0.96] |
| 2 | 0.99 | [0.95, 1.00] | 0.88 | [0.80, 0.93] |
| 3 | 0.98 | [0.93, 1.00] | 0.83 | [0.74, 0.89] |
| 4 | 0.96 | [0.90, 0.99] | 0.80 | [0.71, 0.87] |
| 5 | 0.92 | [0.85, 0.96] | 0.79 | [0.70, 0.86] |
| 6 | 0.88 | [0.80, 0.93] | 0.71 | [0.61, 0.79] |
| 7 | 0.81 | [0.72, 0.88] | 0.74 | [0.65, 0.82] |
| 8 | 0.73 | [0.63, 0.81] | 0.74 | [0.64, 0.81] |
| 9 | 0.70 | [0.60, 0.78] | 0.69 | [0.59, 0.78] |
| 10 | 0.62 | [0.52, 0.71] | 0.66 | [0.56, 0.75] |

(b)

**Figure 7.9** Estimated classification accuracy of the WTA classifier (*red line*) and the NN classifier (*golden line*) for noisy characters as a function of the noise level $\sigma$. The table in (b) shows the numerical values of the average classification accuracy and the confidence intervals at level $\alpha = 0.05$.

function $\lfloor . \rfloor_0$ ensured that only nonnegative values were sent to the decision layer. As in Section 7.5.1, it is easy to establish a relationship between the chosen similarity and quality measure.

### *Numerical Results*

The average computation times were 24 *msec* for the WTA and 250 *msec* for the NN classifier. The average was derived from all 5,000 trials.

Figure 7.9 presents the average classification accuracy of both classifiers as a function of the standard deviation $\sigma$. As expected, accuracy of both classifiers degraded with increasing corruption of the input data. For low noise levels $\sigma$, the WTA classifier was superior to the NN classifier. It is unclear whether the larger error rate of the NN classifier was caused by a wrong choice or poor approximation of the similarity measure. Other choices of similarity measures did not improve the results of NN. This result underpins the importance of choosing appropriate quality and similarity measures, which are not only problem dependent, but also tailored to the characteristics of the algorithm.

As a final remark, we note that the *state of the art* algorithm for computing the underlying similarity measure is the *Graduated Assignment* algorithm [116]. Using the standard setting, an NN classifier employing the Graduated Assignment algorithm failed for this problem. Average computation time for a single classification task was about 20 *min*. Thus, for a single match, Graduated Assignment required about 4 *min*. The NN classifier using the ACS procedure was about 5,000 times faster. The factor $\alpha_V$ in the formulation of the similarity measure eliminated most

potential vertex correspondences so that the resulting association graphs are kept small. Since almost all vertices of both graphs under consideration had at least one match, Graduated Assignment compared graphs of roughly the same order as shown in Table 7.2. In contrast, ACS solved a clique problem in a dense association graph of order up to 800 (where almost all vertices of both graphs were involved). The complexity of that problem corresponds to a graph matching problem, in which the graph to be compared are both of order $\sqrt{800} \approx 28$. To contrast the different time performances, we note that the WTA classifier was about $50,000$ times faster than the Graduated Assignment algorithm. For this reason, we excluded the Graduated Assignment algorithm for comparison with the WTA classifier.

### 7.5.3 Image Recognition

In our final experiment, we applied the WTA classifier to an image recognition problem. We used two subsets of the segmented images described in Section 4.6.2 and Appendix B.2.1.

#### *Evaluation Procedure*

We conducted two test series. The first test series was a 4-category and the second test series an 8-category problem. The segmented images of both problems and their categories are shown in Figure 7.10 and 7.11.

The evaluation procedure was identical for both test series. For one cycle, we first randomly selected a model from each category. Next, we presented each image to the WTA and NN classifier. We conducted 100 cycles giving a total of $2,600$ trials for the 4-category and $3,600$ trials for the 8-category problem. We ensured that each cycle used a different set of models.

#### *Setting of the Classifiers*

For decision making, we selected the following similarity and quality measure :

1. *Similarity measure for the NN classifier*: For the NN rule, we have chosen the Bischoff-Reuß-Wysotzki measure. To approximate that measure, we applied the technique of merged association graphs. Both the measure and the merging technique are described in Section 4.6.3.
2. *Quality measure for the WTA classifier*: For the WTA classifier, we used the same quality measure as in Section 7.5.2 based on the merged association graph of the Bischoff-Reuß-Wysotzki measure.

#### *Numerical Results*

Table 7.3 summarizes the results of both test series. In line with experiment 3 of Section 7.5.1, performance of the WTA classifier was satisfactory for the 4-category


**Figure 7.10** Dataset of 26 segmented images from 4 different categories.


**Figure 7.11** Dataset of 36 segmented images from 8 different categories.

| *error* | WTA | NN |
|---|---|---|
| 4-categories | 0.5% | 0.4% |
| 8-categories | 9.8% | 1.6% |

| *time* | WTA | NN |
|---|---|---|
| 4-categories | 14 | 82 |
| 8-categories | 31 | 95 |

**Table 7.3** Results of the 4-category and 8-category problem. Shown are the percentages of misclassifications of the WTA and NN classifier (*left*) and their average computation time in *msec* (*right*).

| WTA | | | | |
|---|---|---|---|---|
| | FA | FI | NR | AS |
| FA | 0 | 0 | 0 | 0 |
| FI | 0 | 0 | 0 | 0 |
| NR | 0 | 0 | 0 | 0.5 |
| AS | 0 | 0 | 0 | 0 |

| NN | | | | |
|---|---|---|---|---|
| | FA | FI | NR | AS |
| FA | 0 | 0 | 0 | 0 |
| FI | 0 | 0 | 0 | 0 |
| NR | 0 | 0 | 0 | 0.4 |
| AS | 0 | 0 | 0 | 0 |

**Table 7.4** Results of the 4-category problem. Shown are the confusion matrices for the WTA classifier (*left*) and the NN classifier (*right*). An entry in the $i$-th row and $j$-th column shows the percentage with which an image of category $Y_i$ was identified as an image of category $Y_j$. The acronyms represent the following categories: FA = FARMER, FI = FISH, NR = NEWSREADER, and AS = ASIAN.

problem and degraded in the 8-category problem with respect to accuracy and speed. In particular, the average factor by which the WTA classifier was faster than the NN classifier declined to about 3, which was the lowest of all experiments. To explain this phenomenon, we consider the average time performance of the NN classifier. From Table 7.3, we see that the NN classifiers required an average of 82 *msec* to make a decision in the 4-category problem, and 95 *msec* in the 8-category problem. Thus, doubling the number of models did not double the average computation time as is the case for the WTA classifier. Since the NN classifier spent most of its computation time matching input-model pairs from the same category, adding further models from the four other categories did not strongly influence the time performance. Thus, we have a similar effect as in experiment 4 of Section 7.5.1.

The confusion matrices of both classifiers for the 4-category and 8-category problem are presented in Tables 7.4, 7.5, and 7.6. In the 4-category problem, misclassification of both classifiers was caused by mistaking images from the category NEWSREADER for images from the category ASIAN. As opposed to the 4-category problem, causes for misclassifications of both classifiers differ in the 8-category problem. This shows that elimination of competition behaves differently than maximum selection, despite the fact that the quality measure was derived from the chosen similarity measure.

To summarize, the results confirm our previous observations on random data that the WTA classifier satisfactorily performed for a small number of pairwise dissimilar models.

| | WTA | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | FA | FI | N1 | N2 | OD | MA | AS | CA |
| FA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| FI | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N1 | 0 | 0 | 0 | 2.4 | 0 | 0 | 0 | 0 |
| N2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OD | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| AS | 0 | 0 | 0.1 | 6.0 | 0 | 0 | 0 | 0 |
| CA | 0 | 0 | 0 | 0.3 | 0 | 1.0 | 0 | 0 |

**Table 7.5** Results of the 8-category problem. Shown is the confusion matrix for the WTA classifier. Entries represent percentages of misclassifications. The acronyms represent the following categories: FA = FARMER, FI = FISH, N1 = NEWSREADER1, N2 = NEWSREADER2, OD = OUTDOOR, MA = MAN, AS = ASIAN, and CA = CAMERA.

| | NN | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | FA | FI | N1 | N2 | OD | MA | AS | CA |
| FA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| FI | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N1 | 0 | 0 | 0 | 1.0 | 0 | 0.1 | 0.3 | 0 |
| N2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OD | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| AS | 0 | 0 | 0 | 0.2 | 0 | 0 | 0 | 0 |
| CA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 7.6** Results of the 8-category problem. Shown is the confusion matrix for the NN classifier. Matrix entries and acronyms have the same meaning as in Table 7.5.

## 7.6 Conclusion

In this chapter, we presented and examined a WTA classifier for structures. The WTA classifier implements a mechanism of selective attention by combining techniques from anytime computing with the principle *elimination of competition*. In contrast to traditional approaches, computing resources are dynamically allocated in a competitive manner. subnetworks with high quality are considered to be promising and may proceed with their computation, whereas subnetworks with poor quality are interrupted and excluded from further competition.

In extensive empirical studies we showed that the WTA classifier is best suited for a small number of pairwise dissimilar models. There are two disadvantages inherent to the WTA approach. The first problem is that formulating an appropriate quality measure is complicated and highly problem dependent. The second dilemma is that

the principle elimination of competition is a pure heuristic, which is difficult to analyze in mathematical terms.

We conclude this chapter with an outlook on further research. To cope with a large number of models, construction of hierarchically organized WTA classifiers may provide a solution. In addition, extension to K-nearest neighbor competition is straightforward. First investigations are presented in [174]. Finally, learning appropriate models for the WTA classifier is considered in the next part of this thesis.

# II Learning Machines for Structures

# 8 Introduction to Structural Learning

The aim of this chapter is to provide a brief introduction to the basic settings of supervised and unsupervised learning. Furthermore, its objective is to motivate structural neural learning machines by discussing the shortcomings of some existing methods in structural learning. Sections 8.1 and 8.2 briefly review the basic supervised and unsupervised learning task. In Section 8.3, we discuss selected methods commonly used in structural learning to motivate structural neural learning machines.

## 8.1 Supervised Learning

This section is devoted to a brief overview of the basic setting of supervised learning.

Let $\mathcal{X} \subseteq \mathcal{G}_{\mathcal{A}}$ be an input domain of attributed graphs, and let $\mathcal{Y}$ be a set of output values. Assume that we are given a *training sample*

$$\mathcal{Z} = \big\{(X_1, y_1), \ldots, (X_p, y_p)\big\} \subseteq \mathcal{X} \times \mathcal{Y}$$

consisting of $p$ *training graphs* $X_i \in \mathcal{X} \subseteq \mathcal{G}_{\mathcal{A}}$ drawn from some *input space* $\mathcal{X}$ together with corresponding output values $y_i \in \mathcal{Y}$. For the sake of convenience, we assume that the sample $\mathcal{Z}$ is drawn independently and identically distributed (iid) according to some unknown distribution function $F_{\mathcal{X}\mathcal{Y}}$. On the basis of the given sample $\mathcal{Z}$, the goal of *supervised learning* is to find a function (*hypothesis*)

$$H : \mathcal{X} \to \mathcal{Y}$$

from a *hypothesis space*[1] $\mathcal{H}$ that *best* predicts the output values of unseen data $(X, y)$ according to some optimality criterion.

To precisely specify an optimality criterion for learning, we introduce the notion of *loss* or *risk*. A *loss function*

$$l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$$

1. A hypothesis space is simply a set of functions (hypotheses) that are considered by a particular learning machine.

is a function that measures the discrepancy $l(x, y)$ between the actual output value $H(X) = x$ and the desired output value $y$. Based on a loss function $l$, the *risk functional*

$$R : \mathcal{H} \to \mathbb{R}, \quad H \mapsto R[H] = E_{\mathcal{XY}}\big[l(H(\mathcal{X}), \mathcal{Y})\big]$$

is useful to measure the quality of a given hypothesis $H \in \mathcal{H}$. Now we are able to present a formal definition of supervised learning. For a given iid sample $\mathcal{Z}$, the goal is to find a hypothesis $H^* \in \mathcal{H}$ such that

$$H^* = \arg\min_{H \in \mathcal{H}} R[H]. \tag{8.1}$$

Thus, supervised learning minimizes the risk functional $R$ over the hypothesis space $\mathcal{H}$ on the basis of a training sample $\mathcal{Z}$.

The main problem of supervised learning as defined in (8.1) is that we are in general unable to evaluate the risk functional $R[H]$ for a given hypothesis $H$, because the joint distribution $F_{\mathcal{XY}}$ is unknown. The only information at our disposal is contained in the training sample $\mathcal{Z}$. To overcome this problem, we use the *inductive principle of empirical risk minimization* [373]. The basic idea of this principle is to estimate the unknown risk functional $R[H]$ of a given hypothesis $H \in \mathcal{H}$ by the *empirical risk*

$$R_{\text{emp}}[H, \mathcal{Z}] = \frac{1}{p} \sum_{i=1}^{p} l\big(x_i, y_i\big),$$

where $x_i = H(X_i)$ is the output of hypothesis $H$ given the input data $X_i$. The empirical risk function solely relies on the information contained in the training sample $\mathcal{Z}$. The principle of empirical risk minimization then approximates the hypothesis $H^*$ of (8.1) by

$$H^*_{\text{emp}} = \arg\min_{H \in \mathcal{H}} R_{\text{emp}}[H, \mathcal{Z}].$$

It turns out that empirical risk minimization does not guarantee a small actual risk, if the number $p$ of training examples is limited. To obtain a consistent learning principle, uniform convergence as defined by

$$\lim_{|\mathcal{Z}| \to \infty} P\left(\sup_{H \in \mathcal{H}} \big|R[H] - R_{\text{emp}}[H, \mathcal{Z}]\big|\right) = 0$$

is a necessary and sufficient condition.

In this chapter, we focus on two types of supervised learning problems, *classification* and *function approximation*. The former problem can be viewed as a special case of the latter one, where the function to be approximated takes values from a finite discrete set.

## 8.2 Unsupervised Learning

In this section, we briefly describe the basic setting of unsupervised learning.

Suppose that we are given a training sample

$$\mathcal{X} = \{X_1, \ldots, X_p\} \subseteq \mathcal{Z}$$

consisting of $p$ unlabeled training patterns $X_i$ drawn from some pattern space $\mathcal{Z}$. The basic task of *central clustering* is to find $k$ *cluster centers* or *models*

$$\mathcal{Y} = \{Y_1, \ldots, Y_k\} \subseteq \mathcal{Z}$$

such that a cost function also known as *average distortion*

$$E(M, \mathcal{Y}; \mathcal{X}) = \frac{1}{p} \sum_{j=1}^{k} \sum_{i=1}^{p} m_{ij} D(X_i, Y_j) \tag{8.2}$$

is minimized with respect to a given *distortion measure* $D$. The average distortion $E(M, \mathcal{Y}; \mathcal{X})$ is a function of the variables $M$ and $\mathcal{Y}$ given the set $\mathcal{X}$ of training patterns. The size $k$ of the cluster set $\mathcal{Y}$ has to be determined a priori or by a problem dependent complexity measure [44]. The matrix $\boldsymbol{M} = (m_{ij})$ is a binary *membership matrix* from $\mathbb{B}^{p \times k}$ with the constraints

$$\sum_{j=1}^{p} m_{ij} = 1 \tag{8.3}$$

for all $i = [1 : p]$. The constraints assure that each data point is represented by a unique model. Fuzzy clustering or topology preserving clustering methods like self-organizing feature maps relax the hard constraints on the membership matrix $M$ and demand entries $m_{ij}$ in the range of $[0, 1]$ subject to (8.3).

An appropriate distortion measure $D$ depends on the particular application domain. For feature vectors, the most common choice of $D$ is the quadratic cost function

$$D(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|^2$$

with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$.

Finding a minimum of the average distortion (8.2) is effectively a search of optimal assignments $m_{ij}$ in the discrete space of membership matrices with exponentially many states. Various strategies have been proposed to optimize (8.2). These can be generally divided into two categories: *hard optimizers* that only adjust the model closest to a given input, and *soft optimizers* that adjust a set of models for a given input. Examples of hard optimizers are the *K-means* algorithm [168] and *simple competitive learning* [166]. Soft optimizers include, for example, self-organizing feature maps [213] or probabilistic partitioning algorithms [45].

## 8.3 Structural Learning

The goal of this section is to motivate structural neural learning machines. To this end, we briefly review some common approaches in structural learning and discuss their limitations.

Structural learning refers to learning on data represented in terms of graphs. Learning on graphs is difficult for two main reasons:

1. There is no natural labeling of the items in a graph. To compare graphs, correspondences between the items of both graphs must be established.
2. There is structural variation in the domain of graphs in the sense that for different graphs, the number of vertices and edges may vary.

As a consequence, it is unclear how to provide a well-defined formulation of simple statistical concepts for data analysis like the mean or variance of a given set of graphs. This shortcoming motivates research on combining structural pattern recognition with statistical methods [47, 195]. Among the few approaches that bridge the gap between structural and statistical pattern recognition, we discuss (1) pairwise proximity-based methods, (2) kernel methods, (3) pointwise embedding into vector spaces, and (4) recurrent neural networks.

### Proximity-Based Methods

Given a finite set of objects, proximity-based methods represent each data point by a set of proximities to the other data points of that set. This approach is often motivated by the assumption that proximity is more important than a feature or a class, because it is the proximity measure that serves as a crucial factor in the process of human recognition and categorization [79, 123, 277]. For an overview on learning algorithms designed for pairwise proximity data, we refer to [277] for classification problems and [145] for clustering problems.

*Problems:* Successful application of proximity-based methods highly depends on the definition of an appropriate proximity measure. The choice of a proximity measure, however, is problem dependent and often requires expert domain knowledge. Proximity-based methods applied to attributed graphs face the additional problem that the characteristics of structural proximity measures are not well investigated. The main reason for this shortcoming is that the choice of a proximity measure implicitly presupposes knowledge about the distribution of the data. Since it is unclear how to transfer elementary statistical concepts like the mean or variance to the domain of attributed graphs, we may find it intricate to formulate suitable probabilistic models for attributed graphs for solving problems in structural pattern recognition.

Another disadvantage is that calculating the proximities between all pairs of

data points is computationally inefficient, requiring a complexity of at least $O(p^2)$, where $p$ is the number of data points. For structural proximity measures, we are faced with the problem of (approximately) solving $O(p^2)$ graph matching problems, which may be intractable in a practical setting.

A third problem with proximity-based methods is the *reconstruction problem*. The reconstruction problem calls for the data in the original space given a point in the proximity space. It is unclear to what extent reconstruction of original data by given points in the proximity space is distorted.

Finally, we note that proximity-based methods usually require knowledge of the training sample as a whole and are therefore not intended for incremental learning problems.

#### Pointwise Embedding into Vector Spaces

The goal of pointwise embeddings is to transform graphs to vectors in an Euclidean space, where we have a plethora of powerful techniques for intelligent data analysis at our disposal. Multidimensional scaling [77] and related tools constitute popular unsupervised techniques for data visualization. These methods aim at projecting given data points to low dimensions, whereby pairwise proximities are preserved as far as possible. Another approach is to directly extract feature vectors from the graphs. Popular approaches rely on graph-spectral decomposition methods [244]. Features of weighted graphs are extracted from the eigenvalues and eigenvectors of graph Laplacians. These features can be used to represent the structure of a graph and are easy to compute.

*Problems:* Approaches that use pairwise proximities have the same shortcomings as pairwise proximity-based methods. Challenging problems of spectral decomposition are (i) the reconstruction problem, and (ii) noise sensitivity.

#### Kernel Methods

Training and classification of the support vector machine (SVM) can be formulated in terms of $k(\boldsymbol{x}, \boldsymbol{x}_i)$, $\boldsymbol{x}_i$ being a support vector and $k$ the kernel. Thus, a modification of the kernel to more complex data allows transfer of the SVM to more complex domains. In such, the resulting classification model can be interpreted as a linear classifier in a high dimensional feature space if, and only if, the kernel decomposes into $k(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x}) \cdot \Phi(\boldsymbol{y})$. This is valid for positive definite kernels. Analogously, other kernel based methods such as kernel principal component analysis rely solely on an appropriate choice of $k$, and the task thus reduces to the design of kernels for structured data.

Recently, various methods of extending kernels to nonstandard data have been proposed. An overview of kernels for structures is presented in [106]. For kernels based on common substructures, [137, 380] introduced the basic principle of composite kernels. Simple kernels defined on subparts of given structures can be extended

by generic operations to more complex, convolutional kernels. In particular, strong closure properties for positive definite kernels hold, allowing easy construction of problem specific versions.

*Problems:* As a proximity-based method, the problems of pairwise proximity data transfer to kernel methods.

In addition, evaluating a complete structural kernel that captures the whole structure of the graphs under consideration is computationally inefficient. Otherwise, we would have an efficient method to solve the graph isomorphism problem, which is not suspected to be in class P. Standard approaches in structural pattern recognition solve the complexity problem by applying approximate methods. This does not work for support vector learning, because approximation of a complete structural kernel may result in an indefinite Gram matrix. A second problem is the number of support vectors after learning. Although we may use approximate solutions to predict the outcome of unseen data, approximation of hard problems for even a moderate number of support vectors is impracticable.

This motivated the search for alternative graph kernels, which are less expensive to compute [105, 201, 202]. Such kernels are problem dependent and may work fine for selected problems. But in a general setting, loss of structural information may be uncontrollable.

**Recursive Neural Networks**

Recursive neural networks decompose the structures into constituents and recursively process the basic parts. In doing so, the data already processed sets a context for further computation in such that the single parts can be integrated to a whole structure.

Simple recursive networks constitute a well-established tool for time series data. Assume $x_t$ denotes the sequence entry at time point $t$. Then the dynamic is given by the equation $c_t = f(x_t, c_{t-1})$ whereby $f$ is some function computed by the network, and $c_t$ denotes the network state at time point $t$. A more detailed overview of recursive network models can be found in [219]. This dynamic can immediately be generalized to more complex recursive structures. Recursive networks as presented in [97, 125, 133, 340] process tree structures as inputs. Given a binary tree $T$ with root label $x$ and subtrees $L$ and $R$, the state of the network $c_T$ after processing $T$ is defined as $c_T = f(x, c_L, c_R)$.

Recursive models have been thoroughly investigated mainly for supervised learning. An overview of various aspects including learnability, dynamical properties, and training algorithms can be found in [135]. An increasing interest in unsupervised recursive processing of structured data can be observed, where we refer to [12, 18] for overviews of this topic.

*Problems:* According to the recursive paradigm, RNNs can only process linear and quasi-linear structures like strings (sequences), trees, and directed positional acyclic

graphs.[2] Obviously, directed positional acyclic graphs constitute a rather restricted class of graphs for problems in structural pattern recognition. The usual strategy to overcome this limitation is to propose heuristics, which transform more general classes of graphs to recursive structures, which in turn can be processed by an RNN. As for pairwise proximity-based approaches and embeddings into vector spaces, transformation of highly structured data into a space of less structured data may occur at the expense of structural information.

### Structural Neural Learning Machines

Although the long term vision of combining structural and statistical pattern recognition has been formulated, only little progress has been made to achieve powerful tools for intelligent data analysis, in particular for methods that directly operate in the domain of attributed graphs. All approaches discussed in the previous section aim at leaving the unexploited domain of attributed graphs in favor of less expressive domains, for which more powerful analytical techniques are available. Recursive neural networks differ from the other approaches in such that they operate on a subdomain of the domain of attributed graphs.

Our goal is to construct learning machines that directly operate on attributed graphs. Since we are also interested in integrating the symbolic and sub-symbolic paradigm, Part II aims at constructing structural neural learning machines that

- can adaptively process arbitrary attributed graphs,
- capture structural information of the data,
- are robust against noise in the attributes and structural errors,
- are computationally more efficient as structural proximity-based methods, and
- can be performed in batch and incremental mode.

What we want is a general framework for classification, function approximation, and clustering problems without restriction to certain classes of graphs. Also, most of the disadvantages of other approaches should be avoided including, for example, the reconstruction problem and the problem of uncontrolled loss of structural information.

The starting point of the second part of this work is the clustering method for weighted graphs proposed by Gold et al. [118]. Their work focus on minimizing a clustering objective from an algorithmic point of view. Here, we present a principled and generic machinery that can be used to construct structural neural learning machines, and is independent of the particular optimization algorithm for minimizing an error criterion.

2. A positional graph is a graph on which each vertex has a prespecified order.

# 9 The Structural Dot Product

This chapter proposes the fundamental concept of the *structural dot product* for attributed graphs. Based on this concept, we develop a theory on the geometry of structures. We show that the structural dot product is a structural similarity measure with the same geometrical properties as the conventional dot product for feature vectors. By introducing the structural dot product, we lay the ground for differential analysis of functions defined on graphs.

## 9.1 Introduction

To determine the activation of a conventional unit, we evaluate the dot product of a vector of input signals and a weight vector. Since the dot product is linear in one variable, we can exploit its gradient information to formulate learning rules for minimizing error functions of neural learning machines.

To construct neural learning machines for graphs, the key idea is to replace the weight vector associated with conventional units by a weight graph, and the dot product by a suitable structural similarity measure. Both modifications are simple and their practical realization is of technical nature. What remains is the problem of formulating iterative methods for optimizing functions defined on attributed graphs. Descent techniques that use local information of the objective to be optimized are inapplicable, because a meaningful concept of a derivative for functions on graphs is apparently unknown [195].

The aim of this section is to establish a theoretical basis upon which we can formulate concepts of a derivative for functions on graphs. The key concept is the structural dot product for attributed graphs, which extends the dot product concept for feature vectors. Extension of the dot product is motivated by the assumption that an extended version shares many properties with its original concept and therefore simplifies construction and analysis of structural neural learning machines. Geometrical results derived from the structural dot product confirm our assumption to some extent. In particular, we can show that the structural dot product is a dot product in a geometrical sense. Most importantly, as we will see in the next chapter, the structural dot product enables us to construct spaces for which concepts of a derivative of functions on those spaces can be defined. This completely justifies our approach.

To commence, we present a geometrical view of complex attributed graphs in the next section. This view motivates the structural dot product and another theory on the geometry of structures. What makes the whole development of our theory complicated is that the structural dot product is a dot product, but not an inner product. Therefore, Section 9.3 is intended to accentuate the difference of a dot product and an inner product. In Section 9.4, we extend the dot product of attributed matrices, where the attributes are elements of some inner product space. Section 9.5 proposes the *structural dot product* of two attributed graphs as a maximizer of their adjacency matrices over all possible labelings of the vertices. In Section 9.6, we develop a theory of the geometry of structures. The results constitute one important part of our mathematical toolkit for analyzing functions on graphs. We conclude with a summary in Section 9.7.

## 9.2 Vector Representations of Attributed Graphs

The aim of this section is to present a geometrical view of attributed graphs. The geometrical point of view motivates the whole theory developed in this and subsequent chapters. Moreover, it is aimed at introducing the common terminology for this part.

Let $\mathcal{V}$ be an inner product space over a field $K$ with null element $\mathbf{0} \in \mathcal{V}$. Suppose that $\mathcal{V}$ is equipped with an inner product $\langle ., . \rangle$. For convenience, we restrict ourselves to the Euclidean space $\mathcal{V} = \mathbb{R}^d$ with the usual scalar product, but point out that it is possible to extend the whole theory to the more general case of arbitrary inner product $K$-vector spaces $\mathcal{V}$.

In what follows, the set $\mathcal{A}$ of attributes is a subset from $\mathbb{R}^d$. Unless otherwise stated, the null element $\mathbf{0}_d \in \mathbb{R}^d$ is contained in $\mathcal{A}$.

### *Complex Attributed Graphs of Bounded Order*

Let $\mathbb{G}_{\mathcal{A}}^n$ be the set of complex attributed graphs of order $n$ with the null element $\mathbf{0}_d \in \mathcal{A}$ as null attribute $\epsilon$. Note that complex graphs are unrelated to complex numbers. The term *complex* refers to graphs that are composed of a proper graph in a graph-theoretical sense and fictitious (imaginary) vertices and edges. For a definition of complex graphs we refer to Section 2.2.7.

Graphs of order less than $n$ can be aligned to graphs of order $n$ by adding isolated vertices tagged with the null attribute $\epsilon = \mathbf{0}_d$. Suppose that $X$ is a complex attributed graph of order $m < n$. To align $X$, we insert $p = n - m$ vertices such that the adjacency matrix $\boldsymbol{X'}$ of the aligned graph $X'$ is of the form

$$\boldsymbol{X'} = \begin{pmatrix} \boldsymbol{X} & \mathbf{0}_{m,p} \\ \mathbf{0}_{p,m} & \mathbf{0}_{p,p} \end{pmatrix},$$

where $\mathbf{0}_{m,p}$, $\mathbf{0}_{p,m}$, and $\mathbf{0}_{p,p}$ are padding zero matrices. Note that each entry of a zero matrix $\mathbf{0}_{a,b}$ is a $d$-dimensional zero vector $\mathbf{0}_d \in \mathcal{A}$.

By aligning, we can embed real attributed graphs of order less than $n$ into $\mathbb{G}_{\mathcal{A}}^n$. Since all what matters is that the graphs are finite, we assume that $n$ is sufficiently large. Hence, we may drop the superscript $n$ and simply write $\mathbb{G}_{\mathcal{A}}$ instead of $\mathbb{G}_{\mathcal{A}}^n$.

***Remark 9.1***
*Aligning graphs is a purely technical trick to enable a derivation of strong mathematical results. In a practical implementation of neural learning machines, aligning is done implicitly. This is one important key issue to make structural neural learning machines beneficial for data analysis in the domain of graphs. To clarify this issue, we will occasionally return to this remark.*

### *Labeled and Unlabeled Graphs in Euclidean Spaces*

In the following, we show how attributed graphs can be embedded into a Euclidean space. The geometrical view presented here is useful to get a clear picture of the whole theory developed in this and the following chapters.

Let $X = (V, E, \boldsymbol{X})$ be an attributed graph from $\mathbb{G}_{\mathcal{A}}$. We use a labeling to name the vertices. A *labeling* is a bijection $\nu : V \to [1:n]$ that assigns each vertex $v \in V$ a unique number $\nu(v)$. We use the numbers as identifiers for the vertices to deal with them conveniently.

We call the set $\mathbb{G}_{\mathcal{A}}$ the *space of labeled graphs*. In this space, we consider two graphs as equal whenever their adjacency matrices are equal. Thus, the space may distinguish between isomorphic graphs that arise from one another by relabeling the vertices. A *null graph* 0 of $\mathbb{G}_{\mathcal{A}}$ is a graph, in which each item is assigned the null attribute $\epsilon$. Since the adjacency matrix of a null graph is invariant under relabeling of its vertices, we may speak about *the* uniquely determined null graph of $\mathbb{G}_{\mathcal{A}}$.

Let $\simeq$ be the isomorphism relation defined on $\mathbb{G}_{\mathcal{A}}$. The quotient set

$$\big[\mathbb{G}_{\mathcal{A}}\big] = \mathbb{G}_{\mathcal{A}} / \simeq,$$

of isomorphism classes is called the *space of unlabeled graphs*. The space of labeled and unlabeled graphs is related by the mapping

$$[.] : \mathbb{G}_{\mathcal{A}} \to \big[\mathbb{G}_{\mathcal{A}}\big], \quad X \mapsto [X],$$

which transforms a labeled graph to an unlabeled graph. We call the mapping [.] *unlabeling operation*.

We may view the space $\mathbb{G}_{\mathcal{A}}$ as the set of all attributed matrices from $\mathcal{A}^{n\times n}$. This allows us to geometrically represent an unlabeled graph as a set of vectors in a Euclidean space. To this end, we consider a representative $X = (V, E, \boldsymbol{X})$ of an unlabeled graph $[X]$. Then the equivalence class $[X]$ is completely described by

$$[X] = \big\{X^\pi \,:\, \pi \in \mathcal{S}_n\big\},$$

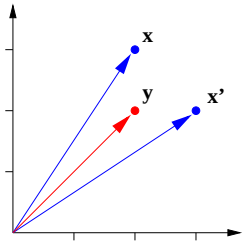



**Figure 9.1** Vector representations of the unlabeled graphs $[X]$ and $[Y]$. Blue refers to vectors from $\mathcal{V}_{[X]}$ and red to the vector from $\mathcal{V}_{[Y]}$.

where $\mathcal{S}_n$ is the set of all permutations acting on $V$, and $X^\pi$ denotes the graph obtained by applying $\pi$ on the vertices of $X$. In terms of attributed adjacency matrices, we can equivalently write

$$[\boldsymbol{X}] = \left\{\boldsymbol{X_P} \,:\, \boldsymbol{X_P} = \boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P},\, \boldsymbol{P} \in \Pi^n\right\},$$

where $\Pi^n$ is the set of all $(n \times n)$-permutation matrices. The vector representation $\boldsymbol{a} = \mathrm{vec}(\boldsymbol{A})$ of an attributed matrix $\boldsymbol{A} = \boldsymbol{a}_{ij}$ is obtained by concatenating its columns. Thus, $\boldsymbol{a}$ is of the form

$$\boldsymbol{a} = (\underbrace{\boldsymbol{a}_{11}, \ldots, \boldsymbol{a}_{n1}}_{1^{st}\text{ column}}, \underbrace{\boldsymbol{a}_{12}, \ldots, \boldsymbol{a}_{n2}}_{2^{nd}\text{ column}}, \ldots, \underbrace{\boldsymbol{a}_{1n}, \ldots, \boldsymbol{a}_{nn}}_{n^{th}\text{ column}})^\mathsf{T}.$$

Note that $\boldsymbol{a}$ is a $d \cdot n^2$-dimensional vector, because each attribute $\boldsymbol{a}_{ij}$ is a $d$-dimensional vector. Using the vector representation of the attributed adjacency matrices, we obtain the desired Euclidean representation of the unlabeled graph $[X]$

$$\mathcal{V}_{[X]} = \left\{\boldsymbol{x_P} \,:\, \boldsymbol{x_P} = \mathrm{vec}\left(\boldsymbol{X_P}\right),\, \boldsymbol{P} \in \Pi^n\right\}.$$

In this representation, each vector $\boldsymbol{x_P}$ represents a labeled graph. Note that the cardinality $|\mathcal{V}_{[X]}|$ depends on the number and size of the cells of the automorphism partition of $X$. To illustrate this, we provide the following example.

***Example 9.2***
Suppose that $\mathcal{A} \subseteq \mathbb{R}$.

1. Let $[X]$ be the unlabeled graph consisting of two isolated vertices with weight 2 and 3. Since $[X]$ has two vertices, there are 2! ways to label the vertices. The resulting adjacency matrices are of the form

$$\boldsymbol{X} = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{X'} = \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}.$$

Since $\boldsymbol{X} \neq \boldsymbol{X'}$, the labeled graphs $X$ and $X'$ are unequal in $\mathbb{G}_\mathcal{A}$. The vector

representations of $X$ and $X'$ are of the form

$$\boldsymbol{x} = \mathrm{vec}(\boldsymbol{X}) = (2,0,0,3)^\mathsf{T}$$
$$\boldsymbol{x'} = \mathrm{vec}(\boldsymbol{X'}) = (3,0,0,2)^\mathsf{T}$$

giving

$$\mathcal{V}_{[X]} = \{\boldsymbol{x}, \boldsymbol{x'}\}.$$

2. Consider a weighted graph $Y$ with two isolated vertices with equal weight 1. Then the adjacency matrix

$$\boldsymbol{Y} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

of $Y$ is invariant under any permutation of both vertices. Hence, the set $\mathcal{V}_{[Y]}$ consists of a singleton $\boldsymbol{y} = \mathrm{vec}(\boldsymbol{Y}) = (1,0,0,1)^\mathsf{T}$.

Thus, $[X]$ and $[Y]$ are unlabeled graphs of equal order but with different numbers of labeled representatives. All the vectors of $\mathcal{V}_{[X]}$ and $\mathcal{V}_{[Y]}$ lie in the plane spanned by the unit vectors $\boldsymbol{u_1} = (1,0,0,0)^\mathsf{T}$ and $\boldsymbol{u}_3 = (0,0,1,0)^\mathsf{T}$. Thus, we are able to illustrate the vector representations of $[X]$ and $[Y]$. Figure 9.1 shows their position in the plane.

## 9.3 Dot Products and Inner Products

This section serves to accentuate the difference of a *dot product* and an *inner product*. The dot product describes the geometric relationship of two vectors, whereas the inner product is a purely algebraic concept, which generalizes the dot product in arbitrary vector spaces. What we want is a concept for graphs similar to an inner product of vectors. In non-vector spaces without well-defined algebraic operations like the domain of attributed graphs, it is unclear how to formulate an inner product. An alternative approach is using the geometrical definition of a dot product, which is independent of algebraic operations.

The *dot product* of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined by

$$\boldsymbol{x} \cdot \boldsymbol{y} = \|\boldsymbol{x}\| \, \|\boldsymbol{y}\| \cos\theta,$$

where $\theta$ is the angle between the vectors and $\|.\|$ is the Euclidean norm. It follows that $\boldsymbol{x} \cdot \boldsymbol{y} = 0$ if $\boldsymbol{x}$ is perpendicular to $\boldsymbol{y}$. Provided that $\boldsymbol{y}$ is normalized, the dot product therefore has the geometric interpretation as the length of the orthogonal projection of $\boldsymbol{x}$ onto the line $\mathbb{R}\boldsymbol{y}$.

A trigonometric calculation yields

$$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}^\mathsf{T}\boldsymbol{y},$$

showing that the dot product is the usual *scalar product*. The characteristic properties of a scalar product lead to the definition of an inner product. An *inner product* $\langle ., . \rangle$ on $\mathbb{R}^n$ is a positive definite symmetric bilinear form. In detail, an inner product satisfies the following properties: let $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^n$ be vectors, and let $w \in \mathbb{R}$ be a constant. Then we have

1. $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$
2. $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = 0$ if, and only if, $\boldsymbol{x} = 0$
3. $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, \boldsymbol{x} \rangle$
4. $\langle \boldsymbol{x} + \boldsymbol{y}, \boldsymbol{z} \rangle = \langle \boldsymbol{x}, \boldsymbol{y} \rangle + \langle \boldsymbol{y}, \boldsymbol{z} \rangle$
5. $\langle w\boldsymbol{x}, \boldsymbol{y} \rangle = w \langle \boldsymbol{x}, \boldsymbol{y} \rangle$

## 9.4 Matrix Dot Products

In this section, we extend the dot product of vectors to attributed matrices and define the matrix dot product space.

Let $\mathcal{A} = \mathbb{R}^d$ be a Euclidean space, and let $\mathcal{A}^{n \times n}$ be the set of attributed $(n \times n)$-matrices. An attributed matrix $\boldsymbol{X} \in \mathcal{A}^{n \times n}$ is then of the form $\boldsymbol{X} = (\boldsymbol{x}_{ij})$ where the entries $\boldsymbol{x}_{ij}$ are $d$-dimensional vectors.

Let $N = n^2$. We may regard $\mathcal{A}^{n \times n}$ as the $N$-fold direct sum of $\mathcal{A}$ by stacking the columns of each matrix to a $N$-dimensional tuple with components from $\mathcal{A}$. Expanding the components, we obtain a $(dN)$-dimensional vector over $\mathbb{R}$. Then, as a direct sum of a vector space, the set $\mathcal{A}^{n \times n}$ is itself a vector space over $\mathbb{R}$ with addition

$$\boldsymbol{X} + \boldsymbol{Y} = (\boldsymbol{x}_{ij} + \boldsymbol{y}_{ij})$$

and scalar multiplication

$$w\boldsymbol{X} = (w\boldsymbol{x}_{ij})$$

for all $\boldsymbol{X} = (\boldsymbol{x}_{ij})$, $\boldsymbol{Y} = (\boldsymbol{y}_{ij}) \in \mathcal{A}^{n \times n}$, and $w \in \mathbb{R}$. In addition, the inner product on $\mathcal{A}$ induces an inner product

$$\boldsymbol{X} \bullet \boldsymbol{Y} = \sum_{i=1}^{n} \sum_{j=1}^{n} \boldsymbol{x}_{ij}^{\mathsf{T}} \boldsymbol{y}_{ij}.$$

on $\mathcal{A}^{n \times n}$, called *matrix dot product*. Thus, together with the matrix dot product $\bullet$, $(\mathcal{A}^{n \times n}, +, \cdot)$ forms an inner product space called *matrix dot product space*.

Finally, we introduce *scalar-matrix multiplication*, a concept combining ordinary scalar and matrix multiplication. This concept enables us, for example, to rearrange rows and columns of an attributed matrix in terms of matrix operations. The *left scalar-matrix multiplication* $\boldsymbol{Z} = \boldsymbol{W}\boldsymbol{X}$ of a real-valued matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ and an attributed matrix $\boldsymbol{X} \in \mathcal{A}^{n \times n}$ is defined by

$$\boldsymbol{z}_{ij} = \sum_{k=1}^{n} w_{ik} \boldsymbol{x}_{kj}$$

for all $i, j \in [1\!:\!n]$. Note that each term $w_{ik}\boldsymbol{x}_{kj}$ is a scalar multiplication of a scalar

$w_{ik}$ with a vector $\boldsymbol{x}_{kj}$. In addition, $\boldsymbol{Z}$ is an attributed matrix. Similarly, the *right scalar-matrix multiplication* $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}$ is of the form

$$\boldsymbol{z}_{ij} = \sum_{k=1}^{n} \boldsymbol{x}_{kj} w_{ik}$$

for all $i, j \in [1:n]$. As for standard matrix multiplication, the left and right scalar-matrix multiplication are not commutative. For convenience, we do not distinguish between scalar-matrix multiplication and the usual matrix multiplication in our notation.

To illustrate the use of the scalar-matrix multiplication, consider an attributed matrix $\boldsymbol{X}$ and a permutation matrix $\boldsymbol{P} \in \Pi^n$. The effect of the right scalar-matrix multiplication $\boldsymbol{X}\boldsymbol{P}$ of $\boldsymbol{X}$ and $\boldsymbol{P}$ is to permute the columns of $\boldsymbol{X}$ in the same way that the columns of the identity matrix $\boldsymbol{I}_n$ were permuted in forming $\boldsymbol{P}$.

## 9.5 Structural Dot Products

The structural dot product is the first of two supporting pillars for construction of neural learning machines for attributed graphs. We introduce the structural dot product and discuss some issues from the perspective of graph matching.

First, we introduce addition and scalar multiplication on the space of labeled graphs $\mathbb{G}_{\mathcal{A}}$ by regarding it as a Euclidean space of matrices. Let $X$ and $Y$ be attributed graphs with adjacency matrix $\boldsymbol{X}$ and $\boldsymbol{Y}$, and let $w \in \mathbb{R}$ be a scalar. Then

$$\begin{aligned} X + Y = Z \text{ with } \boldsymbol{Z} &= \boldsymbol{X} + \boldsymbol{Y} \\ wX = Z \text{ with } \boldsymbol{Z} &= w\boldsymbol{X}. \end{aligned}$$

Addition and scalar multiplication of graphs are defined via the usual componentwise addition and scalar multiplication of their adjacency matrices. Thus, the triple $(\mathbb{G}_{\mathcal{A}}, +, \cdot)$ is a vector space called *structural space*. Clearly, the sum of graphs depends on the particular labeling of their vertices and is therefore meaningless in a graph-theoretical sense.

A *structural dot product* $X \bullet Y$ of $X$ and $Y$ is of the form

$$X \bullet Y = \max_{\boldsymbol{P} \in \Pi^n} \boldsymbol{X}_{\boldsymbol{P}} \bullet \boldsymbol{Y} = \max_{\boldsymbol{P} \in \Pi^n} \boldsymbol{P}^{\mathsf{T}} \boldsymbol{X} \boldsymbol{P} \bullet \boldsymbol{Y}. \tag{9.1}$$

The next result tells us that the definition of a structural dot product is well-defined, i.e. independent of the labeling of the vertices of $Y$.

***Lemma 9.1***
*Let $X$ and $Y$ be attributed graphs with adjacency matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$. Then*

$$\max_{\boldsymbol{P} \in \Pi^n} \boldsymbol{X}_{\boldsymbol{P}} \bullet \boldsymbol{Y} = \max_{\boldsymbol{P}, \boldsymbol{Q} \in \Pi^n} \boldsymbol{X}_{\boldsymbol{P}} \bullet \boldsymbol{Y}_{\boldsymbol{Q}}. \tag{9.2}$$

***Proof*** By definition of the matrix dot product, we have

$$\boldsymbol{X} \bullet \boldsymbol{Y} = \boldsymbol{Q}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{Q} \bullet \boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q}$$

for all $\boldsymbol{Q} \in \Pi^n$. Then

$$\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q} = \boldsymbol{Q}\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P}\boldsymbol{Q}^{\mathsf{T}} \bullet \boldsymbol{Q}\boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q}\boldsymbol{Q}^{\mathsf{T}}$$

for all $\boldsymbol{P}, \boldsymbol{Q} \in \Pi^n$. From $\boldsymbol{Q}\boldsymbol{Q}^{\mathsf{T}} = \boldsymbol{I}_n$ follows

$$\boldsymbol{Q}\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P}\boldsymbol{Q}^{\mathsf{T}} \bullet \boldsymbol{Q}\boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q}\boldsymbol{Q}^{\mathsf{T}} = \boldsymbol{Q}\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P}\boldsymbol{Q}^{\mathsf{T}} \bullet \boldsymbol{Y}.$$

Together with the usual matrix multiplication, permutation matrices form a group. Hence, there exists a permutation matrix $\boldsymbol{R} \in \Pi^n$ with $\boldsymbol{R} = \boldsymbol{P}\boldsymbol{Q}^{\mathsf{T}}$. This shows that for all permutation matrices $\boldsymbol{P}, \boldsymbol{Q} \in \Pi^n$, there is a permutation matrix $\boldsymbol{R} \in \Pi^n$ such that

$$\boldsymbol{R}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{R} \bullet \boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P}\boldsymbol{Q}^{\mathsf{T}} \bullet \boldsymbol{Y}. \tag{9.3}$$

Conversely, for each $\boldsymbol{R} \in \Pi^n$, we have

$$\boldsymbol{R}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{R} \bullet \boldsymbol{Y} = \boldsymbol{R}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{R} \bullet \boldsymbol{I}_n^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{I}_n. \tag{9.4}$$

The assertion follows from (9.3) and (9.4). ■

We provide a geometrical interpretation to get an intuitive idea about the structural dot product of the given graphs $X$ and $Y$. To this end, we consider the vector representations $\mathcal{V}_{[X]}$ and $\mathcal{V}_{[Y]}$ of the unlabeled graphs $[X]$ and $[Y]$. Then we can express the structural dot product in terms of $\mathcal{V}_{[X]}$ and $\mathcal{V}_{[Y]}$ by

$$X \bullet Y = \max_{\boldsymbol{x} \in \mathcal{V}_{[X]}} \boldsymbol{x}^{\mathsf{T}}\boldsymbol{y},$$

where $\boldsymbol{y} \in \mathcal{V}_{[Y]}$ is arbitrarily chosen. Note that $\boldsymbol{x}^{\mathsf{T}}\boldsymbol{y}$ is the inner product derived from the matrix dot product $\boldsymbol{X} \bullet \boldsymbol{Y}$. From Lemma 9.1, it follows that $\boldsymbol{x}^{\mathsf{T}}\boldsymbol{y}$ is independent of the choice of $\boldsymbol{y}$. Since all the vectors of $\mathcal{V}_{[X]}$ have the same length, the structural dot product is the inner product of vectors from $\mathcal{V}_{[X]}$ and $\mathcal{V}_{[Y]}$ with the smallest enclosing angle. This is illustrated by the next example.

***Example 9.3***
Suppose that $\mathcal{A} \subseteq \mathbb{R}$. Consider the labeled graphs $X$ and $Y$ determined by their adjacency matrices

$$\boldsymbol{X} = \begin{pmatrix} 1 & 0 \\ 0 & 5 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{Y} = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}.$$

The vector representations of the unlabeled graphs $[X]$ and $[Y]$ are

$$\begin{aligned} \mathcal{V}_{[X]} &= \left\{\boldsymbol{x} = (1,0,0,5)^{\mathsf{T}}, \boldsymbol{x}' = (5,0,0,1)^{\mathsf{T}}\right\} \\ \mathcal{V}_{[Y]} &= \left\{\boldsymbol{y} = (3,0,0,1)^{\mathsf{T}}, \boldsymbol{y}' = (1,0,0,3)^{\mathsf{T}}\right\}. \end{aligned}$$

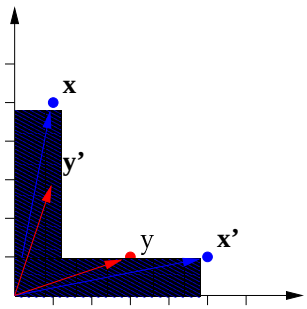



**Figure 9.2** Geometrical interpretation of the structural product $X \bullet Y$. Blue refers to $[X]$ and red to $[Y]$.

Since the structural dot product maximizes problem (9.2), we have

$$\begin{aligned} X \bullet Y &= \max\left\{\boldsymbol{x} \cdot \boldsymbol{y}, \boldsymbol{x} \cdot \boldsymbol{y}', \boldsymbol{x}' \cdot \boldsymbol{y}, \boldsymbol{x}' \cdot \boldsymbol{y}'\right\} \\ &= \max\left\{8, 16, 16, 8\right\} = 16. \end{aligned}$$

Thus, the structural dot product is the inner product

$$\boldsymbol{x} \cdot \boldsymbol{y}' = \boldsymbol{x}' \cdot \boldsymbol{y}.$$

As shown in Figure 9.2, the pairs $(\boldsymbol{x}, \boldsymbol{y}')$ and $(\boldsymbol{x}', \boldsymbol{y})$ are the vectors from $\mathcal{V}_{[X]} \times \mathcal{V}_{[Y]}$ with the smallest angle.

***Definitions:*** The geometrical interpretation prompts some definitions. Let $X$ and $Y$ be attributed graphs from $\mathbb{G}_{\mathcal{A}}$.

1. *Optimal Rotation*: An optimal rotation of $X$ towards $Y$ is a permutation matrix $\boldsymbol{P}$ with[1]

$$\boldsymbol{X} \bullet \boldsymbol{Y} = \boldsymbol{X}_{\boldsymbol{P}} \bullet \boldsymbol{Y}.$$

By $\mathcal{R}(X, Y) \subseteq \Pi^n$, we denote the set of all optimal rotations of $X$ towards $Y$.

2. *Optimal Relabeling*: We call a permutation $\pi \in \mathcal{S}_n$ an optimal relabeling of $X$ to $Y$ if its matrix representation $\boldsymbol{P}_\pi \in \Pi^n$ is an optimal rotation of $X$ towards $Y$. By $\mathcal{R}_{X,Y} \subseteq \mathcal{S}_n$ we denote the set of all permutations that are optimal relabelings of $X$ to $Y$.

3. *Angle*: Given an optimal rotation $\boldsymbol{P} \in \mathcal{R}(X, Y)$, the angle between $X$ and $Y$ is the angle $\theta \in [0, \pi]$ between the vectors $\mathrm{vec}(\boldsymbol{X}_{\boldsymbol{P}})$ and $\mathrm{vec}(\boldsymbol{Y})$. Since

$$\|\boldsymbol{x}\| = \|\boldsymbol{x}'\| \qquad \text{and} \qquad \|\boldsymbol{y}\| = \|\boldsymbol{y}'\|$$

for all $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{V}_{[X]}$ and all $\boldsymbol{y}, \boldsymbol{y}' \in \mathcal{V}_{[Y]}$, the angle of $X$ and $Y$ is well-defined according to Lemma 9.1.

---

1. Note that an optimal rotation $\boldsymbol{P}$ maximizes (9.1), i.e. $\boldsymbol{X}_{\boldsymbol{P}} \bullet \boldsymbol{Y} \geq \boldsymbol{X}_{\boldsymbol{Q}} \bullet \boldsymbol{Y}$ for all $\boldsymbol{Q} \in \Pi^n$. Hence, we may write $\boldsymbol{X} \bullet \boldsymbol{Y} = \boldsymbol{X}_{\boldsymbol{P}} \bullet \boldsymbol{Y}$.

Note that optimal rotations and optimal relabelings are different representations for the same concept. We shall make use of both notions.

### 9.5.1 Complexity of the Structural Dot Product

We show that determination of the structural dot product is NP-complete.

***Proposition 9.1***
*Problem (9.1) is NP-complete.*

***Proof*** Let $X$ and $Y$ be binary graphs of order $|X| = n$ and $|Y| = m$. As usual $\boldsymbol{X} = (x_{ij})$ and $\boldsymbol{Y} = (y_{ij})$ denote the respective adjacency matrices. Then we can rewrite (9.1) by

$$X \bullet Y = \max_{\boldsymbol{P} \in \Pi^{n \times m}} \sum_{r=1}^{n} \sum_{i=1}^{m} \sum_{s=1}^{n} \sum_{j=1}^{m} p_{ri} p_{sj} x_{rs} y_{ij}, \tag{9.5}$$

where $\Pi^{n \times m}$ denotes the set of all $(n \times m)$-permutation submatrices, and $p_{ri}$ and $p_{sj}$ are elements of $\boldsymbol{P}$. Since all elements occuring on the right hand side of (9.5) are either one or zero, maximizing (9.5) is equivalent to maximizing the number of terms $p_{ri} p_{sj} x_{rs} y_{ij}$ for which all four elements are one. This in turn is equivalent to finding the maximum number of matching edges in two graphs. Hence, problem (9.5) is equivalent to the maximum common subgraph problem, which is NP-complete [104]. Since the maximum common subgraph problem is a special case of the structural dot product, problem (9.1) is, in the worst case, also NP-complete. ■

The proof of Proposition 9.1 has two notable implications:

- As shown in (9.5), computing the structural dot product requires no alignment of the smaller graph to the size of the larger one. This implication is important for practical issues, especially because of the intractability of the structural dot product. See also Remark 9.1.
- Let $s(X, Y)$ denote the number of items of a maximum common subgraph of binary graphs $X$ and $Y$. From the proof of Proposition 9.1, it follows that $s(X, Y)$ is equal to the structural dot product $X \bullet Y$. Hence, all properties of the structural dot product also hold for $s(X, Y)$. This implication provides a new geometrical view on the maximum common subgraph, which is illustrated in Section 9.6.

Since problem (9.1) is a graph matching problem, we pose the question as to whether there is an equivalent transformation of the structural dot product to the MWCP in an association graph. Since vertex and edge attributes may have negative values, transformation to an equivalent MWCP is not straightforward as it is for the standard problems listed in Corollary 3.1.

***Proposition 9.2***
*Problem (9.1) is equivalent to the MWCP in an association graph.*

***Proof*** Throughout the proof, we consider the matrix representations of partial morphisms from $X$ to $Y$. Let $X$ and $Y$ be attributed graphs from $\mathbb{G}_{\mathcal{A}}$ of order $|X| = n$ and $|Y| = m$ with adjacency matrices $\boldsymbol{X} = (\boldsymbol{x}_{ij})$ and $\boldsymbol{Y} = (\boldsymbol{y}_{ij})$. Similarly, as in the proof of Proposition 9.1, we may rewrite (9.1) by

$$X \bullet Y = \max_{\boldsymbol{P} \in \Pi^{n \times m}} \sum_{r=1}^{n} \sum_{i=1}^{m} \sum_{s=1}^{n} \sum_{j=1}^{m} p_{ri} p_{sj} \boldsymbol{x}_{rs}^{\mathsf{T}} \boldsymbol{y}_{ij}.$$

Let $L : \mathbb{R} \to \mathbb{R}$ be a linear function with $0 < L(\boldsymbol{x}_{rs}^{\mathsf{T}} \boldsymbol{y}_{ij}) \leq 1$ for all $r, s, i, j$. It is easy to show that such a linear function exists. Clearly, a maximizer $\boldsymbol{P}$ for problem (9.1) is also a maximizer for the matching objective

$$f(\boldsymbol{P}, X, Y) = \sum_{r=1}^{n} \sum_{i=1}^{m} \sum_{s=1}^{n} \sum_{j=1}^{m} p_{ri} p_{sj} L\left(\boldsymbol{x}_{rs}^{\mathsf{T}} \boldsymbol{y}_{ij}\right) \tag{9.6}$$

over the feasible region $\Pi^{n \times m}$ of all $(n \times m)$-permutation sub-matrices. This holds only because the feasible region of problem (9.1) consists of total monomorphisms and contains no partial monomorphisms. Since $L$ is a transformation of all terms $\boldsymbol{x}_{rs}^{\mathsf{T}} \boldsymbol{y}_{ij}$ to the interval $]0, 1]$, we may relax the feasible region to the set $\mathcal{M}$ of all partial monomorphisms from $X$ to $Y$ without effecting maximization of (9.6). According to Proposition 3.2, the feasible region $\mathcal{M}$ is $\mathfrak{p}$-closed. Hence, by Theorem 3.1, problem (9.6) constrained over $\mathcal{M}$ is equivalent to the MWCP in an association graph $X \diamond Y$. This implies that the maximum weight cliques of $X \diamond Y$ are in one-to-one correspondence with the optimal solutions of problem (9.1). ■

## 9.6 Geometry of Structures

This section derives fundamental geometrical ideas from the notion of the structural dot product. Besides other geometrical properties, we show that the structural dot product is a conventional dot product in the sense that

$$X \bullet Y = \|X\| \|Y\| \cos\theta,$$

where $\theta$ is a well-defined angle between $X$ and $Y$.

In contrast to standard dot products, a structural dot product is not bilinear and therefore not an inner product. This shortcoming makes construction of tools for data analysis complicated. But we can show that structural dot products share some useful properties with inner products. We begin our analysis by providing some basic properties of the structural dot product.

### *Algebraic Properties*

***Proposition 9.3***
*Let $X$, $Y$, $Z \in \mathbb{G}_{\mathcal{A}}$ be attributed graphs with adjacency matrix $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$, respectively. Then*

1. $X \bullet X = \boldsymbol{X} \bullet \boldsymbol{X} \geq 0$ (*positive definite, part* 1)
2. $X \bullet X = 0 \;\Leftrightarrow\; \boldsymbol{X} = \boldsymbol{0}_{n,n}$ (*positive definite, part* 2)
3. $X \bullet Y = Y \bullet X$ (*symmetric*)
4. $wX \bullet Y = w(X \bullet Y)$ *for all* $w \in \mathbb{R}_+^0$ (*positive homogeneous*)
5. $(X + Y) \bullet Z \leq X \bullet Y + Y \bullet Z$ (*sublinear*)
6. $X \simeq Y \;\Rightarrow\; X \bullet X = Y \bullet X$ (*invariant under relabeling*)

***Proof***

1. Let $\boldsymbol{P}$ be a an optimal rotation of $X$ towards $Y$. Since the matrix dot product is an inner product, we have

$$X \bullet X = \boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{X} = \left\|\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P}\right\| \left\|\boldsymbol{X}\right\| \cos\alpha,$$

where $\alpha$ is the angle between $\operatorname{vec}\left(\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P}\right)$ and $\operatorname{vec}\left(\boldsymbol{X}\right)$. From

$$\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} = \boldsymbol{X} \bullet \boldsymbol{X}$$

follows

$$\left\|\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P}\right\| = \left\|\boldsymbol{X}\right\|.$$

Hence,

$$X \bullet X = \left\|\boldsymbol{X}\right\|^2 \cos\alpha \leq \boldsymbol{X} \bullet \boldsymbol{X}.$$

A structural dot product maximizes the matrix dot product of the adjacency matrices over all rotations. Therefore we have

$$X \bullet X \geq \boldsymbol{I}_n^{\mathsf{T}}\boldsymbol{X}\boldsymbol{I}_n \bullet \boldsymbol{X} = \boldsymbol{X} \bullet \boldsymbol{X}.$$

Combining the last two inequalities yields

$$X \bullet X = \boldsymbol{X} \bullet \boldsymbol{X} \geq 0,$$

where the positiveness of the structural dot product follows from the positiveness of the matrix dot product.

2. Follows directly from the positive definiteness of the matrix dot product.
3. Let $\boldsymbol{P}$ be an optimal rotation of $X$ towards $Y$. Since the matrix dot product

is symmetric, we have

$$\begin{aligned} X \bullet Y &= \boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{Y} \\ &= \boldsymbol{Y} \bullet \boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} \\ &= \boldsymbol{Q}\boldsymbol{Y}\boldsymbol{Q}^\mathsf{T} \bullet \boldsymbol{Q}\boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P}\boldsymbol{Q}^\mathsf{T} \end{aligned}$$

for all permutation matrices $\boldsymbol{Q} \in \Pi^n$. Applying Lemma 9.1 yields $X \bullet Y \leq Y \bullet X$. With the same argumentation, we obtain $Y \bullet X \leq X \bullet Y$. Finally, combining both inequalities proves the third property.

4. Since $w \geq 0$, we have

$$\begin{aligned} wX \bullet Y &= \max_{\boldsymbol{P} \in \Pi^n} \boldsymbol{P}^\mathsf{T}\big(w\boldsymbol{X}\big)\boldsymbol{P} \bullet \boldsymbol{Y} \\ &= \max_{\boldsymbol{P} \in \Pi^n} w\left(\boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{Y}\right) \\ &= w\big(X \bullet Y\big). \end{aligned}$$

5. Let $\boldsymbol{P}$ be an optimal rotation of $X + Y$ towards $\boldsymbol{Z}$. Then

$$\begin{aligned} \big(X + Y\big) \bullet Z &= \boldsymbol{P}^\mathsf{T}\big(\boldsymbol{X} + \boldsymbol{Y}\big)\boldsymbol{P} \bullet \boldsymbol{Z} \\ &= \big(\boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} + \boldsymbol{P}^\mathsf{T}\boldsymbol{Y}\boldsymbol{P}\big) \bullet \boldsymbol{Z} \\ &= \boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{Z} + \boldsymbol{P}^\mathsf{T}\boldsymbol{Y}\boldsymbol{P} \bullet \boldsymbol{Z} \\ &\leq \max_{\boldsymbol{Q} \in \Pi^n} \boldsymbol{Q}^\mathsf{T}\boldsymbol{X}\boldsymbol{Q} \bullet \boldsymbol{Z} + \max_{\boldsymbol{R} \in \Pi^n} \boldsymbol{R}^\mathsf{T}\boldsymbol{Y}\boldsymbol{R} \bullet \boldsymbol{Z} \\ &= X \bullet Z + Y \bullet Z. \end{aligned}$$

6. Let $X \simeq Y$. Then there are permutation matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$ with $\boldsymbol{Y} = \boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P}$ and $\boldsymbol{X} = \boldsymbol{Q}^\mathsf{T}\boldsymbol{Y}\boldsymbol{Q}$. The assertion follows from combining the two inequalities

$$\begin{aligned} X \bullet X &= \max_{\boldsymbol{R} \in \Pi^n} \boldsymbol{R}^\mathsf{T}\boldsymbol{X}\boldsymbol{R} \bullet \boldsymbol{X} \geq \boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{X} = \boldsymbol{Y} \bullet \boldsymbol{X} = Y \bullet X \\ Y \bullet X &= \max_{\boldsymbol{R} \in \Pi^n} \boldsymbol{R}^\mathsf{T}\boldsymbol{Y}\boldsymbol{R} \bullet \boldsymbol{X} \geq \boldsymbol{Q}^\mathsf{T}\boldsymbol{Y}\boldsymbol{Q} \bullet \boldsymbol{X} = \boldsymbol{X} \bullet \boldsymbol{X} = X \bullet X. \end{aligned}$$

■

### *The Generalized Frobenius Norm for Structures*

A *norm* on $\mathbb{G}_\mathcal{A}$ is a function

$$\|.\| : \mathbb{G}_\mathcal{A} \to \mathbb{R}, \quad X \mapsto \|X\|$$

with the following properties for all $X, Y \in \mathbb{G}_\mathcal{A}$ and all $w \in \mathbb{R}$:

1. $\|X\| = 0 \Leftrightarrow X = 0$
2. $\|wX\| = |w|\|X\|$
3. $\|X + Y\| \leq \|X\| + \|Y\|$.

Although the structural product is not an inner product, it induces a norm in the usual way.

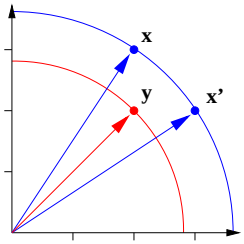



**Figure 9.3** Graphs as points on a ball $\mathcal{B}_{[X]}$. Shown are the vector representations $\mathcal{V}_{[X]}$ and $\mathcal{V}_{[Y]}$ from the unlabeled graphs $[X]$ and $[Y]$ of Example 9.2.

***Theorem 9.1***
*The function*

$$\|.\| : \mathbb{G}_{\mathcal{A}} \to \mathbb{R}, \quad X \mapsto \sqrt{X \bullet X}$$

*is a* norm.

***Proof*** Consider the vector space $\mathcal{A}^N$ with $N = n^2$. Then from elementary linear algebra, it follows that

$$\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^\top \boldsymbol{x}}$$

is a norm on $\mathcal{A}^N$. Now let $X$ be an attributed graph of order $n$ with adjacency matrix $\boldsymbol{X}$. Applying the vec-operator on $\boldsymbol{X}$ yields a vector $\boldsymbol{x} = \mathrm{vec}(\boldsymbol{X}) \in \mathcal{A}^N$. From

$$\sqrt{X \bullet X} = \sqrt{\boldsymbol{X} \bullet \boldsymbol{X}} = \sqrt{\boldsymbol{x}^\top \boldsymbol{x}}$$

follows the assertion. ■

We call the norm $\|.\|$ defined in Theorem 9.1 a *generalized Frobenius norm*. It generalizes the Frobenius norm

$$\|\boldsymbol{A}\|^2 = \mathrm{tr}\left(\boldsymbol{A}^\top \boldsymbol{A}\right)$$

of matrices with real-valued entries.

Since the norm of a graph $X$ is independent of its labeling, we have the following geometric interpretation: the unlabeled graph $[X]$ is a set of points on a ball $\mathcal{B}_{[X]}$ with center $\mathbf{0} \in \mathcal{A}^N$ and radius $\|X\|$. The points are given by the vector representation $\mathcal{V}_{[X]}$ of $[X]$. Each vector of $\mathcal{V}_{[X]}$ is a *support* of $\mathcal{B}_{[X]}$. Figure 9.3 provides an illustrative example.

A direct implication of Theorem 9.1 is that

$$\|X - Y\| = \sqrt{(X - Y) \bullet (X - Y)} \tag{9.7}$$

is a metric on $\mathbb{G}_{\mathcal{A}}$, called *generalized Frobenius metric*. It is important to note that $\|X - X'\|$ can be nonzero for isomorphic graphs $X$ and $X'$. Hence, the generalized

Frobenius metric depends on the labeling of the graphs and is therefore meaningless in a graph-theoretical sense.

***Remark 9.4***
*It is notable that the structural dot product induces the same norm as the standard dot product. Consequently, the generalized Frobenius metric is actually the Euclidean metric when we regard the vector representation of the labeled graphs.*

***The Structural Frobenius Metric***

The generalized Frobenius norm is not appropriate in some cases, because distinct isomorphic graphs from $\mathbb{G}_{\mathcal{A}}$ have nonzero Frobenius distance. The generalized Frobenius metric is derived from the generalized Frobenius norm in the standard manner. Another way to derive a metric from the generalized Frobenius norm is to take the structure of the given graphs into account. We define the distance measure

$$\delta(X,Y) = \min_{\boldsymbol{P}\in\Pi^n} \|\boldsymbol{X_P} - \boldsymbol{Y}\|$$

for all $X, Y \in \mathbb{G}_{\mathcal{A}}$. The distance $\delta(X,Y)$ minimizes the Frobenius metric over all labelings of the vertices. The next result shows how the distance $\delta$ is related to the structural dot product.

***Lemma 9.2***
*Let $X$ and $Y$ be attributed graphs. Then*

$$\delta(X,Y) = \sqrt{\|X\|^2 - 2\big(X \bullet Y\big) + \|Y\|^2}.$$

***Proof*** We have

$$\begin{aligned}\delta(X,Y)^2 &= \min_{\boldsymbol{P}\in\Pi^n} \big\|\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} - \boldsymbol{Y}\big\|^2 \\ &= \min_{\boldsymbol{P}\in\Pi^n} \Big\{ \big(\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} - \boldsymbol{Y}\big) \bullet \big(\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} - \boldsymbol{Y}\big) \Big\}.\end{aligned}$$

From the bilinearity of the matrix dot product follows

$$\begin{aligned}\delta(X,Y)^2 &= \min_{\boldsymbol{P}\in\Pi^n} \Big\{ \boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} - 2\big(\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{Y}\big) + \boldsymbol{Y} \bullet \boldsymbol{Y} \Big\} \\ &= \min_{\boldsymbol{P}\in\Pi^n} \Big\{ \big\|X\big\|^2 - 2\big(\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{Y}\big) + \big\|\boldsymbol{Y}\big\|^2 \Big\}.\end{aligned}$$

The terms $\|X\|$ and $\|Y\|$ are constant for all permutation matrices. Thus, minimizing the last equation is equivalent to maximizing the term $\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{Y}$. We obtain

$$\begin{aligned}\delta(X,Y)^2 &= \big\|X\big\|^2 - \max_{\boldsymbol{P}\in\Pi^n} 2\big(\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{Y}\big) + \big\|\boldsymbol{Y}\big\|^2 \\ &= \big\|X\big\|^2 - 2\big(X \bullet Y\big) + \big\|Y\big\|^2.\end{aligned}$$

Taking the square root yields the assumption. ■

Theorem 9.2 proves that $\delta$ is indeed a metric. Because of Lemma 9.2, we call the metric $\delta$ *structural Frobenius metric* induced by the structural dot product.

***Theorem 9.2***
*The mapping*

$$\delta : \mathbb{G}_{\mathcal{A}} \times \mathbb{G}_{\mathcal{A}} \to \mathbb{R}, \quad \big(X, Y\big) \mapsto \delta(X, Y)$$

*is a distance metric on* $\mathbb{G}_{\mathcal{A}}$.

***Proof*** Let $X$, $Y$, and $Z$ be attributed graphs.

1. First we show that $\delta(X, Y) = 0$ if, and only if, $X \simeq Y$.

⇒: Let $\delta(X, Y) = 0$. Then there is an optimal rotation $\boldsymbol{P}$ of $X$ towards $Y$ with

$$\big\|\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} - \boldsymbol{Y}\big\| = 0.$$

Thus, we have $\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} = \boldsymbol{Y}$. This proves $X \simeq Y$.

⇐: Let $X \simeq Y$. From Proposition 9.3(6) together with Lemma 9.2 follows

$$\begin{aligned}\delta(X, Y)^2 &= \|X\|^2 - 2\big(X \bullet Y\big) + \|Y\|^2 \\ &= \|X\|^2 - 2\big(X \bullet X\big) + \|X\|^2 \\ &= 0.\end{aligned}$$

2. Next we show that $\delta(X, Y) = \delta(Y, X)$: follows from Proposition 9.3(3) together with Lemma 9.2.

3. Finally, we show that $\delta(X, Z) \leq \delta(X, Y) + \delta(Y, Z)$: let $\boldsymbol{P} \in \mathcal{R}(X, Z)$ and $\boldsymbol{Q} \in \mathcal{R}(Y, Z)$ be optimal rotations. Suppose that $\boldsymbol{R}$ is a permutation matrix such that

$$\delta(X, Y) = \big\|\boldsymbol{R}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{R} - \boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q}\big\|.$$

According to Lemma 9.1, such a permutation matrix $\boldsymbol{R}$ exists. Then we have

$$\begin{aligned}\delta(X, Z) &= \big\|\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} - \boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q} + \boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q} - \boldsymbol{Z}\big\| \\ &\leq \big\|\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} - \boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q}\big\| + \big\|\boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q} - \boldsymbol{Z}\big\| \\ &\leq \big\|\boldsymbol{R}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{R} - \boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q}\big\| + \big\|\boldsymbol{Q}^{\mathsf{T}}\boldsymbol{Y}\boldsymbol{Q} - \boldsymbol{Z}\big\| \\ &= \delta(X, Y) + \delta(Y, Z).\end{aligned}$$

■

The structural Frobenius norm gives rise to the following geometrical interpretation. Suppose that $X$ and $Y$ are graphs with vector representations $\mathcal{V}_{[X]}$ and $\mathcal{V}_{[Y]}$. The vectors of $\mathcal{V}_{[X]}$ are the supports of the ball $\mathcal{B}_{[X]}$. Each support $\boldsymbol{x} \in \mathcal{V}_{[X]}$ defines a *sector* $S_{\boldsymbol{x}}$ with

$$\boldsymbol{z} \in S_{\boldsymbol{x}} \;\Rightarrow\; \boldsymbol{z}^{\mathsf{T}}\boldsymbol{x} \geq \boldsymbol{z}^{\mathsf{T}}\boldsymbol{x}'$$

for all $\boldsymbol{x}' \in \mathcal{V}_{[X]}$. The sector $S_{\boldsymbol{x}}$ consists of all points that are closer to $\boldsymbol{x}$ than

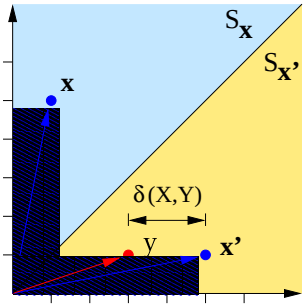



**Figure 9.4** Geometrical interpretation of the structural Frobenius metric of given graphs $X$ and $Y$. The supports $\boldsymbol{x}$ and $\boldsymbol{x}'$ of $[X]$ determine sectors $S_{\boldsymbol{x}}$ and $S_{\boldsymbol{x}'}$. Since the vector representation $\boldsymbol{y}$ of $Y$ lies in sector $S_{\boldsymbol{x}'}$, the distance $\delta(X,Y)$ is equal to the Euclidean distance of $\boldsymbol{x}'$ and $\boldsymbol{y}$.

to any other support from $\mathcal{V}_{[X]}$. Thus, the supports of $\mathcal{B}_{[X]}$ determine a Voronoi tessellation. To compute the structural Frobenius norm $\delta(X,Y)$, we first determine the sector in which the vector representation $\boldsymbol{y}$ of the labeled graph $Y$ falls and then determine the Euclidean distance between the support of that sector and the vector $\boldsymbol{y}$. The resulting distance is independent of the labeling of $Y$.

***Example 9.5***
Consider the labeled graphs $X$ and $Y$ from Example 9.3. The vector representation of the unlabeled graph $[X]$ is

$$\mathcal{V}_{[X]} = \left\{\boldsymbol{x} = (1,0,0,5)^{\mathsf{T}}, \boldsymbol{x}' = (5,0,0,1)^{\mathsf{T}}\right\}.$$

The vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$ determine sectors $S_{\boldsymbol{x}}$ and $S_{\boldsymbol{x}'}$. The vector representation of the labeled graph $Y$ is $\boldsymbol{y} = (3,0,0,1)^{\mathsf{T}}$. The vector $\boldsymbol{y}$ is an element of the sector $S_{\boldsymbol{x}'}$. Hence, the structural Frobenius metric is given by

$$\delta(X,Y) = \left\|\boldsymbol{x}' - \boldsymbol{y}\right\|.$$

Figure 9.4 illustrates this example.

### *The Structural Cauchy-Schwarz Inequality*

The following Theorem extends a famous inequality known as the Cauchy-Schwarz inequality.

***Theorem 9.3 (Structural Cauchy-Schwarz Inequality)***
*Let $X$ and $Y$ be attributed graphs. Then*

$$\left|X \bullet Y\right| \leq \left\|X\right\|\left\|Y\right\|.$$

***Proof*** Let $\boldsymbol{P}$ be an optimal rotation of $X$ towards $Y$. Then we have

$$\left|X \bullet Y\right| = \left|\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P} \bullet \boldsymbol{Y}\right|.$$

From the Cauchy-Schwarz inequality for the matrix dot product follows

$$\big|X \bullet Y\big| \leq \big\|\boldsymbol{P}^{\mathsf{T}}\boldsymbol{X}\boldsymbol{P}\big\|\big\|\boldsymbol{Y}\big\| = \big\|\boldsymbol{X}\big\|\big\|\boldsymbol{Y}\big\|.$$

■

***The Angle of Attributed Graphs***

Using the structural Cauchy-Schwarz inequality, we are in the position to show that the angle of attributed graphs has a geometrical meaning. For two nonzero graphs $X$ and $Y$, the angle $\theta \in [0, \pi]$ between $X$ and $Y$ is defined (indirectly in terms of its cosine) by

$$\cos\theta = \frac{X \bullet Y}{\|X\|\,\|Y\|}. \tag{9.8}$$

The structural Cauchy-Schwarz inequality shown in Theorem 9.3 implies that

$$-1 \leq \frac{X \bullet Y}{\|X\|\,\|Y\|} \leq 1$$

and thus assures that this angle is well-defined. Although a structural dot product $X \bullet Y$ is not an inner product, (9.8) shows that $X \bullet Y$ has the same geometrical properties as the standard dot product.

As an aside, having the concept of an angle for attributed graphs, we are in the position of defining structural orthogonality in the usual way.

***The Structural Minkowski Inequality***

Consider the product space

$$\mathbb{G}_{\mathcal{A}}^{p} = \underbrace{\mathbb{G}_{\mathcal{A}} \times \cdots \times \mathbb{G}_{\mathcal{A}}}_{\text{p times}}.$$

An element of $\mathbb{G}_{\mathcal{A}}$ is a tuple of the form $(X_i)_{i=1}^{p}$ with $X_i \in \mathbb{G}_{\mathcal{A}}$ for all $i \in [1\,{:}\,p]$. A useful tool is the structural Minkowski inequality.

***Theorem 9.4 Structural Minkowski Inequality***
*Let $(X_i)_{i=1}^{p}$ and $(Y_i)_{i=1}^{p}$ be tuples of attributed graphs from $\mathbb{G}_{\mathcal{A}}^{p}$. Then we have*

$$\sqrt{\sum_{i=1}^{p} \big(X_i + Y_i\big) \bullet \big(X_i + Y_i\big)} \leq \sqrt{\sum_{i=1}^{p} X_i \bullet X_i} + \sqrt{\sum_{i=1}^{p} Y_i \bullet Y_i}.$$

***Proof*** Without loss of generality, we may assume that

$$\sum_{i=1}^{p} \big(X_i + Y_i\big) \bullet \big(X_i + Y_i\big) > 0.$$

Since the structural dot product is sublinear, we have

$$\sum_{i=1}^{p} \big(X_i + Y_i\big) \bullet \big(X_i + Y_i\big) \leq \sum_{i=1}^{p} \big(X_i + Y_i\big) \bullet X_i + \sum_{i=1}^{p} \big(X_i + Y_i\big) \bullet Y_i .$$

From the structural Cauchy-Schwarz inequality, it follows that

$$\begin{aligned} \big|\big(X_i + Y_i\big) \bullet X_i\big| &\leq \big\|X_i + Y_i\big\| \big\|X_i\big\| \\ \big|\big(X_i + Y_i\big) \bullet Y_i\big| &\leq \big\|X_i + Y_i\big\| \big\|Y_i\big\| \end{aligned}$$

for all $i \in [1:p]$. This gives the inequality

$$\sum_{i=1}^{p} \big(X_i + Y_i\big) \bullet \big(X_i + Y_i\big) \leq \underbrace{\sum_{i=1}^{p} \big\|X_i + Y_i\big\| \big\|X_i\big\|}_{=A} + \underbrace{\sum_{i=1}^{p} \big\|X_i + Y_i\big\| \big\|Y_i\big\|}_{=B} .$$

The terms $A$ and $B$ at the right hand side of the last inequality can be viewed as standard scalar-product of vectors from $\mathbb{R}^p$. Applying the usual Cauchy-Schwarz inequality yields

$$\begin{aligned} A &\leq \sqrt{\sum_{i=1}^{p} \big\|X_i + Y_i\big\|^2} \sqrt{\sum_{i=1}^{p} \big\|X_i\big\|^2} \\ B &\leq \sqrt{\sum_{i=1}^{p} \big\|X_i + Y_i\big\|^2} \sqrt{\sum_{i=1}^{p} \big\|Y_i\big\|^2} . \end{aligned}$$

We arrive at

$$\begin{aligned} \sum_{i=1}^{p} \big(X_i + Y_i\big) \bullet \big(X_i + Y_i\big) \leq \sqrt{\sum_{i=1}^{p} \big\|X_i + Y_i\big\|^2} \sqrt{\sum_{i=1}^{p} \big\|X_i\big\|^2} \\ + \sqrt{\sum_{i=1}^{p} \big\|X_i + Y_i\big\|^2} \sqrt{\sum_{i=1}^{p} \big\|Y_i\big\|^2} . \end{aligned}$$

Since $\big\|X_i + Y_i\big\|^2 = \big(X_i + Y_i\big) \bullet \big(X_i + Y_i\big)$, the assertion follows from dividing both sides of the last inequality by the term

$$\sqrt{\sum_{i=1}^{p} \big\|X_i + Y_i\big\|^2} .$$

■

### 9.6.1 The Weighted Mean

Next, we define the *weighted mean* of graphs. Let $\boldsymbol{M}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$ be the adjacency matrices of the graphs $M$, $X$, and $Y$, respectively, and let $\eta \in [0,1]$ be a constant. An attributed graph $M$ is a *weighted mean* of $X$ and $Y$ if there is an optimal rotation $\boldsymbol{P}$ from $X$ towards $Y$ such that

$$\boldsymbol{M} = \eta \boldsymbol{X}_{\boldsymbol{P}} + (1-\eta)\boldsymbol{Y}. \tag{9.9}$$

Note that the unlabeled weighted mean $[M]$ is in general not uniquely determined, because $\boldsymbol{X}_{\boldsymbol{P}}$ needs not be uniquely determined. But Theorem 9.5 shows that $M$ is at least a weighted mean of $X$ and $Y$ in a metrical sense.

***Theorem 9.5***
*Let $M$, $X$, and $Y$ be attributed graphs with adjacency matrices $\boldsymbol{M}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$. Let $\boldsymbol{P} \in \mathcal{R}(X,Y)$ be an optimal rotation of $X$ towards $Y$, and let $\eta \in [0,1]$ be a constant. Suppose that there is an optimal rotation $\boldsymbol{P} \in \mathcal{R}(X,Y)$ with*

$$\boldsymbol{M} = \eta \boldsymbol{X}_{\boldsymbol{P}} + (1-\eta)\boldsymbol{Y}.$$

*Then the following equations hold:*

$$\delta(X,M) = (1-\eta)\,\delta(X,Y)$$
$$\delta(M,Y) = \eta\,\delta(X,Y).$$

***Proof*** We only show the first equation. The proof for the second equation is similar. We begin by showing that $\boldsymbol{P}$ is also an optimal rotation of $X$ towards $M$. For this, let $\boldsymbol{Q} \in \Pi^n$. Note that

$$\begin{aligned}\boldsymbol{Q}^\top \boldsymbol{X} \boldsymbol{Q} \bullet \boldsymbol{M} &= \boldsymbol{Q}^\top \boldsymbol{X} \boldsymbol{Q} \bullet \Big(\eta \boldsymbol{P}^\top \boldsymbol{X} \boldsymbol{P} + (1-\eta)\boldsymbol{Y}\Big) \\ &= \eta\Big(\boldsymbol{Q}^\top \boldsymbol{X} \boldsymbol{Q} \bullet \boldsymbol{P}^\top \boldsymbol{X} \boldsymbol{P}\Big) + (1-\eta)\Big(\boldsymbol{Q}^\top \boldsymbol{X} \boldsymbol{Q} \bullet \boldsymbol{Y}\Big).\end{aligned}$$

From the proof of Proposition 9.3(1) follows

$$\boldsymbol{Q}^\top \boldsymbol{X} \boldsymbol{Q} \bullet \boldsymbol{P}^\top \boldsymbol{X} \boldsymbol{P} \leq \boldsymbol{P}^\top \boldsymbol{X} \boldsymbol{P} \bullet \boldsymbol{P}^\top \boldsymbol{X} \boldsymbol{P} = X \bullet X.$$

By assumption, $\boldsymbol{P}$ is an optimal rotation of $X$ towards $Y$ and therefore

$$\boldsymbol{Q}^\top \boldsymbol{X} \boldsymbol{Q} \bullet \boldsymbol{Y} \leq \boldsymbol{P}^\top \boldsymbol{X} \boldsymbol{P} \bullet \boldsymbol{Y} = X \bullet Y.$$

Combining both inequalities finally yields

$$\boldsymbol{Q}^\top \boldsymbol{X} \boldsymbol{Q} \bullet \boldsymbol{M} \leq \boldsymbol{P}^\top \boldsymbol{X} \boldsymbol{P} \bullet \boldsymbol{M}$$

for all $\boldsymbol{Q} \in \Pi^n$. Hence, $\boldsymbol{P}$ is an optimal rotation of $X$ towards $M$. Now we are in

the position to show the first equation. We have

$$\begin{aligned}\delta(X, M) &= \big\|\boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} - \boldsymbol{M}\big\| \\ &= \big\|\boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} - \eta\boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} - (1-\eta)\boldsymbol{Y}\big\| \\ &= (1-\eta)\big\|\boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} - \boldsymbol{Y}\big\| \\ &= (1-\eta)\,\delta(X, Y).\end{aligned}$$

Equality of the second and third line follows from Proposition 9.3(4) using $w = 1 - \eta \geq 0$. ■

## 9.7 Conclusion

This chapter proposed a structural dot product for attributed graphs as an analogon to the standard dot product for vectors. We derived a geometrical view of structures and proved the following key properties of the structural dot product:

1. Determining $X \bullet Y$ is an NP-complete graph matching problem.
2. $X \bullet Y$ can be mapped to an equivalent MWCP in an association graph.
3. $X \bullet Y$ induces a generalized Frobenius norm and metric.
4. $X \bullet Y$ induces a structural Frobenius metric as a minimizer of the generalized Frobenius metric.
5. The structural Cauchy-Schwarz and structural Minkowski inequality holds.
6. We have

   $$X \bullet Y = \big\|X\big\|\big\|Y\big\|\cos\theta,$$

   where $\theta \in [0, \pi]$ is a well-defined angle of $X$ and $Y$.
7. Let $\boldsymbol{P} \in \mathcal{R}(X, Y)$ be an optimal rotation. The graph determined by the adjacency matrix

   $$\eta\boldsymbol{X}_{\boldsymbol{P}} + (1-\eta)\boldsymbol{Y}$$

   is a weighted mean of $X$ and $Y$ in a Euclidean sense.

Since the structural dot product induces a metric, we can introduce concepts from topology and infinitesimal analysis, including open and closed neighborhoods, limits, and continuity. If we can show that each Cauchy sequence of attributed graphs converges, then we are in the position to introduce concepts of a derivative for functions on graphs. The next chapter is devoted to these issues.

# 10 Analysis of Functions on Graphs

In 2001, Jolion [195] published a counter paper discussing some trends and challenges towards the next generation of graph-based representations. In the context of combining structural and statistical pattern recognition, he noted:

> *"What is the meaning of the derivative of a* (function at a) *graph? Maybe this is only nonsense. However, if we can somehow define this concept, we can move from discrete to continuous optimization. What about a Newton-Raphson's algorithm working in a space of graphs? In order to compute or estimate a derivative, one may first be able to compute a difference. So what is the meaning of a difference between two graphs? Not possible?"*

One of the key issues for the success of almost any learning machine in the domain of feature spaces, which is missing in the domain of attributed graphs, are powerful mathematical tools to minimize error functions using local gradient information. Jolion's statement indicates that the concept of a derivative of functions defined on attributed graphs seems to be far off.

This chapter is devoted to answering the questions posed by Jolion. In particular, we show that the gradient of a differentiable real-valued function on attributed graphs is a well-defined attributed graph pointing in the direction of steepest ascent. This result facilitates application of iterative algorithms based on local gradient information to optimize smooth functions on attributed graphs. Besides the structural dot product, differential analysis of structural functions constitutes the second of two supporting pillars for the construction of neural learning machines for attributed graphs.

## 10.1 Introduction

In the previous chapter, we suggested the structural dot product as a combiner of an input graph and weight graph to determine the activation of a unit. Feeding input graphs into a network is half of the story; the other half is formulating learning rules that minimize some error function defined on attributed graphs. Usually, a learning rule uses local gradient information of the error function. Gradient information, in turn, presupposes existence of a derivative. A concept of a derivative for functions of attributed graphs is unknown, as indicated by Jolion [195]. This shortcoming is

part of the more general problem of a lack of mathematical tools for data analysis in the domain of attributed graphs, as several researchers [47, 120, 121, 126, 244] point out.

In this chapter, we show how we can transfer concepts from differential analysis of functions on vector spaces to permutation invariant functions on graph spaces in a meaningful way. The key idea is to work with two metric spaces. The first metric space consists of the set of labeled graphs together with the generalized Frobenius metric. The space of labeled graphs is a Banach space isomorphic to a Euclidean space. In Banach spaces, we have all the analytical concepts at our disposal, including, for example, limits, continuity, and differentiability. But the main problem is that all these concepts appear meaningless in the sense that certain analytical concepts or properties could depend on the particular labeling of the vertices. To keep track of the structural properties, we consult a second metric space, the set of all unlabeled graphs (isomorphism classes) together with the structural Frobenius metric. This space captures full information of the structure of a graph and abstracts from the particular labeling. Since algebraic operations on unlabeled graphs are undefined, it is impossible to define a derivative of functions on unlabeled graphs. Transition from labeled to unlabeled graphs combines the advantages of both metric spaces, namely analytical with structural traceability. As a result, we can show that analytical concepts defined in the space of labeled graphs have a well-defined structural meaning in the space of unlabeled graphs.

This chapter is organized as follows: In Section 10.2, we introduce metric spaces of labeled and unlabeled graphs and the notion of limit. Section 10.3 introduces structural functions on attributed graphs and the notion of continuity. Differentiability is considered in Section 10.4, and optimization of smooth structural functions in Section 10.5. Finally, Section 10.6 concludes with a summary and an outlook on further research.

## 10.2 Metric Spaces of Structures

In this section, we introduce two metric spaces, the space of labeled graphs together with the generalized Frobenius metric, and the space of unlabeled graphs together with the structural Frobenius metric. Metric spaces have enough structure to introduce the notion of limit. We show that the limit of a sequence of labeled graphs is a well-defined structural concept independent of the labeling of the vertices. Using the concept of limit, we can show that the space of labeled graphs is a Banach space and the space of unlabeled graphs is complete.

A *metric space* is a pair $(\mathcal{S}, d)$ consisting of a set $\mathcal{S}$ and a metric $d$ defined on $\mathcal{S}$. The metric $d$ defines a topology on $\mathcal{S}$, where the open balls with center $\boldsymbol{z} \in \mathcal{S}$ and radius $\rho$ are of the form

$$\mathcal{B}(\boldsymbol{z}, \rho) = \{\boldsymbol{x} \in \mathcal{S} \,:\, d(\boldsymbol{z}, \boldsymbol{x}) < \rho\}.$$

The open ball $\mathcal{B}(\boldsymbol{z}, \rho)$ is called $\rho$-*neighborhood* of $\boldsymbol{z}$. By

$$\overline{\mathcal{B}}(\boldsymbol{z}, \rho) = \{\boldsymbol{x} \in \mathcal{S} \,:\, d(\boldsymbol{z}, \boldsymbol{x}) \leq \rho\}.$$

we denote the closed ball with center $\boldsymbol{z}$ and radius $\rho$.

To investigate functions on attributed graphs, we consider two metric spaces coevally:

1. *Labeled Graph Space* $(\mathbb{G}_{\mathcal{A}}, \|.\|)$: Obviously, $\mathbb{G}_{\mathcal{A}}$ together with the generalized Frobenius metric $\|.\|$ is a metric space. Since $\mathbb{G}_{\mathcal{A}}$ is finite dimensional, the resulting metric space is isomorphic to a Euclidean space [205]. We denote the open and closed balls with center $Z \in \mathbb{G}_{\mathcal{A}}$ and radius $\rho$ by $\mathcal{B}_F(Z, \rho)$ and $\overline{\mathcal{B}}_F(Z, \rho)$, respectively. The subscript $F$ in $\mathcal{B}_F(Z, \rho)$ emphasizes that the underlying metric is the generalized Frobenius metric.

2. *Unlabeled Graph Space* $(\big[\mathbb{G}_{\mathcal{A}}\big], \delta)$: According to Lemma 9.1 and 9.2, the structural Frobenius metric $\delta$ is independent of the labeling of its arguments. Hence, $\delta$ is well-defined in $\big[\mathbb{G}_{\mathcal{A}}\big]$. From Theorem 9.2 then, it follows that $(\big[\mathbb{G}_{\mathcal{A}}\big], \delta)$ is a metric space. By $\mathcal{B}_\delta([Z], \rho)$ we denote the open ball and by $\overline{\mathcal{B}}_\delta([Z], \rho)$ the closed ball with center $[Z]$ and radius $\rho$. The subscript $\delta$ in $\mathcal{B}_\delta(Z, \rho)$ emphasizes that the underlying metric is the structural Frobenius metric.

An immediate consequence is that unlabeling preserves open and closed sets in some way.

***Lemma 10.1***
*Let $Z \in \mathbb{G}_{\mathcal{A}}$ be an attributed graph and let $\rho > 0$ be a positive constant. Then*

$$\begin{aligned} X \in \mathcal{B}_F(Z, \rho) \;&\Rightarrow\; [X] \in \mathcal{B}_\delta([Z], \rho) \\ X \in \overline{\mathcal{B}}_F(Z, \rho) \;&\Rightarrow\; [X] \in \overline{\mathcal{B}}_\delta([Z], \rho). \end{aligned}$$

***Proof*** The assertion directly follows from

$$\delta([X], [Y]) = \delta(X, Y) = \min_{\boldsymbol{P} \in \Pi^n} \|\boldsymbol{X}_{\boldsymbol{P}} - \boldsymbol{Y}\| \leq \|\boldsymbol{X} - \boldsymbol{Y}\|.$$

■

The converse statement of Lemma 10.1 is invalid, because labeling of an unlabeled graph is not well-defined. Next, we show that convergence of a sequence of labeled graphs is preserved by unlabeling.

***Lemma 10.2***
*Suppose that $(X_i) = (X_i)_{i=0}^{\infty}$ is a convergent sequence of attributed graphs in $\mathbb{G}_{\mathcal{A}}$ with limit $Z$. Then*

$$\lim_{i \to \infty} \big[X_i\big] = \big[Z\big].$$

***Proof*** Let $\varepsilon > 0$. Since $(X_i)$ is convergent to $Z$ in $\mathbb{G}_{\mathcal{A}}$, there is an $i_0 \geq 0$ such

that $X_i \in \mathcal{B}_F(Z, \varepsilon)$ for all $i > i_0$. From Lemma 10.1, it follows $[X_i] \in \mathcal{B}_\delta\big([Z], \varepsilon\big)$. This implies convergence of $\big([X_i]\big)$ to $[Z]$. ■

Suppose that $(X_i)$ is convergent with limit $Z$. From Lemma 10.1 follows that

$$\lim_{i \to \infty} X_i^\pi = Z^\pi$$

for all $\pi \in \mathcal{S}_n$. This shows that the convergence behavior is locally independent of the labeling of the vertices. For the same reasons as in Lemma 10.1, convergence of a sequence of unlabeled graphs does not imply convergence of a sequence of labeled graphs. This is illustrated by the following example.

***Example 10.1***
Suppose that $Z \in \mathbb{G}_\mathcal{A}$ is an attributed graph with $Z \neq Z^\pi$ for $\pi \in \mathcal{S}_n$. The sequence $([X_i])$ of unlabeled graphs in $\big[\mathbb{G}_\mathcal{A}\big]$ with $[X_i] = [Z]$ for all $i \geq 0$ is convergent with limit $[Z]$. Since labeling of the vertices of unlabeled graphs is not well-defined, there are $n!$ possibilities to label the vertices of each element of the sequence $([X_i])$. Depending on the chosen labeling, the resulting sequence of labeled graphs $(X_i)$ may or may not converge. If we choose $X_i = Z$ for all $i \geq 0$, then the sequence $(X_i)$ trivially converges to $Z$. But the sequence $(X_i)$ with

$$X_i = \begin{cases} Z^\pi & : \ i \mod 2 = 0 \\ Z & : \ i \mod 2 = 1 \end{cases}$$

for all $i \geq 0$ diverges.

Suppose that $(\mathcal{S}, d)$ is a metric space. A *Cauchy sequence* is a sequence $(X_i)$ in $\mathcal{S}$ so that for every positive real number $\varepsilon > 0$, there is an integer $i_0 \geq 0$ with

$$d(X_i, X_j) < \varepsilon$$

for all integers $i, j > i_0$. A metric space in which every Cauchy sequence has a limit is called *complete*. A complete normed vector space is a *Banach space*. Obviously, $\big(\mathbb{G}_\mathcal{A}, \|.\|\big)$ is a Banach space.

Since $\big[\mathbb{G}_\mathcal{A}\big]$ has no algebraic structure, it is not a vector space and therefore can never be a Banach space. Hence, the best we can do is to show whether $\big(\big[\mathbb{G}_\mathcal{A}\big], \delta\big)$ is a complete space.

***Theorem 10.1***
*In $\big[\mathbb{G}_\mathcal{A}\big]$ every Cauchy sequence converges to a limit in $\big[\mathbb{G}_\mathcal{A}\big]$.*

***Proof*** Let $\big([X_i]\big)$ be a Cauchy sequence in $\big[\mathbb{G}_\mathcal{A}\big]$. We first show that $\big([X_i]\big)$ is bounded. For this, we choose $\varepsilon = 1$. Since $\big([X_i]\big)$ is a Cauchy sequence, we can find an integer $i_0 > 0$ with

$$\delta\big([X_i], [X_j]\big) < 1$$

for all $i, j > i_0$. From Lemma 9.2 follows

$$\delta\big([X_i],[X_j]\big) = \delta(X_i, X_j) = \sqrt{\|X_i\|^2 - 2(X_i \bullet X_j) + \|X_j\|^2}.$$

Applying the structural Cauchy-Schwarz inequality (Theorem 9.3) yields

$$\begin{aligned}\delta(X_i, X_j) &\geq \sqrt{\|X_i\|^2 - 2\|X_i\|\|X_j\| + \|X_j\|^2} \\ &= \big\|X_i\big\| - \big\|X_j\big\|.\end{aligned}$$

This implies $\big\|X_i\big\| - \big\|X_j\big\| < 1$ for all $i, j > i_0$. In particular, for $k = i_0 + 1$, we have $\big\|X_i\big\| < 1 + \big\|X_k\big\|$ for all $i > i_0$. Hence, from

$$\big\|X_i\big\| \leq \max\Big\{\big\|X_1\big\|, \ldots, \big\|X_{i_0}\big\|, 1 + \big\|X_k\big\|\Big\} = M$$

for all $i \geq 0$, it follows that $\big([X_i]\big)$ is bounded. Clearly, the sequence $\big([X_i]\big)$ is contained in the closed ball $\overline{\mathcal{B}}_\delta(0, M)$, which is a compact subset of the metric space $\big[\mathbb{G}_\mathcal{A}\big]$. Then by the theorem of Bolzano-Weierstrass for metric spaces, the sequence $\big([X_i]\big)$ has a convergent subsequence $\big([X'_i]\big)$ with limit $[Z]$. It remains to show that the whole sequence $\big([X_i]\big)$ converges to $[Z]$. For any $\varepsilon > 0$, we can find an $i_0 \geq 0$ with

$$\delta\big([X_i],[X_j]\big) < \frac{\varepsilon}{2}$$

for all $i, j > i_0$. Since $\big([X'_i]\big)$ is a subsequence with limit $[Z]$, there is a $k > i_0$ with

$$\delta\big([X_k],[Z]\big) < \frac{\varepsilon}{2}.$$

Since $\delta$ is a metric, we have

$$\delta\big([X_i],[Z]\big) \leq \delta\big([X_i],[X_k]\big) + \delta\big([X_k],[Z]\big) < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

This shows that $\big([X_i]\big)$ converges to $[Z]$. ■

From Theorem 10.1, it follows that $\big(\big[\mathbb{G}_\mathcal{A}\big], \delta\big)$ is a complete metric space.

We conclude this section with some remarks on product spaces of labeled and unlabeled graphs. Since $\mathbb{G}_\mathcal{A}$ is isomorphic to a Euclidean space of dimension $N$, the product space

$$\mathbb{G}_\mathcal{A}^p = \underbrace{\mathbb{G}_\mathcal{A} \times \cdots \times \mathbb{G}_\mathcal{A}}_{\text{p times}}$$

is isomorphic to a Euclidean space of dimension $pN$. An element of $\mathbb{G}_\mathcal{A}$ is a vector of the form $(X_i)_{i=1}^p$ with $X_i \in \mathbb{G}_\mathcal{A}$ for all $i \in [1\!:\!p]$. The generalized Frobenius norm on $\mathbb{G}_\mathcal{A}^p$ is defined by

$$\big\|(X_i)_{i=1}^p\big\|^2 = \sum_{i=1}^p X_i \bullet X_i.$$

Using the structural Minkowski inequality proven in Theorem 9.4, it is straightforward to check that $\|.\|$ is also a norm on the product space $\mathbb{G}_{\mathcal{A}}^p$. The norm on the product space induces the generalized Frobenius metric on $\mathbb{G}_{\mathcal{A}}^p$ in the usual way. The structural Frobenius metric of $(X_i)_{i=1}^p$ and $(Y_i)_{i=1}^p$ is given by

$$\delta\Big((X_i)_{i=1}^p, (Y_i)_{i=1}^p\Big)^2 = \big\|(X_i)_{i=1}^p\big\|^2 - 2\sum_{i=1}^p X_i \bullet Y_i + \big\|(Y_i)_{i=1}^p\big\|^2.$$

## 10.3 Functions on Attributed Graphs

This section introduces and studies functions on attributed graphs that are invariant under relabeling of the vertices.

A *function on attributed graphs* is a mapping

$$f : \mathbb{G}_{\mathcal{A}}^p \to \mathcal{S}, \quad \big(X_1, \ldots, X_p\big) \mapsto f\big(X_1, \ldots, X_p\big). \tag{10.1}$$

We say $f$ is a *structural function* if it satisfies the *permutation invariance property*

$$f\big(X_1, \ldots, X_p\big) = f\left(X_1^{\pi_1}, \ldots, X_p^{\pi_p}\right)$$

for all $\big(X_1, \ldots, X_p\big) \in \mathbb{G}_{\mathcal{A}}^p$ and all permutations $\pi_1, \ldots, \pi_p \in \mathcal{S}_n$. For convenience of presentation, we focus on functions defined on $\mathbb{G}_{\mathcal{A}}$.

Suppose that $f : \mathbb{G}_{\mathcal{A}} \to \mathcal{S}$ is a structural function. The unlabeling operation gives rise to a uniquely determined function $\widehat{f} : \big[\mathbb{G}_{\mathcal{A}}\big] \to \mathcal{S}$ so that the following diagram commutes:

$$\begin{array}{ccc} \mathbb{G}_{\mathcal{A}} & \xrightarrow{[.]} & \big[\mathbb{G}_{\mathcal{A}}\big] \\ \big\downarrow f & & \big\downarrow \widehat{f} \\ \mathcal{S} & \xrightarrow{\text{id}} & \mathcal{S} \end{array}$$

Uniqueness of $\widehat{f}$ follows from the permutation invariance property of $f$ and $[.]$. Hence, we may identify structural functions $f$ on $\mathbb{G}_{\mathcal{A}}$ with functions on $\big[\mathbb{G}_{\mathcal{A}}\big]$.

A useful property of structural functions is that they are closed under linear combination, multiplication, and composition.

***Proposition 10.1***
*Let $w_1, w_2 \in \mathbb{R}$. Suppose that $f, f_1, f_2 : \mathbb{G}_{\mathcal{A}} \to \mathcal{S}$ and $g : \mathbb{G}_{\mathcal{A}} \to \mathbb{G}_{\mathcal{A}}$ are structural functions.*

1. $w_1 f_1 + w_2 f_2 : \mathbb{G}_{\mathcal{A}} \to \mathcal{S}, \quad X \mapsto w_1 f_1(X) + w_2 f_2(X)$
2. $f_1 f_2 : \mathbb{G}_{\mathcal{A}} \to \mathcal{S}, \quad X \mapsto f_1(X) f_2(X)$
3. $f \circ g : \mathbb{G}_{\mathcal{A}} \to \mathcal{S}, \quad X \mapsto f(g(X))$

*are structural functions.*

***Proof*** Trivial. ■

Next, we provide simple examples of functions on attributed graphs, which are partly important for the following considerations.

***Example 10.2***
Structural proximity measures and graph invariants like maximum clique weight, weighted degree, diameter, and so on are structural functions on graphs. In addition, consider the following functions:

1. *Unlabeling operation.* The function

$$[.] : \mathbb{G}_{\mathcal{A}} \to \big[\mathbb{G}_{\mathcal{A}}\big], \quad X \mapsto [X]$$

is trivially a structural function.

2. *Linear map.* Let $W \in \mathbb{G}_{\mathcal{A}}$ be a graph with adjacency matrix $\boldsymbol{W}$. Then the function

$$L_W : \mathbb{G}_{\mathcal{A}} \to \mathbb{R}, \quad X \mapsto \boldsymbol{W} \bullet \boldsymbol{X}$$

violates, in general, the permutation invariant property.

3. *Rotation/Relabeling.* Let $\pi \in \mathcal{S}_n$ be a permutation. The function

$$R_\pi : \mathbb{G}_{\mathcal{A}} \to \mathbb{G}_{\mathcal{A}}, \quad X \mapsto X^\pi$$

trivially violates the permutation invariant property. A rotation or relabeling is a *homomorphism* of vector spaces in the sense that

$$\begin{aligned} R_\pi(X + Y) &= R_\pi(X) + R_\pi(Y) \\ R_\pi(wX) &= wR_\pi(X) \end{aligned}$$

for all $X, Y \in \mathbb{G}_{\mathcal{A}}$ and all $w \in \mathbb{R}$. Since the second equation is evident, we only show the first equation. Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be the adjacency matrices of $X$ and $Y$. Suppose that $\boldsymbol{P}$ is the permutation matrix representing $\pi$. We have

$$\begin{aligned} R_\pi(X + Y) = (X + Y)^\pi &= \boldsymbol{P}^\mathsf{T}(\boldsymbol{X} + \boldsymbol{Y})\boldsymbol{P} \\ &= \boldsymbol{P}^\mathsf{T}\boldsymbol{X}\boldsymbol{P} + \boldsymbol{P}^\mathsf{T}\boldsymbol{Y}\boldsymbol{P} \\ &= X^\pi + Y^\pi = R_\pi(X) + R_\pi(Y). \end{aligned}$$

4. *Parameterized structural dot product.* Let $W \in \mathbb{G}_{\mathcal{A}}$ be a graph. The structural dot product

$$S_W : \mathbb{G}_{\mathcal{A}} \to \mathbb{R}, \quad X \mapsto W \bullet X$$

with parameter $W$ is a structural function. The function $S_W$ can be expressed as a pointwise maximizer of linear maps

$$S_W(X) = \max_{\pi \in \mathcal{S}_n} L_{W^\pi}(X) = \max_{\boldsymbol{P} \in \Pi^n} \boldsymbol{W_P} \bullet \boldsymbol{X},$$

where $\boldsymbol{W_P}$, $\boldsymbol{W}$, and $\boldsymbol{X}$ denote the adjacency matrices of $W^\pi$, $W$, and $X$.

A metric space has sufficient structure to define continuity of functions. The next result shows that continuity of a structural function $f$ at $Z$ implies continuity of $\widehat{f}$ at $[Z]$.

***Lemma 10.3***
*Let $(\mathcal{S}, d)$ be a metric space. Suppose that $f : \mathbb{G}_{\mathcal{A}} \to \mathcal{S}$ is a structural function. If $f$ is continuous at $Z \in \mathbb{G}_{\mathcal{A}}$, then the function $\hat{f}$ is continuous at $[Z]$.*

***Proof*** Let $\varepsilon > 0$. From continuity of $f$ at $Z$, it follows that there exists a positive constant $\rho > 0$ with

$$d\big(f(X), f(Z)\big) < \varepsilon$$

for all $X \in \mathcal{B}_F(Z, \rho)$. From Lemma 10.1, together with the fact that $f$ is a structural function, it follows that

$$d\big(\hat{f}([X]), \hat{f}([Z])\big) = d\big(f(X), f(Z)\big) < \varepsilon$$

for all $[X] \in \mathcal{B}_\delta([Z], \rho)$. Hence, $\hat{f}$ is convergent at $[Z]$. ■

Lemma 10.3 is useful because in order to determine continuity of a function $\widehat{f}$ at $[Z]$, it is sufficient to check whether $f$ is continuous for an arbitrary labeling of $[Z]$, e.g. $Z$. In addition, from continuity of $f$ at $Z$, we can conclude continuity of $f$ at $Z^\pi$ for all $\pi \in \mathcal{S}_n$.

***Example 10.3***
The functions (1)-(4) from Example 10.2 are continuous. Note that only functions (1) and (4) are structural functions.

1. *Unlabeling operation*: Continuity of [.] follows from Lemma 10.2.

2. *Linear map*: Continuity of $L_W$ follows from the continuity of linear maps in Euclidean spaces.

3. *Rotation/Relabeling*: Continuity of $R_\pi$ follows from Lemma 10.3.

4. *Parameterized structural dot product*: Continuity of $S_W$ follows from the fact that a pointwise maximizer of continuous linear maps is continuous.

## 10.4 Differentiability

A Banach space provides enough structure to define concepts of a derivative. Hence, it is straightforward to apply an analysis of differentiable functions to structural functions on $\mathbb{G}_{\mathcal{A}}$. Since algebraic operations in $\big[\mathbb{G}_{\mathcal{A}}\big]$ are unknown, we cannot define a derivative of functions on $\big[\mathbb{G}_{\mathcal{A}}\big]$. But we can use the complete metric space $\big[\mathbb{G}_{\mathcal{A}}\big]$ to investigate whether the derivative of a function on $\mathbb{G}_{\mathcal{A}}$ is a well-defined graph-theoretical concept.

Consider a function $f : \mathbb{G}_{\mathcal{A}} \to \mathbb{R}$. The *Fréchet derivative* of $f$ at $X \in \mathbb{G}_{\mathcal{A}}$ is a linear map

$$Df(X) : \mathbb{G}_{\mathcal{A}} \to \mathbb{R}, \quad W \mapsto Df(X) \cdot W = Df(X)(W)$$

such that the limit

$$\lim_{\|W\|\to 0} \frac{\big(f(X+W)-f(X)\big)-Df(X)\cdot W}{\|W\|} = 0$$

exists for any non-zero $W \in \mathbb{G}_{\mathcal{A}}$. The function $f$ is said to be *Fréchet differentiable* at $X$ if its derivative exists at that point. If $\mathcal{S} \subseteq \mathbb{G}_{\mathcal{A}}$, then $f$ is said to be differentiable on $\mathcal{S}$ if $f$ is differentiable at every point $X \in \mathcal{S}$. Under these conditions the following properties of the Fréchet derivative are preserved:

1. Differentiability at $X \in \mathbb{G}_{\mathcal{A}}$ implies continuity at $X$.
2. $Df = 0$, where $f$ is constant.
3. $Df = W$, where $f(X) = \boldsymbol{W} \bullet \boldsymbol{X} + h$ is a linear map.
4. Existence of rules of calculus.
   (a) *Linearity*:
   $$D\big(w_1 f_1 + w_2 f_2\big)(X)\cdot W = w_1 Df_1(X)\cdot W + w_2 Df_2(X)\cdot W$$
   (b) *Product Rule*:
   $$D(f_1 f_2)(X)\cdot W = f_2(X)Df_1(X)\cdot W + f_1(X)Df_2(X)\cdot W$$
   (c) *Chain Rule*:
   $$D(f_2 \circ f_1)(X)\cdot W = Df_2(f_1(X))\cdot\big(Df_1(X)\cdot W\big)$$

From elementary calculus, it follows that the derivative of a structural function $f$ at $X$ is a linear map of the form

$$Df(X)\cdot W = L_G(W) = \boldsymbol{G} \bullet \boldsymbol{W}.$$

The graph $G$ determined by the adjacency matrix $\boldsymbol{G}$ is the *gradient* $\nabla f(X)$ of $f$ at $X$ pointing in the direction of steepest ascent. We pose two crucial questions:

1. *Does the derivative of $f$ at $X^\pi$ exist for all $\pi \in \mathcal{S}_n$?*
2. *Is the gradient of $f$ at $X$ isomorphic to the gradient of $f$ at $X^\pi$?*

Theorem 10.2 answers both questions positively.

***Theorem 10.2***
*Let $f : \mathbb{G}_{\mathcal{A}} \to \mathbb{R}$ be Fréchet differentiable at $X \in \mathbb{G}_{\mathcal{A}}$. If $f$ is a structural function, then $f$ is Fréchet differentiable at $X^\pi \in \mathbb{G}_{\mathcal{A}}$ with*

$$Df(X) = R_\pi \circ Df\left(X^\pi\right) \tag{10.2}$$

*for all $\pi \in \mathcal{S}_n$.*

***Proof*** Let $(Df)^\pi\left(X^\pi\right)$ be a shortcut notation for the composition $R_\pi \circ Df\left(X^\pi\right)$. We first assume that $f$ is Fréchet differentiable at $X^\pi \in \mathbb{G}_{\mathcal{A}}$ for all $\pi \in \mathcal{S}_n$ and

prove (10.2). Suppose that there is a permutation $\pi \in \mathcal{S}_n$ with

$$L = Df(X) \neq (Df)^\pi (X^\pi) = L'.$$

Note that $L$ and $L'$ are linear maps on $\mathbb{G}_\mathcal{A}$. For $W \neq 0$ we define the mappings

$$r(W) = \frac{f(X+W) - f(X) - L(W)}{\|W\|}$$

$$r_\pi(W) = \frac{f\left(X^\pi + W^\pi\right) - f\left(X^\pi\right) - L'(W^\pi)}{\|W\|}.$$

Note that the generalized Frobenius norm is independent of the labeling, i.e. $\|W\| = \|W^\pi\|$. Since $f$ is Fréchet differentiable at $X$ and $X^\pi$, we have

$$\lim_{\|W\| \to 0} r(W) = 0 \quad \text{and} \quad \lim_{\|W\| \to 0} r_\pi(W) = 0$$

for any sequence $W \to 0$. Using the mappings $r$ and $r_\pi$, we approximate $f$ in a neighborhood of $X$ and $X^\pi$ by the linear functions $L$ and $L'$ and obtain

$$f\big(X+W\big) = f\big(X\big) + L(W) + r\big(W\big)\,\|W\| \tag{10.3a}$$

$$f\big(X^\pi + W^\pi\big) = f\big(X^\pi\big) + L'(W^\pi) + r_\pi\big(W\big)\,\|W\|\,. \tag{10.3b}$$

From linearity of $R_\pi$, it follows that $X^\pi + W^\pi = (X+W)^\pi$. Since $f$ is a structural function, we have $f(X+W) = f((X+W)^\pi) = f(X^\pi + W^\pi)$ and $f(X) = f(X^\pi)$. Subtracting equation (10.3b) from (10.3a) yields

$$L(W) - L'(W^\pi) = \big(r_\pi(W) - r(W)\big) \cdot \|W\|. \tag{10.4}$$

Let $w \in \mathbb{R}$ with $w \neq 0$, and let $H \neq 0$ be a complex attributed graph from $\mathbb{G}_\mathcal{A}$. Since equation (10.4) holds for any $W$, it also holds for $W = wH$. For sufficiently small $w$, equation (10.4) reduces to

$$L(H) - L'(H^\pi) = \big(r_\pi(wH) - r(wH)\big)\|H\|.$$

Taking the limit $w \to 0$ yields

$$L(H) - L'(H^\pi) = 0.$$

Let us rewrite the last equation in terms of attributed matrices. Suppose that the linear mappings $L$ and $L'$ are of the form $L(X) = \boldsymbol{L} \bullet \boldsymbol{X}$ and $L'(X) = \boldsymbol{L'} \bullet \boldsymbol{X}$. With $\boldsymbol{P} \in \Pi_n$ as a matrix representation of $\pi$, we have

$$\begin{aligned} L(H) - L'(H^\pi) &= \boldsymbol{L} \bullet \boldsymbol{H} - \boldsymbol{L'} \bullet \boldsymbol{P}^\mathsf{T}\boldsymbol{H}\boldsymbol{P} \\ &= \boldsymbol{L} \bullet \boldsymbol{H} - \boldsymbol{P}\boldsymbol{L'}\boldsymbol{P}^\mathsf{T} \bullet \boldsymbol{H} \\ &= 0. \end{aligned}$$

Since $H$ was chosen arbitrarily, we have $\boldsymbol{L} = \boldsymbol{P}\boldsymbol{L'}\boldsymbol{P}^\mathsf{T}$ and therefore, $\boldsymbol{L'} = \boldsymbol{P}^\mathsf{T}\boldsymbol{L}\boldsymbol{P}$. Hence, $L' = L^\pi = (Df)^\pi(X^\pi)$ contradicts our assumption and proves equation

(10.2).

Now it is easy to show that $f$ is Fréchet differentiable at $X^\pi$ for all $\pi \in \mathcal{S}_n$. Consider the mapping $r_\pi$ defined by

$$r_\pi(W) = \frac{f(X^\pi + W^\pi) - f(X^\pi) - L^\pi(W)}{\|W\|}.$$

Using a similar argumentation as in the first part of the proof, we have

$$r_\pi(W) = \frac{f(X + W) - f(X) - \boldsymbol{P}^\top \boldsymbol{L}\boldsymbol{P} \bullet \boldsymbol{P}^\top \boldsymbol{W}\boldsymbol{P}}{\|W\|}.$$

From

$$\boldsymbol{P}^\top \boldsymbol{L}\boldsymbol{P} \bullet \boldsymbol{P}^\top \boldsymbol{W}\boldsymbol{P} = \boldsymbol{L} \bullet \boldsymbol{W}$$

follows

$$\begin{aligned} r_\pi(W) &= \frac{f(X + W) - f(X) - \boldsymbol{L} \bullet \boldsymbol{W}}{\|W\|} \\ &= \frac{f(X + W) - f(X) - L(W)}{\|W\|} \\ &= r_W. \end{aligned}$$

The existence of the Fréchet derivative of $f$ at $X^\pi$ follows by taking the limit. ■

From Theorem 10.2, it follows that

1. the existence of a derivative is independent from the labeling and
2. the gradients at isomorphic graphs are isomorphic.

In particular, the gradient of $f$ at $X^\pi$ is obtained by the gradient of $X$ in the same way as $X^\pi$ is obtained from $X$.

***Example 10.4***
Consider functions (1)-(4) from Example 10.2. Since we have defined the derivative for real-valued functions only, we are not able to discuss differentiability for all four functions.

1. *Unlabeling operation*: The unlabeling operation [.] is not real-valued.
2. *Linear map*: A linear map $L_W(X) = \boldsymbol{W} \bullet \boldsymbol{X}$ is differentiable with gradient $\nabla f(X) = \boldsymbol{W}$. It is easy to verify that $DL_W(X)$ depends continuously on $X$.
3. *Rotation/Relabeling*: The rotation $\mathbb{R}_\pi$ is not real-valued.
4. *Parameterized structural dot product*: A pointwise maximizer of Fréchet differentiable functions is in general not Fréchet differentiable. As we will see in the next chapter, the parameterized structural dot product $S_W$ is Fréchet differentiable almost everywhere except on a set with Lebesgue measure zero.

We conclude this section by presenting a necessary first-order condition for a local optimum solution $X_*$ of a continuously Fréchet differentiable (smooth) function $f$. Analogous to elementary calculus, the gradient at a local optimal solution $X_*$ is the null graph 0. Note that the null graph is invariant under relabeling.

## 10.5 Optimization

In this section, we address the issue of optimizing Fréchet differentiable functions on attributed graphs from the perspective of minimizing error functions of neural learning machines.

A simple learning problem in structural pattern recognition, including the training of neural networks, can be posed as an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & \widehat{f}(X) = \sum_{q=1}^{p} \widehat{f}_i(X) \\ \text{subject to} & X \in \left[\mathbb{G}_{\mathcal{A}}\right], \end{array} \tag{10.5}$$

where $\widehat{f}_q : \left[\mathbb{G}_{\mathcal{A}}\right] \to \mathbb{R}$ are the component functions of $\widehat{f}$. To solve problem (10.5) using optimization methods based on local gradient information, we relax the domain $\left[\mathbb{G}_{\mathcal{A}}\right]$ and move to the space $\mathbb{G}_{\mathcal{A}}$. Then problem (10.5) is equivalent to

$$\begin{array}{ll} \text{minimize} & f(X) = \sum_{q=1}^{p} f_i(X) \\ \text{subject to} & X \in \mathbb{G}_{\mathcal{A}}, \end{array} \tag{10.6}$$

where $f$ is the unique structural function with $f(X) = \hat{f}([X])$. We assume that the component functions $f_q : \mathbb{G}_{\mathcal{A}} \to \mathbb{R}$ are differentiable. Then the objective $f$ is also a differentiable structural function.

Note that this assumption is stronger than it may appear at first glance. For instance, the parameterized structural dot product $S_W$ fails to satisfy this condition at exceptional points. Hence, we can not assume that useful error functions built upon the structural dot product are differentiable. But what we can say is that such error functions are differentiable almost everywhere, as shown in the next chapter. Therefore, it is justifiable to first investigate the smooth case before turning to nonsmooth functions.

Nonlinear optimization of differentiable functions is one of the most important areas of applied mathematics. An overview of standard techniques can be found in [26, 27, 94, 239]. Since $\mathbb{G}_{\mathcal{A}}$ is a Banach space isomorphic to a Euclidean vector space, optimization methods that operate in $\mathbb{R}^n$ also work in $\mathbb{G}_{\mathcal{A}}$.

Typically, optimization procedures are iterative algorithmic maps $\mathsf{A}$ that start from a given initial point $X_0 \in \mathbb{G}_{\mathcal{A}}$. The algorithmic map iteratively generates a sequence $(X_t)$ of labeled graphs by $X_{t+1} \in \mathsf{A}(X_t)$, which hopefully converges to an optimal solution $X_*$. A basic algorithm to iteratively solve problem (10.6) has the following form:

**Algorithm 9** (Basic Iterative Algorithm)

**Initialization:**

set $X_0 \in \mathbb{G}_{\mathcal{A}}$ and $t = 0$

**Procedure:**

**repeat**

find $D_t \in \mathbb{G}_{\mathcal{A}}$ with $f(X_t + \eta D_t) < f(X_t)$ for some $\eta > 0$ (*direction finding*)

find $\eta_t > 0$ such that $\eta_t \approx \arg\min_{\eta > 0} f(X_t + \eta D_t)$ (*line search*)

set $X_{t+1} = X_t + \eta_t D_t$ and $t = t + 1$ (*updating*)

**until** $\boldsymbol{x}_t$ is sufficiently close to $X_*$ (*termination criterion*)

**Output:** $X_t$

The simplest iterative algorithms have been devised for problems (10.5) with smooth objective $f$. Popular examples are gradient, line conjugate gradient, and (Quasi-) Newton methods. All these approaches exploit derivative information from the objective function $f$. Direction finding may reduce to determine the direction opposite to the gradient, which is locally the direction of steepest descent. Line search usually applies efficient univariate methods from optimization of differentiable functions or some polynomial interpolation. Finally, useful termination criteria can be derived from the necessary condition for a local optimum of smooth functions, stating that the gradient must be zero at each local solution.

New in Algorithm 9 is that the direction $D_t$ is an attributed graph, which locally points in a direction of ascent. The direction $D_t$ is structurally independent of the labeling of $X_t$. Suppose that $X_t^\pi$ is a relabeled version of $X_t$ for some $\pi \in \mathcal{S}_n$. Then

$$\begin{aligned} f\big(X_t + \eta D_t\big) &= f\big(R_\pi(X_t + \eta D_t)\big) \\ &= f\big(R_\pi(X_t) + \eta R_\pi(D_t)\big), \end{aligned}$$

where equality in the first line follows from the fact that $f$ is invariant under permutations. Equality in the second line follows from linearity of relabeling. Hence, we have

$$X'_{t+1} = X_t^\pi + \eta_t D_t^\pi = \big(X_t + \eta_t D_t\big)^\pi = X_{t+1}^\pi.$$

Clearly, $X_{t+1}$ and $X'_{t+1}$ are structurally equivalent (isomorphic). This shows that Algorithm 9 is independent of the labeling and performs a well-defined descent of the objective function $\widehat{f}$ of the original problem (10.5).

Basically, we may choose the most effective optimization technique to minimize the error function of a neural network. For a discussion on efficient learning algorithms for standard neural networks, we refer to [32, 138, 275]. Since our goal is to construct structural neural learning machines, it is sufficient to consider any optimization method. Once constructive existence of such systems has been proved, the focus may shift to efficiency. Hence, it is compatible with our goals to restrict

ourselves to simple optimization methods.

For training neural networks in the domain of feature vectors, *incremental gradient descent* is a simple standard technique for minimizing the network's error function. Incremental gradient descent in its simplest form differs from standard gradient descent in that at each iteration $t$, $X$ is changed incrementally by a sequence of $p$ sub-iterations. Each sub-iteration $q$ is a gradient descent step for the $q$-th component function $f_q$. Thus, an iteration step from $t$ to $t+1$ consists of a cycle of $p$ sub-iteration steps. Incremental gradient methods are supported by a number of theoretical convergence analyses [28, 101, 128, 245, 246, 251, 360], and by empirical evidence of superior convergence behavior compared with standard (batch) gradient methods [275]. Algorithm 10 presents an incremental gradient descent procedure, which cycles through the component functions $f_q$ with fixed order.[1]

**Algorithm 10** (Incremental Gradient Descent Algorithm)

**Initialization:**

set $X_0 \in \mathbb{G}_{\mathcal{A}}$ and $t = 0$

**Procedure:**

**repeat**
  set $\widetilde{X}_{t,1} = X_t$
  **for** $q \in [1:p]$
    find $G_{t,q} = \nabla f_q(\widetilde{X}_{t,q})$ (*direction finding*)
    choose $\eta_{t,q} > 0$ (*line search*)
    set $\widetilde{X}_{t,q+1} = \widetilde{X}_{t,q} + \eta_{t,q} G_{t,q}$ (*incremental-updating*)
  **end for**
  set $X_{t+1} = \widetilde{X}_{t,p+1}$ and $t = t+1$ (*updating*)
**until** $X_t$ sufficiently close to $X_*$ (*termination criterion*)

**Output:** $X_t$

## 10.6 Conclusion

In this chapter we presented a structurally consistent approach to the differential analysis of structural functions on attributed graphs. The key result is that the gradient of a real-valued structural function at some graph is a well-defined graph pointing in the direction of steepest ascent. This result allows us to apply any gradient-based optimization method to optimize differentiable structural functions.

1. In a practical setting, the component functions of $f$ are usually randomly chosen at each sub-iteration step. For analytical purposes, we assume a fixed order.

Three ideas have been combined to establish differential analysis on structural functions: first, construction of the structural dot product in analogy to the conventional dot product, which is the *root* of Euclidean spaces. Second, relaxation of the notion of a real graph to a complex graph in order to obtain complete metric spaces. For incomplete spaces, transition to the limit may not be possible and the derivative is undefined. Third, coeval consideration of the space of labeled graphs together with the space of unlabeled graphs. The former space enables us to apply analytical concepts and the latter keeps track of the structural information of those concepts.

We conclude this chapter with a twofold outlook. First, note that we only have derived results necessary for the construction of neural learning machines. But the door is open to explore the analysis and calculus of variations for structural functions. Further investigations might provide useful tools to efficiently solve problems in structural pattern recognition using alternative learning procedures. Second, the key problem is that error functions of neural learning machines are functions built upon the parameterized structural dot product. Since the structural dot product is not differentiable, we cannot expect that useful error functions are differentiable. Hence, gradient-based techniques are inapplicable and lead to the field of nonsmooth analysis, which will be discussed in the next chapter.

# 11 Nonsmooth Analysis

Learning is usually posed as an optimization problem. If the objective function of the optimization problem is smooth, we can exploit derivative information to perform local descent. But if the objective is nonsmooth, we need generalized concepts of a derivative. In this chapter, we introduce basic definitions and results from nonsmooth analysis and apply them to nonsmooth structural functions.

## 11.1 Introduction

As shown in the previous chapter, the parameterized structural dot product is nonsmooth. We can therefore assume that error functions defined on the structural dot product are in general also nonsmooth. To minimize nonsmooth functions, we need generalized concepts of a derivative. The study of generalized derivative concepts emerged in the early 1970's and has now gradually developed into a mature field of mathematics called *nonsmooth analysis*.

The success of the Fréchet derivative concept to analyze smooth functions is based on the following properties:

($D_1$) The function $g(\boldsymbol{x}) = f(\boldsymbol{\xi}) + Df(\boldsymbol{\xi})(\boldsymbol{x} - \boldsymbol{\xi})$ is a first order approximation of $f$ at $\boldsymbol{\xi}$ such that

$$\lim_{\boldsymbol{x} \to \boldsymbol{\xi}} \frac{\big|f(\boldsymbol{x}) - g(\boldsymbol{x})\big|}{\|\boldsymbol{x} - \boldsymbol{\xi}\|} = 0. \tag{11.1}$$

($D_2$) The first order approximation $g$ of $f$ is an affine function that is considerably simpler than the original function $f$.[1]

($D_3$) The rules of calculus enable us to compute the derivative of complicated functions that are compositions of simpler functions.

To analyze a function $f$ in a neighborhood of some point $\boldsymbol{\xi}$, we can only exploit

1. An affine function is a function of the form

$$g : \mathbb{R}^n \to \mathbb{R}^m, \quad \boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b},$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$.

the properties of a Fréchet derivative when $f$ is actually Fréchet differentiable at $\boldsymbol{\xi}$. This requirement is too strong for many applications including, for example, control theory and nonlinear programming. Therefore, various alternative concepts of the Fréchet and other classical derivatives of smooth analysis have been suggested, which allow us to establish an analysis of nonsmooth functions. The proposed generalized derivative concepts replace the nonexistent classical derivative to describe the local behavior for every point in the domain of functions of a particular class.

The starting point for the analysis of nonsmooth function is the theory of convex sets and convex functions by Fenchel, Rockefellar, and others for which [301] is a standard reference. Clarke generalized the subdifferential concept of convex analysis to the broader class of locally Lipschitz continuous functions by introducing a relaxed version of the directional derivative [58]. His contribution was a milestone in nonconvex, nonsmooth analysis and initiated a vivid activity for systematically studying nonsmooth problems. Clarke's approach, however, has some limitations that lead to developments of other generalized derivative concepts, for example the *co-derivatives* proposed by Mordukhovich [267, 268], *Michel and Penot's derivatives* [259], and *B-derivatives* introduced by Treiman [355, 356]. Although the different concepts of generalized derivatives are very useful for analytical purposes, their definitions are often complicated and hard to calculate. Since we have an additional complexity overhead when dealing with graphs, the focus is on derivative concepts that are simple to compute, yet sufficiently general in order to obtain the necessary theoretical results.

This chapter introduces basic definitions and results from nonsmooth analysis tailored to the specific needs of structural neural learning machines. For more general and comprehensive treatments we refer to [58, 301]. Section 11.2 provides the basic mathematical toolkit. In Section 11.3, we apply the results of nonsmooth analysis to the parameterized structural dot product. Optimization methods for nonsmooth structural functions are discussed in Section 11.4. Finally, Section 11.5 concludes this chapter.

## 11.2 Generalized Gradients

In this section we gather the basic toolkit that will be used later to minimize nonsmooth error functions on attributed graphs. Parts of the treatment are based on [58, 240].

### Terminology

Throughout this section, our setting is the vector space $\mathbb{R}^n$ equipped with an inner product $\boldsymbol{x}^\mathsf{T}\boldsymbol{y}$ and the usual norm $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^\mathsf{T}\boldsymbol{x}}$. Let $\boldsymbol{\xi} \in \mathbb{R}^n$ be a point. By $\boldsymbol{x} \to \boldsymbol{\xi}$ we denote a sequence $(\boldsymbol{x})_{i=0}^{\infty}$ in $\mathbb{R}^n$ with $\lim_{i\to\infty} \boldsymbol{x}_i = \boldsymbol{\xi}$. Similarly, $\alpha \downarrow 0$ is a notational shortcut for a sequence $(\alpha)_{i=0}^{\infty}$ in $\mathbb{R}_+^0$ with $\lim_{i\to\infty} \alpha_i = \boldsymbol{\xi}$. We define the sum of

two sets $\mathcal{S}$ and $\mathcal{S}'$ in $\mathbb{R}^n$ by

$$\mathcal{S} + \mathcal{S}' = \{\boldsymbol{x} + \boldsymbol{y} : \boldsymbol{x} \in \mathcal{S}, \boldsymbol{y} \in \mathcal{S}'\}.$$

For $w \in \mathbb{R}$ we define

$$w\mathcal{S} = \{w\boldsymbol{x} : \boldsymbol{x} \in \mathcal{S}\}.$$

### The Lipschitz Condition

An important class of nonsmooth functions are locally Lipschitz continuous functions. Let $U \subseteq \mathbb{R}^n$ be an open set. A function $f : U \to \mathbb{R}$ is *Lipschitz continuous* on $V \subseteq U$ if there exists a constant $L$ such that

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq L\|\boldsymbol{x} - \boldsymbol{y}\| \tag{11.2}$$

for all $\boldsymbol{x}, \boldsymbol{y} \in V$. The constant $L$ satisfying (11.2) is called *Lipschitz constant* for $f$ on $V$. The function $f$ is *locally Lipschitz continuous* if every point $\boldsymbol{x} \in U$ admits a neighborhood $V \subseteq U$ of $\boldsymbol{x}$ on which $f$ is Lipschitz continuous.

Note that a locally Lipschitz continuous function needs neither be differentiable, nor admits directional derivatives in the classical sense. Conversely, it can be shown that a differentiable function is locally Lipschitz continuous.

According to Rademacher's Theorem 11.1, locally Lipschitz functions are Fréchet differentiable almost everywhere. Let $\mathcal{D}_f \subseteq U$ denote the set of all $\boldsymbol{x} \in U$ at which $f$ admits a Fréchet derivative $Df(\boldsymbol{x})$.

***Theorem 11.1* (*Rademacher's Theorem*)**
*Let $U \subseteq \mathbb{R}^n$ be a nonempty open set, and let $f : U \to \mathbb{R}$ be a locally Lipschitz continuous function. Then the set*

$$\Omega_f = U \setminus \mathcal{D}_f$$

*has Lebesgue measure zero.*

***Proof*** [397]. ■

### Directional Derivatives and Bouligand Subdifferentials

Consider the function $f : U \to \mathbb{R}$ defined on the open set $U \subseteq \mathbb{R}^n$. The function $f$ is *directionally differentiable* at $\boldsymbol{\xi} \in U$ if the limit

$$f'(\boldsymbol{\xi}, \boldsymbol{d}) = \lim_{\alpha \downarrow 0} \frac{f(\boldsymbol{\xi} + \alpha\boldsymbol{d}) - f(\boldsymbol{\xi})}{\alpha}$$

exists for all directions $\boldsymbol{d} \in \mathbb{R}^n$. In this case, the value $f'(\boldsymbol{\xi}, \boldsymbol{d})$ is the *directional derivative* of $f$ at $\boldsymbol{\xi}$ in the direction $\boldsymbol{d}$. We call the function $f$ directionally differentiable if $f$ is directionally differentiable at all points $\boldsymbol{x} \in U$.

The directional derivative of $f$ at $\boldsymbol{\xi}$ can be regarded as a mapping

$$f'(\boldsymbol{\xi},.) : \mathbb{R}^n \to \mathbb{R}, \quad \boldsymbol{d} \mapsto f'(\boldsymbol{\xi},\boldsymbol{d}).$$

If $f'(\boldsymbol{\xi},.)$ is a linear mapping, then $f$ is *Gâteaux differentiable* at $\boldsymbol{\xi}$. Thus, the directional derivative naturally generalizes the classical Gâteaux derivative. From elementary calculus, it follows that Gâteaux differentiability is a weaker concept than Fréchet differentiability in the sense that a Fréchet differentiable function is also Gâteaux differentiable, but the converse statement does not hold in general. For locally Lipschitz continuous functions on finite dimensional Banach spaces, however, Gâteaux and Fréchet differentiability coincide.

Although the directional derivative is usually practicable to determine, it is in general of limited interest because of its poor first order approximation properties. But as for Gâteaux and Fréchet differentiability in smooth analysis, the situation is different for locally Lipschitz continuous functions on finite dimensional Banach spaces.

***Theorem 11.2***
*Let $f : U \to \mathbb{R}$ be locally Lipschitz continuous on the open set $U \subseteq R^n$. If $f$ is directionally differentiable at $\boldsymbol{\xi} \in U$, then*

$$\lim_{\boldsymbol{x}\to\boldsymbol{\xi}} \frac{\big|f(\boldsymbol{x}) - f(\boldsymbol{\xi}) - f'(\boldsymbol{\xi},\boldsymbol{x}-\boldsymbol{\xi})\big|}{\|\boldsymbol{x}-\boldsymbol{\xi}\|} = 0, \tag{11.3}$$

where $\boldsymbol{x} \to \boldsymbol{\xi}$ is a sequence in $U$ converging to $\boldsymbol{\xi}$.

***Proof*** [318]. ■

Suppose that $f$ from Theorem 11.2 is directionally differentiable at $\boldsymbol{\xi}$. The classical definition of Fréchet differentiability is recovered by requiring that the mapping $f'(\boldsymbol{\xi},.)$ is linear. The existence of the limit (11.3) links the directional derivative concept to the Fréchet derivative in such that the key properties $(D_1)$-$(D_3)$ are preserved in some way.[2] Property $(D_1)$ of first order approximation directly follows from Theorem 11.2 using the existence of the limit (11.3). Referring to property $(D_2)$, the approximation function $g(\boldsymbol{x}) = f(\boldsymbol{\xi}) + f'(\boldsymbol{\xi},\boldsymbol{x}-\boldsymbol{\xi})$ of $f$ in a neighborhood of $\boldsymbol{\xi}$ is not necessarily affine, but still considerably simpler, because $f'(\boldsymbol{\xi},.)$ is positively homogeneous.[3] As we will see later, property $(D_3)$, the rules of calculus, applies to a certain extent.

So far, we have considered directional derivatives that locally characterize the change of a function along a given direction. Now we turn to a concept that generalizes the gradient for nonsmooth functions. Let $f : U \to \mathbb{R}$ be a locally

2. Properties $(D_1)$-$(D_3)$ are formulated on page 237.
3. A function $h : U \to \mathbb{R}$ is positively homogeneous at $\boldsymbol{\xi} \in U$, if $h(\alpha\boldsymbol{\xi}) = \alpha h(\boldsymbol{\xi})$ for all $\alpha \geq 0$.

Lipschitz continuous function defined on the open set $U \subseteq \mathbb{R}^n$. The set

$$\partial_B f(\boldsymbol{\xi}) = \left\{\boldsymbol{\gamma} \in \mathbb{R}^n \,:\, \exists (\boldsymbol{x}_i)_{i=1}^{\infty} \subseteq D_f \text{ s.t. } \boldsymbol{\gamma} = \lim_{\boldsymbol{x}_i \to \boldsymbol{x}} \nabla f(\boldsymbol{x}_i)\right\}$$

is called the *Bouligand-subdifferential* (*B-subdifferential*) of $f$ at $\boldsymbol{\xi}$. Note that the definition of a B-subdifferential makes sense, because a locally Lipschitz continuous function is Fréchet differentiable almost everywhere by Rademacher's Theorem 11.1. Hence, the gradients $\nabla f(\boldsymbol{x}_i)$ exist.

As opposed to the directional derivative, the concept of B-subdifferential is too weak to preserve the existence of calculus rules. A stronger and more promising concept for locally Lipschitz continuous functions is the *generalized gradient* based on *Clarke derivatives* [58].

### Clarke Derivatives and Generalized Gradients

Let $f : U \to \mathbb{R}$ be a locally Lipschitz continuous function defined on the open set $U \subseteq \mathbb{R}^n$. The *Clarke directional derivative* (*Clarke derivative*) of $f$ at $\boldsymbol{\xi} \in U$ in the direction $\boldsymbol{d} \in \mathbb{R}^n$ is defined by

$$f^{\circ}(\boldsymbol{\xi}, \boldsymbol{d}) = \limsup_{\substack{\alpha \downarrow 0 \\ \boldsymbol{x} \to \boldsymbol{\xi}}} \frac{f(\boldsymbol{x} + \alpha \boldsymbol{d}) - f(\boldsymbol{x})}{\alpha},$$

where $\alpha \downarrow 0$ and $\boldsymbol{x} \to \boldsymbol{\xi}$ are sequences such that $\boldsymbol{x} + \alpha \boldsymbol{d}$ is always in $U$.

The concept of the Clarke derivative is naturally linked to the concept of *strict differentiability* [58] rather than that of Fréchet differentiability. It differs from traditional definitions of directional derivatives in the following ways: first, it does not presuppose the existence of a limit; second, it involves the behavior in a neighborhood of $\boldsymbol{\xi}$; third, the base point $\boldsymbol{x}$ of the difference quotient varies.

The *generalized gradient* $\partial f(\boldsymbol{\xi})$ of $f$ at $\boldsymbol{\xi}$ is the set

$$\partial f(\boldsymbol{\xi}) = \left\{\boldsymbol{\gamma} \,:\, f^{\circ}(\boldsymbol{\xi}, \boldsymbol{d}) \geq \boldsymbol{\gamma}^{\mathsf{T}} \boldsymbol{d} \text{ for all } \boldsymbol{d} \in \mathbb{R}^n\right\}.$$

The elements of the generalized gradient $\partial f(\boldsymbol{\xi})$ are called *subgradients* of $f$ at $\boldsymbol{\xi}$. Theorem 11.3 summarizes some properties of the generalized gradient.

***Theorem 11.3***
*Let $f : U \to \mathbb{R}$ be a locally Lipschitz continuous function with Lipschitz constant $L$ defined on the open set $U \subseteq \mathbb{R}^n$. Then the following holds for $\boldsymbol{\xi} \in U$:*

1. $\partial f(\boldsymbol{\xi}) = con\big(\partial_B f(\boldsymbol{\xi})\big)$.
2. *$\partial f(\boldsymbol{\xi})$ is nonempty, compact, and convex.*
3. *$\|\boldsymbol{\gamma}\| \leq L$ for all $\boldsymbol{\gamma} \in \partial f(\boldsymbol{\xi})$.*
4. *For any direction $\boldsymbol{d} \in \mathbb{R}^n$, we have*

$$f^{\circ}(\boldsymbol{\xi}, \boldsymbol{d}) = \max\left\{\boldsymbol{\gamma}^{\mathsf{T}} \boldsymbol{d} \,:\, \boldsymbol{\gamma} \in \mathbb{R}^n\right\}.$$

5. *If $f$ is continuously Fréchet differentiable in a neighborhood of $\boldsymbol{\xi}$ then*

$$\partial_B f(\boldsymbol{\xi}) = \partial f(\boldsymbol{\xi}) = \{\nabla f(\boldsymbol{\xi})\}.$$

***Proof*** [58, 240]. ■

The first property relates the generalized gradient to the B-subdifferential. Convexification of the B-subdifferential yields stronger rules of calculus. Note that $\partial_B f(\boldsymbol{\xi})$ is nonempty and compact, but not convex in general. The fourth property indicates that the Clarke directional derivative and the generalized gradients lead to dual concepts that are obtainable from one another. Referring to the fifth property, it is necessary to require continuous differentiability rather than differentiability to prove equality.

### Regular Functions

*Regular functions* are an important class of functions for which we can derive strong theoretical results.

The Clarke derivative is a useful concept for nonsmooth analysis of locally Lipschitz functions, because it allows a systematic extension of generalized gradients for convex functions to the class of locally Lipschitz functions. For convex functions, the Clarke derivative reduces to the directional derivative.[4] But in general, equality $f^\circ(\boldsymbol{\xi}, \boldsymbol{d}) = f'(\boldsymbol{\xi}, \boldsymbol{d})$ is not satisfied, even if $f'(\boldsymbol{\xi}, \boldsymbol{d})$ exists. This equality, however, yields strong theoretical advantages. Therefore, it is reasonable to investigate functions for which the Clarke derivative and directional derivative are equivalent. These functions are called regular functions.

To present a definition of regular functions in formal terms, let $f : U \to \mathbb{R}$ be a locally Lipschitz continuous function on the open set $U \subseteq \mathbb{R}^n$. We say $f$ is (*subdifferentially*) *regular* at $\boldsymbol{\xi} \in U$ if the following conditions are satisfied:

1. $f$ is directionally differentiable at $\boldsymbol{\xi}$
2. $f^\circ(\boldsymbol{\xi}, \boldsymbol{d}) = f'(\boldsymbol{\xi}, \boldsymbol{d})$ for all $\boldsymbol{d} \in \mathbb{R}^n$.

The next result provides examples of regular functions and shows how we can construct complex regular functions from simple ones.

***Theorem 11.4***
*Let $f, f_1, \ldots, f_k : U \to \mathbb{R}$ be locally Lipschitz continuous functions defined on the open set $U \subseteq \mathbb{R}^n$, and let $\boldsymbol{\xi} \in U$ be a point.*

1. *Convex and continuously Fréchet differentiable functions are regular.*
2. *Let $w_1, \ldots, w_k \in \mathbb{R}^0_+$ be nonnegative values. If $f_1, \ldots, f_k$ are regular at $\boldsymbol{\xi}$, then the nonnegative linear combination $f = w_1 f_1 + \cdots + w_k f_k$ is regular at $\boldsymbol{\xi}$.*

4. Note that convex functions are locally Lipschitz continuous.

3. *Suppose that $f_1$ is regular at $\boldsymbol{\xi}$ and $f_2$ is continuously Fréchet differentiable at $\boldsymbol{\xi}$. Then $f_2 \circ f_1$ is regular at $\boldsymbol{\xi}$.*
4. *If $f_1, \ldots, f_k$ are regular at $\boldsymbol{\xi}$, then $f = \max\{f_1, \ldots, f_k\}$ is regular at $\boldsymbol{\xi}$.*

***Proof*** [58, 240]. ■

### Basic Calculus

We now proceed to show parts of the third key property ($D_3$) of a classical derivative, the existence of calculus rules. This property is indispensable for practical use, because it facilitates the calculation of $\partial f$ when $f$ is composed of simple functionals through linear combination, maximization, composition, and so on. Here we only focus on rules we actually need for our purposes.

The first result shows that the directional derivative and the generalized gradient are closed under linear combination.

***Theorem 11.5***
*Let $f_1, f_2 : U \to \mathbb{R}$ be locally Lipschitz continuous functions defined on the open set $U \subseteq \mathbb{R}^n$. Suppose that $f_1$ and $f_2$ are directionally differentiable at $\boldsymbol{\xi} \in U$. Then*

1. Scalar multiples: *$wf_1$ is directionally differentiable at $\boldsymbol{\xi}$ for all $w \in \mathbb{R}$ and*
   (a) $\partial(\alpha f_1)(\boldsymbol{\xi}) = \alpha \partial f_1(\boldsymbol{\xi})$
   (b) $(\alpha f_1)'(\boldsymbol{\xi}, \boldsymbol{d}) = \alpha f_1'(\boldsymbol{\xi}, \boldsymbol{d})$ *for all $\boldsymbol{d} \in \mathbb{R}^n$.*
2. Finite Sums: *$f_1 + f_2$ is directionally differentiable at $\boldsymbol{\xi}$ and*
   (a) $\partial(f_1 + f_2)(\boldsymbol{\xi}) \subseteq \partial f_1(\boldsymbol{\xi}) + \partial f_2(\boldsymbol{\xi})$
   (b) $(f_1 + f_2)'(\boldsymbol{\xi}, \boldsymbol{d}) = f_1'(\boldsymbol{\xi}, \boldsymbol{d}) + f_2'(\boldsymbol{\xi}, \boldsymbol{d})$ *for all $\boldsymbol{d} \in \mathbb{R}^n$.*

*If $f_1$ and $f_2$ are regular, equality holds in* (2.a).

***Proof*** [58, 318]. ■

Note that directional differentiability is not required to state (1.a) and (2.a). The next result establishes the chain rule for a special setting.

***Theorem 11.6 (Chain Rule)***
*Let $U \subseteq \mathbb{R}^n$ and $V \subseteq \mathbb{R}$ be open sets. If $f : U \to \mathbb{R}$ is locally Lipschitz continuous and directionally differentiable at $\boldsymbol{\xi} \in U$ and $g : V \to \mathbb{R}$ continuously Fréchet differentiable at $\boldsymbol{\zeta} = f(\boldsymbol{\xi})$, then the function $h = g \circ f$ is directionally differentiable at $\boldsymbol{\xi}$ and*

$$\partial h(\boldsymbol{\xi}) = \nabla g(\boldsymbol{\zeta}) \circ \partial f(\boldsymbol{\xi})$$
$$h'(\boldsymbol{\xi}, \boldsymbol{d}) = \nabla g(\boldsymbol{\zeta}) f'(\boldsymbol{\xi}, \boldsymbol{d})$$

*for all $\boldsymbol{d} \in \mathbb{R}^n$.*

***Proof*** [58, 240, 318]. ■

The meaning of $\nabla g(\boldsymbol{\zeta}) \circ \partial f(\boldsymbol{\xi})$ is that any element of $\partial h(\boldsymbol{\xi})$ can be represented as a composition of linear maps defined by $\nabla g(\boldsymbol{\zeta})$ and $\partial f(\boldsymbol{\xi})$. That is, for any element $\boldsymbol{\gamma}_h \in \partial h(\boldsymbol{\xi})$, there is an element $\boldsymbol{\gamma}_f \in \partial f(\boldsymbol{\xi})$ such that

$$\boldsymbol{\gamma}_h^{\mathsf{T}} \boldsymbol{x} = \nabla g(\boldsymbol{\zeta}) \cdot \left( \boldsymbol{\gamma}_f^{\mathsf{T}} \boldsymbol{x} \right)$$

for all $\boldsymbol{x} \in \mathbb{R}^n$.

### Necessary Optimality Condition

A necessary condition for a local optimum point $\boldsymbol{x}_*$ of a continuously Fréchet differentiable function is that the gradient vanishes at $\boldsymbol{x}_*$. If the necessary condition is not satisfied, the gradient provides valuable information about how to approach a local optimum. In this case, the opposite of the gradient is the direction of steepest descent. We want to formulate a similar property for the generalized gradient at a local optimum solution.

Theorem 11.7 presents a necessary optimality condition for locally Lipschitz continuous functions.

***Theorem 11.7***
*Let $f : U \to \mathbb{R}$ be locally Lipschitz continuous functions defined on the open set $U \subseteq \mathbb{R}^n$. If $f$ attains a local minimum or maximum at $\boldsymbol{\xi} \in U$, then*

$$\mathbf{0} \in \partial f(\boldsymbol{\xi}).$$

***Proof*** [250] ■

Suppose that $\mathbf{0}$ fails to belong to the generalized gradient $\partial f(\boldsymbol{\xi})$. Then by Theorem 11.7, the vector $\boldsymbol{\xi}$ cannot be a local optimal solution of $f$. We would like to find a nonzero vector $\boldsymbol{\delta}$ such that there is an $\overline{\alpha} > 0$ with

$$f(\boldsymbol{\xi} + \alpha \boldsymbol{\delta}) < f(\boldsymbol{\xi})$$

for all $0 < \alpha < \overline{\alpha}$. We call the vector $\boldsymbol{\delta}$ *descent direction* of $f$ at $\boldsymbol{x}$.

***Proposition 11.1***
*Let $f$ be locally Lipschitz on $\mathbb{R}^n$, and let $\boldsymbol{\xi} \in \mathbb{R}^n$ be a nonoptimal point. Suppose that $\boldsymbol{\gamma}_* \in \partial f(\boldsymbol{\xi})$ is a subgradient with minimal norm. Then the vector*

$$\boldsymbol{\delta} = -\boldsymbol{\gamma}_*$$

*is a descent direction.*

***Proof*** [58]. ■

## 11.3 Pointwise Maximizers

In this Section, we analyze pointwise maximizers of locally Lipschitz functions and relate the results to pointwise maximizers defined by the structural dot product.

Let $f_1, \ldots, f_k : U \to \mathbb{R}$ be locally Lipschitz continuous functions defined on the open subset $U \subseteq \mathbb{R}^n$. The *pointwise maximizer* of $f_1, \ldots, f_k$ is a function $f : U \to \mathbb{R}$ of the form

$$f(\boldsymbol{x}) = \max\big\{f_1(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x})\big\}.$$

We call the set

$$\operatorname{supp}(f) = \big\{f_1, \ldots, f_k\big\}$$

the *support* of $f$, and its elements $f_i \in \operatorname{supp}(f)$ the *support functions* of $f$. Let $\boldsymbol{\xi} \in U$ be a point. By $\operatorname{supp}(f, \boldsymbol{\xi})$ we denote the set of all support functions $f_i \in \operatorname{supp}(f)$ with $f_i(\boldsymbol{\xi}) = f(\boldsymbol{\xi})$.

Even if all support functions are Fréchet differentiable, the pointwise maximizer may not be. But what we can show is that the pointwise maximizer of locally Lipschitz continuous (continuously Fréchet differentiable) support functions is locally Lipschitz continuous (regular).

***Theorem 11.8***
*Let $U \subseteq \mathbb{R}^n$ be a nonempty open set, and let $f : U \to \mathbb{R}$ be a pointwise maximizer with finite support $\operatorname{supp}(f) = \{f_1, \ldots, f_k\}$. Then*

1. *if $f_1, \ldots, f_k$ are locally Lipschitz, $f$ is locally Lipschitz.*
2. *if $f_1, \ldots, f_k$ are continuously differentiable, $f$ is regular.*

***Proof*** [240]. ■

An immediate consequence of Rademacher's Theorem 11.1 and Theorem 11.8 is that pointwise maximizers of locally Lipschitz continuous support functions are Fréchet differentiable almost everywhere.

***Corollary 11.1***
*Let $f : U \to \mathbb{R}^m$ be a pointwise maximizer of locally Lipschitz continuous support functions. Then $\Omega_f = U \setminus \mathcal{D}_f$ has Lebesgue measure zero.*

For continuously Fréchet differentiable support functions of a pointwise maximizer, we can provide simple characterizations of the Clarke derivative and the generalized gradient.

***Proposition 11.2***
*Let $U \subseteq \mathbb{R}^n$ be a nonempty open set, and let $f : U \to \mathbb{R}$ be a pointwise maximizer with finite support $\operatorname{supp}(f) = \{f_1, \ldots, f_k\}$. Suppose that $f_1, \ldots, f_k$ are continuously*

*Fréchet differentiable at* $\boldsymbol{\xi} \in U$. *Then*

1. $f^\circ(\boldsymbol{\xi}, \boldsymbol{d}) = \max\left\{\nabla f_i(\boldsymbol{\xi})^\mathsf{T}\boldsymbol{d} \,:\, f_i \in \operatorname{supp}(f, \boldsymbol{\xi})\right\}$
2. $\partial f(\boldsymbol{\xi}) = \operatorname{con}\left\{\nabla f_i(\boldsymbol{\xi}) \,:\, f_i \in \operatorname{supp}(f, \boldsymbol{\xi})\right\}$

***Proof*** [240]. ■

Note that the Clarke derivative of the pointwise maximizer in Proposition 11.2 is equal to the directional derivative, because the pointwise maximizer of continuously Fréchet differentiable supports is regular according to Theorem 11.4.

### Parameterized Structural Dot Products

Consider the space $\mathbb{G}_\mathcal{A}$ of labeled graphs of bounded order $n$ with attributes from $\mathcal{A}$. Suppose that $W$ is a complex attributed graph from $\mathbb{G}_\mathcal{A}$. As shown in Section 10.3, the parameterized structural dot product

$$S_W : \mathbb{G}_\mathcal{A} \to \mathbb{R}, \quad X \mapsto W \bullet X$$

is a pointwise maximizer with support

$$\operatorname{supp}\left(S_W\right) = \left\{L_{\boldsymbol{P}} \,:\, L_{\boldsymbol{P}}(X) = \boldsymbol{W_P} \bullet \boldsymbol{X},\, \boldsymbol{P} \in \Pi^n\right\},$$

where $\boldsymbol{W_P} = \boldsymbol{P}^\mathsf{T}\boldsymbol{W}\boldsymbol{P}$, $\boldsymbol{W}$, and $\boldsymbol{X}$ are the adjacency matrices of $W_{\boldsymbol{P}}$, $W$, and $X$. Each support function $L_{\boldsymbol{P}} \in \operatorname{supp}(f)$ is a linear map. As shown in Section 10.4, $L_{\boldsymbol{P}}$ is continuously Fréchet differentiable with gradient $\nabla L_{\boldsymbol{P}}(X) = \boldsymbol{W_P}$.

Applying the results of Section 11.3 to the pointwise maximizer $S_W$ results in a number of properties:

1. Since the support functions of $S_W$ are continuously Fréchet differentiable, the pointwise maximizer $S_W$ is regular and therefore locally Lipschitz continuous. Hence, by Rademacher's Theorem 11.1, the parameterized structural dot product $S_W$ is Fréchet differentiable almost everywhere. At differentiable points $X$, there exists a unique linear map $L_{\boldsymbol{P}}$ with $S_W(X) = L_{\boldsymbol{P}}(X)$. Hence, the gradient of $S_W$ at $X$ is of the form

$$\nabla S_W(X) = \nabla L_{\boldsymbol{P}}(X) = \boldsymbol{W_P}.$$

2. From Proposition 11.2, property (1), it follows that

$$\begin{aligned} S_W^\circ(X, D) &= S_W'(X, D) \\ &= \max\left\{\boldsymbol{W_P} \bullet \boldsymbol{D} \,:\, \boldsymbol{P} \in \Pi^n \text{ with } L_{\boldsymbol{P}} \in \operatorname{supp}\left(S_W, X\right)\right\}, \end{aligned}$$

where $D \in \mathbb{G}_\mathcal{A}$ is an attributed graph with adjacency matrix $\boldsymbol{D}$, called *direction graph*.

3. According to Proposition 11.2, property (2), the generalized gradient of $S_W$

at $X$ is of the form

$$\begin{aligned}\partial S_W(X) &= \operatorname{con}\big\{\nabla L_{\boldsymbol{P}}(X) \,:\, L_{\boldsymbol{P}} \in \operatorname{supp}\big(S_W, X\big)\big\} \\ &= \operatorname{con}\big\{\boldsymbol{W}_{\boldsymbol{P}} \,:\, \boldsymbol{P} \in \Pi^n \text{ with } L_{\boldsymbol{P}} \in \operatorname{supp}\big(S_W, X\big)\big\}.\end{aligned}$$

4. Let $f : \mathbb{R} \to \mathbb{R}$ be a continuously Fréchet differentiable function. Then the function $g = f \circ S_W$ is regular and we may apply the chain rule:

(a) The Clarke directional derivative of $g$ is of the form

$$g^\circ(X, D) = g'(X, D) = f'(X \bullet W) S_W^\circ(X, D) = f'(X \bullet W) \cdot \big(\boldsymbol{W}^* \bullet \boldsymbol{D}\big),$$

where

$$\boldsymbol{W}^* = \max\big\{\boldsymbol{W}_{\boldsymbol{P}} \bullet \boldsymbol{D} \,:\, \boldsymbol{P} \in \Pi^n \text{ with } L_{\boldsymbol{P}} \in \operatorname{supp}\big(S_W, X\big)\big\}.$$

(b) For the generalized gradient of $g$ at $X$ we have

$$\partial g(X) = f'(X \bullet W) \partial S_W(X).$$

Note that $f'(X \bullet W)$ is the classical Fréchet derivative of $f$ at $X \bullet W$.

## 11.4 Nonsmooth Optimization

In this section, we address the issue of optimizing nonsmooth functions from the perspective of minimizing error functions of neural learning machines.

Recall from Section 10.5 that our goal is to solve the following optimization problem

$$\begin{array}{ll} \text{minimize} & \widehat{f}(X) = \sum_{q=1}^{p} \widehat{f}_i(X) \\ \text{subject to} & X \in \big[\mathbb{G}_{\mathcal{A}}\big], \end{array} \tag{11.4}$$

where $\widehat{f}_q : \big[\mathbb{G}_{\mathcal{A}}\big] \to \mathbb{R}$ are the component functions of $\widehat{f}$. To solve (11.4), we changed the metric space and considered the following equivalent formulation

$$\begin{array}{ll} \text{minimize} & f(X) = \sum_{q=1}^{p} f_i(X) \\ \text{subject to} & X \in \mathbb{G}_{\mathcal{A}}, \end{array} \tag{11.5}$$

where $f$ is the unique structural function with $f(X) = \hat{f}([X])$. In Section 10.5, we assumed that the component functions $f_q : \mathbb{G}_{\mathcal{A}} \to \mathbb{R}$ are differentiable. Now we focus on problem (11.5), where the component functions $f_q$ are supposed to be regular functions. Then the objective $f$ is also regular according to Theorem 11.4. Since $\mathbb{G}_{\mathcal{A}}$ is isomorphic to a Euclidean vector space, we may identify a labeled graph with a vector. Hence, the notion of regularity for structural functions makes sense and the whole theory developed in this chapter is applicable to problem (11.5).

Note that regularity is an appropriate assumption, because the parameterized

structural dot product is a regular function and the composition of a smooth function with a regular function is also regular. Hence, error functions of neural learning machines that are smooth for feature vectors are regular for attributed graphs.

In Section 10.5, we presented a basic iterative procedure to minimize smooth objective functions of (11.4). The situation is completely different for nonsmooth objective functions. Nonsmoothness of the objective makes optimization complicated and additional effort is necessary to construct well-behaved descent procedures. The hardest and most challenging problem is direction finding, because the direction opposite to a subgradient is not necessarily a direction of descent. As a consequence, classical line search procedures can no longer be applied unmodified in nonsmooth optimization.

Existing iterative algorithms for nonsmooth optimization problems can be roughly categorized into *subgradient methods* and *bundle methods*. Both approaches assume that (a) the objective is locally Lipschitz continuous, (b) we can evaluate the objective value at each point, and (c) we can determine a subgradient at each point.

To prove constructive existence of structural neural learning machines, incremental subgradient methods turn out to be a good choice, because they are a simple extension of gradient descent methods. According to [13, 250], bundle methods are considered to be more practicable for functions defined on feature vectors. To which extent this empirical observation is valid for nonsmooth functions on attributed graph is an open problem. In the context of structural pattern recognition, an answer to this problem falls into the realm of finding the most efficient learning procedure for structures. Since this issue exceeds the scope of this thesis, we only consider subgradient methods and refer to [250] for an introductory overview of bundle methods.

### Subgradient Methods

Subgradient methods can be viewed as a straightforward generalization of gradient descent, where the gradient $\nabla f(X_t)$ of an objective $f$ is replaced by an arbitrary subgradient $D_t \in \partial f(X_t)$. Algorithm 11 describes a principal method for solving problem (11.4), where the objective $f$ is supposed to be nonsmooth. For detailed expositions on subgradient techniques, we refer to [26, 264, 289, 329].

**Algorithm 11** (Basic Subgradient Algorithm)

**Initialization:**

set $X_0 \in \mathbb{G}_{\mathcal{A}}$ and $t = 0$

**Procedure:**

**repeat**

determine $D_t \in \partial f(X_t)$ (*direction finding*)

choose $\eta_t > 0$ (*line search*)

set $X_{t+1} = X_t + \eta_t D_t$ and $t = t + 1$ (*updating*)

**until** some criterion is satisfied (*termination criterion*)

**Output:** $\boldsymbol{x}_t$

The subgradient algorithm basically operates in the same way as the gradient descent method presented in Algorithm 9 for smooth functions. By Rademacher's Theorem 11.1, we may assume that in almost all iteration steps, the objective $f$ is Fréchet differentiable at the current point $X_t$. Hence, the generalized gradient of $f$ at $X_t$ coincides with the gradient $\nabla f(X_t)$.[5] Direction finding is therefore identical to gradient descent methods in almost all cases. At exceptional points, $f$ is nonsmooth. In these cases, the subgradient method selects an arbitrary subgradient from the generalized gradient $\partial f(X_t)$. For example, suppose that the objective $f$ is the composition $g \circ S_W$ of a smooth function $g : \mathbb{R} \to \mathbb{R}$ and the parameterized structural dot product $S_W$. Then direction finding selects a gradient $\nabla L_{\boldsymbol{P}}(X_t) = \boldsymbol{W_P}$ from a linear map $L_{\boldsymbol{P}} \in \operatorname{supp}\left(S_W, X_t\right)$ and constructs a direction according to

$$D_t = g'(W \bullet X_t) W_{\boldsymbol{P}},$$

where the Fréchet derivative $g'(W \bullet X_t)$ of $g$ at $W \bullet X_t$ is a well-defined scalar.

With a similar argumentation as for smooth objectives, Algorithm 11 is independent of the labeling of the vertices and aims at minimizing the objective function $\widehat{f}$ of the orignal problem (11.4) in the same way as the objective $f$ for the relaxed problem (11.5).

The simple idea of replacing the gradient by an arbitrary subgradient has some limitations:

1. At nonsmooth points, the direction opposite of an arbitrary subgradient is not always a direction of descent.
2. Standard termination criteria that exploit necessary conditions for local optima are inapplicable to subgradient methods, because an arbitrary subgradient may not contain any information about the optimality criterion $\mathbf{0} \in \partial f(\boldsymbol{x})$.

5. In strict terms, the generalized gradient of $f$ at $X$ is of the form $\partial f(X) = \{\nabla f(X)\}$ if $f$ is smooth at $X$.

3. Only weak convergence results of subgradient methods are known [329].

According to Proposition 11.1, all problems can be solved by choosing a subgradient $\gamma_t \in \partial f(X_t)$ with minimal length in the direction finding step of Algorithm 11. This requires knowledge of all subgradients at each iteration point $X_t$, which can be computationally intractable, particularly for functions on graphs whose evaluation is NP-complete. Although subgradient methods that ensure a decrease of the objective $f$ at each iteration step are computationally intractable, they provide a theoretically sound method for the analysis of structural neural learning machines.

In a practical setting, the above stated problems are solved as follows. Since the subgradient method is not a pure descent method, it is common to keep track of the best solution found so far. At each step we set

$$f_t^* = \begin{cases} f(X_0) & : \quad t = 0 \\ \min\left\{f(X_{t-1}), f(X_t)\right\} & : \quad t > 0 \end{cases}.$$

In addition, to enforce convergence and termination, we gradually decrease the step size $\eta_t$. The following result from Lemaréchal [225] provides a theoretically justified choice of step sizes.

***Lemma 11.1***
*Let $X_*$ be a solution of problem (11.4). Suppose that $X_t$ is not a solution and $D_t \in \partial f(X_t)$. Then*

$$\left\|X_{t+1} - X_*\right\| < \left\|X_t - X_*\right\|$$

*whenever*

$$0 < \eta_t < 2 \cdot \frac{f(X_t) - f(X_*)}{\|D_t\|}.$$

*Thus, at each iteration we have*

$$\left\|X_0 - X_t\right\| \leq \sum_{i=0}^{t} \eta_i.$$

***Proof*** [225], p. 544 and p. 545. ■

To ensure global convergence, Lemma 11.1 implies choosing the step sizes $\eta_t$ according to

$$\eta_t \downarrow 0 \text{ such that } \sum_{t=0}^{\infty} \eta_t = \infty.$$

### Incremental Subgradient Methods

As incremental gradient descent procedures, the incremental subgradient method updates $X_t$ incrementally through a cycle of $p$ sub-iterations, where the $q$-th sub-

iteration is a subgradient step for the component function $f_q$. It has been shown that incremental subgradient methods exhibit a behavior similar to incremental gradient descent [206, 273, 338].

Algorithm 12 presents a basic form of an incremental subgradient method for structural functions in line with the incremental gradient procedure described in Section 10.

**Algorithm 12** (Incremental Subgradient Algorithm)

**Initialization:**

set $X_0 \in \mathbb{G}_{\mathcal{A}}$ and $t = 0$

**Procedure:**

**repeat**

set $\widetilde{X}_{t,1} = X_t$

**for** $q \in [1:p]$

choose $D_{t,q} \in \partial f_q(\widetilde{X}_{t,q})$ (*direction finding*)

choose $\eta_{t,q} > 0$ (*line search*)

set $\widetilde{X}_{t,q+1} = \widetilde{X}_{t,q} + \eta_{t,q} D_{t,q}$ (*incremental-updating*)

**end for**

set $X_{t+1} = \widetilde{X}_{t,p+1}$ and $t = t + 1$ (*updating*)

**until** some criterion is satisfied (*termination criterion*)

**Output:** $X_t$

## 11.5 Conclusion

In this chapter, we presented basic concepts and results from nonsmooth analysis to formulate subgradient methods for optimizing structural functions. We showed that compositions of smooth functions and the structural dot product are regular structural functions. For regular functions, the Fréchet derivative exists almost everywhere and the subgradient method is applicable.

This was the last of three chapters aimed at establishing the theoretical foundation for structural neural learning machines. The following chapters apply the theory to formulate supervised and unsupervised neural learning algorithms for attributed graphs.

# 12 Structural Single-Layer Networks

In the previous three chapters, we laid the theoretical groundwork for adaptive processing of attributed graphs. This chapter and the next propose supervised structural neural learning machines. The goal is to present the basic principles and mechanisms of structural neural learning machines.

In this chapter, we are concerned with single layer feedforward networks for attributed graphs. In Section 12.1, we propose a structural model of a neuron. Section 12.2 discusses the representational capabilities of structural units for classification problems. We show that a structural unit implements a structural linear discriminant function. In Section 12.3, we adopt incremental subgradient techniques to minimize empirical risk functionals. The treatment applies to classification and function approximation problems. Section 12.4 extends Rosenblatt's perceptron to attributed graphs. We present a structural version of the perceptron criterion, derive a learning rule, and prove a convergence theorem.

## 12.1 Model of a Structural Neuron

This section extends the classical model of a neuron for feature vectors to a structural model that is capable of processing attributed graphs.

In line with the presentation of standard neuronal models provided in Section 4.2, we identify the following components of a neuronal model for structures:

1. *Structural Unit.* A structural unit (s-unit) is a processing device that transforms attributed graphs to real values. An s-unit $i$ has an activation $u_i$ and an output $x_i$.
2. *Weight graph.* An s-unit $i$ is associated with a *weight graph* $W^i$ from the set $\mathbb{G}_{\mathcal{A}}$ of complex attributed graphs.[1]
3. *Activation function.* Given an input graph $X$ from $\mathcal{G}_{\mathcal{A}}$, the activation of s-unit

1. Note that the terms *weight graph* and *weighted graph* are different notions. The former is a complex graph with attributes from $\mathcal{A} = \mathbb{R}^d$, whereas the latter is a real graph with attributes from $\mathbb{R}$.

$i$ is of the form

$$u_i = W^i \bullet X + h^i,$$

where $h^i$ denotes the *bias* or *threshold*.

4. *Transfer function.* The output $x_i$ of s-unit $i$ is related to its activation by

$$x_i = g(u_i) = g\Big(W^i \bullet X + h^i\Big),$$

where $g$ is a non-decreasing transfer function.

A single layer feedforward network composed of s-units implements a functional relationship

$$H : \mathcal{G}_{\mathcal{A}} \to \mathbb{R}^m, \quad X \mapsto \sum_{i=1}^{m} g\Big(W^i \bullet X + h^i\Big). \tag{12.1}$$

We call networks that realize functions of the form (12.1) *structural networks* or *s-networks*.

The following notation integrates the bias into the weight graph. This is a common technique in the standard case to simplify the mathematical analysis of conventional neural networks. The same holds for s-networks.

***Notation 12.1***
*By $\widehat{W}$ we denote the extended weight graph with adjacency matrix*

$$\widehat{\boldsymbol{W}} = \begin{pmatrix} h & \mathbf{0}_n^\top \\ \mathbf{0}_n & \boldsymbol{W} \end{pmatrix}.$$

*Similarly, we extend any input graph $X$ to a graph $\widehat{X}$ by inserting an isolated vertex with weight 1 in such that the adjacency matrix of $\widehat{X}$ is of the form*

$$\widehat{\boldsymbol{X}} = \begin{pmatrix} 1 & \mathbf{0}_n^\top \\ \mathbf{0}_n & \boldsymbol{X} \end{pmatrix}.$$

*Then the structural dot product of the extended graphs is defined by*

$$\widehat{\boldsymbol{X}} \bullet \widehat{\boldsymbol{W}} = \max_{\boldsymbol{P} \in \Pi_1^{n+1}} \boldsymbol{P}^\top \widehat{\boldsymbol{X}} \boldsymbol{P} \bullet \widehat{\boldsymbol{W}},$$

*where $\Pi_k^{n+1}$ is the subset of all $(n+1)\times(n+1)$-permutation matrices $\boldsymbol{P} = (p_{ij})$ with $p_{kk} = 1$. Thus, the definition of the structural dot product of extended graphs slightly differs from its original definition in the sense that the first vertex of $\widehat{X}$ must not be permuted.*

In the remainder of this section we provide some remarks, which might clarify some issues.

1. Note that no confusion can arise between output $x_i$ of s-unit $i$ and elements

$\boldsymbol{x}_{ij}$ of the adjacency matrix $\boldsymbol{X} = (\boldsymbol{x}_{ij})$. The former has a single index referring to its s-unit and the latter is annotated with a pair of indices referring to an item of $X$.

2. An input graph is usually a real attributed graph from $\mathcal{G}_{\mathcal{A}}$ and therefore a proper graph in a a graph-theoretical sense. For weight graphs, we admit imaginary vertices and edges. This is necessary to obtain a *complete* metric space. Completeness of a metric space in turn is required to define concepts of a derivative for error criteria as functions of $W$. In supervised learning, it does not matter whether a weight graph $W$ has imaginary items or not, because $W$ is not intended to represent objects from the underlying problem domain. Thus, the proposed structural model of a neuron is feasible.

## 12.2 Structural Linear Discriminant Functions

In this section, we focus on classification problems. We show that single layer s-networks implement structural linear discriminant functions. We first consider the two-category case in Section 12.2.1 and then briefly sketch the multi-category case in Section 12.2.2.

### 12.2.1 The Two-Category Case

We begin by reviewing the geometrical interpretation of conventional units for feature vectors. For a more detailed treatment we refer to [32, 77]. Suppose that we are given an input pattern $\boldsymbol{x} \in \mathbb{R}^n$. The activation of a conventional unit can be written as

$$u(\boldsymbol{x}) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} + h,$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is the weight vector associated with that unit and $h \in \mathbb{R}$ denotes the bias. The equation $u(\boldsymbol{x}) = 0$ defines a hyperplane, which divides the input space in two decision regions, one for each category. The orientation of the hyperplane is determined by the weight vector $\boldsymbol{w}$, and its location by the bias $h$. The activation $\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} + h$ of the perceptron is proportional to the signed distance from $\boldsymbol{x}$ to the hyperplane.

The origin of a geometrical interpretation of conventional units are the geometrical properties of the dot product. Since the structural and the conventional dot product have some geometrical properties in common, s-units admit a similar geometrical interpretation. We show that an s-unit separates the input space by a decision surface composed of hyperplane segments. Each hyperplane segment is orthogonal to a labeled representative $W$ of the unlabeled weight graph $[W]$. The location of a hyperplane segment is determined by the bias $h$ and the sector of the labeled graph $W$ orthogonal to that segment. The activation $W \bullet X + h$ of the s-perceptron is proportional to the signed distance from $X$ to its designated hyper-

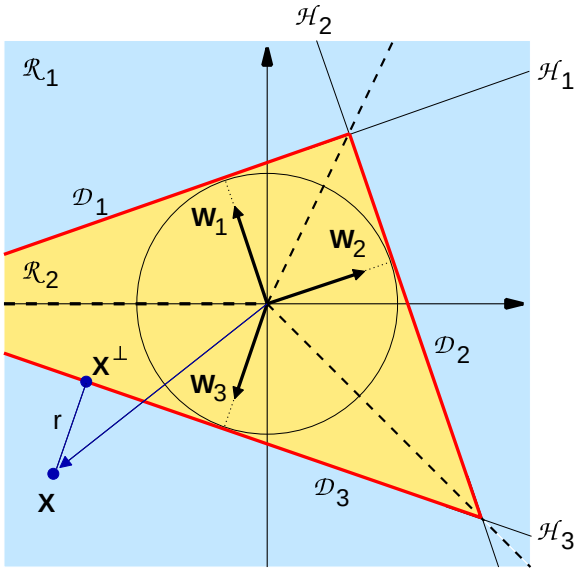



**Figure 12.1** Structural linear decision surface $\mathcal{D}_f$, where $f(X) = W \bullet X + h = 0$ separates the input space in two regions $\mathcal{R}_1$ with $f(X) > 0$ (light blue) and $\mathcal{R}_2$ with $f(X) \leq 0$ (gold). The decision surface $\mathcal{D}_f$ (red line) is composed of the hyperplane segments $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$. The sectors of the spokes representing the labeled graphs $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, and $\boldsymbol{W}_3$ are indicated by the bold-faced dashed lines. The adjacency matrix $\boldsymbol{X}$ of the input $X$ is closest to the spoke representing $\boldsymbol{W}_3$ and is therefore classified by the local discriminant function $f_3 = \boldsymbol{W}_3 \bullet \boldsymbol{X} + h$.

plane. Figure 12.1 presents a schematic example of the decision surface defined by an s-unit.

An s-unit determines a *discriminant function* of the form

$$f(X) = W \bullet X + h \tag{12.2}$$

such that an input graph $X$ is assigned to class $\mathcal{C}_1$ if $f(X) > 0$, and to class $\mathcal{C}_2$ if $f(X) \leq 0$. Note that $X$ can ordinarily be assigned to either class if $H(X) = 0$. Here, without loss of generality, we have chosen to assign $X$ to class $\mathcal{C}_2$.

We can express the discriminant by

$$f(X) = S_W(X) + f_h(X),$$

where $S_W(X) = W \bullet X$ is the parameterized structural dot product and $f_h(X) = h$ is a constant function. Since $S_W(X)$ and $f_h(X)$ are independent of the labeling of $X$, the discriminant $f$ is a structural function. From Section 11.3, it follows that $S_W(X)$ and therefore the discriminant function $f$ is piecewise linear. Thus, the set

$$\mathcal{D}_f = \big\{X \in \mathcal{X} \,:\, f(X) = 0\big\}$$

is composed of hyperplane segments. The *decision surface* $\mathcal{D}_f$ defined by $f$ separates

the input space $\mathcal{X}$ in two *decision regions*

$$\mathcal{R}_1 = \big\{X \in \mathcal{X} \,:\, f(X) > 0\big\}$$
$$\mathcal{R}_2 = \big\{X \in \mathcal{X} \,:\, f(X) \leq 0\big\}.$$

Graphs located in decision region $\mathcal{R}_1$ are assigned to class $\mathcal{C}_1$ and graphs from $\mathcal{R}_2$ are assigned to class $\mathcal{C}_2$.

To characterize the decision surface $\mathcal{D}_f$, we consider the vector representation $\mathcal{V}_{[W]}$ of the unlabeled graph $[W]$. Each vector $\boldsymbol{w} \in \mathcal{V}_{[W]}$ is a *spoke* of the ball $\mathcal{B}_f = \mathcal{B}_F(0, h/\|W\|)$ with center 0 and radius $h/\|W\|$. We call the ball $\mathcal{B}_f$ *decision wheel* of the structural discriminant $f$. Each spoke $\boldsymbol{w}$ is orthogonal to a hyperplane $H_{\boldsymbol{w}}$ tangent to the decision wheel. The decision surface $\mathcal{D}_f$ is then of the form

$$\mathcal{D}_f = \bigcup_{\boldsymbol{w} \in \mathcal{V}_{[W]}} H_{\boldsymbol{w}} \cap S_{\boldsymbol{w}},$$

where $S_{\boldsymbol{w}}$ is the sector of $\boldsymbol{w}$. Each hyperplane segment $\mathcal{D}_{\boldsymbol{w}} = H_{\boldsymbol{w}} \cap S_{\boldsymbol{w}}$ divides a sector in the two regions $\mathcal{R}_1 \cap S_{\boldsymbol{w}}$ and $\mathcal{R}_2 \cap S_{\boldsymbol{w}}$. Thus, the structural discriminant $f$ behaves locally like a linear discriminant.

Now suppose that $\boldsymbol{w}$ is the spoke closest to $X$. Then $X$ is located in sector $S_{\boldsymbol{w}}$ and we have

$$f(X) = W \bullet X + h = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} + h.$$

The following properties hold:

1. Within its sector $S_{\boldsymbol{w}}$, a spoke $\boldsymbol{w}$ points to the decision region $\mathcal{R}_1$. This follows from $f(X) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} + h > 0$ whenever $X \in \mathcal{R}_1$.

2. The discriminant function $f(X)$ gives an algebraic measure of the distance from $\boldsymbol{x}$ to the hyperplane $H_{\boldsymbol{w}}$. The signed distance from $\boldsymbol{x}$ to the hyperplane $H_{\boldsymbol{w}}$ is given by

$$r_{\boldsymbol{w}} = \frac{f(X)}{\|W\|}.$$

The distance $r$ is positive if $X$ lies in $\mathcal{R}_1$ and non-positive otherwise.

3. Let $\hat{r}_{\boldsymbol{w}}$ be the signed distance from $\boldsymbol{x}$ to the hyperplane segment $\mathcal{D}_{\boldsymbol{w}}$. Then $|r_{\boldsymbol{w}}| \leq |\hat{r}_{\boldsymbol{w}}|$, where inequality holds whenever the orthogonal projection $X^{\perp}$ of $X$ onto $H_{\boldsymbol{w}}$ is not in $\mathcal{D}_{\boldsymbol{w}}$. Clearly, we have $r_{\boldsymbol{w}} = 0$ if, and only if, $\hat{r}_{\boldsymbol{w}} = 0$, because $\mathcal{D}_{\boldsymbol{w}} = S_{\boldsymbol{w}} \cap H_{\boldsymbol{w}}$.

As a structural function, $f(X)$ is independent of the labeling of $X$. Relabeling the vertices may rotate $X$ to another sector by preserving all geometrical properties without leaving the decision region.

To summarize, the discriminant function defined in (12.2) divides the input space by a decision surface composed of hyperplane segments. The segments are tangent to the decision wheel of $f$ around the origin with a radius determined by the bias

$h$. The orientation of the segments are determined by the spokes of the decision wheel. The value $r = f(X)/\|W\|$ is the signed distance from $X$ to its designated hyperplane. The absolute value $|r|$ is a lower bound to the absolute distance from $X$ to its designated hyperplane segment. To classify an input $X$, we first identify the sector of a spoke closest to $X$ and then determine the local decision region into which $X$ falls.

***Limitation***

The representational capabilities of an s-unit are limited to special piecewise linear decision surfaces. The hyperplane segments of a decision surface are related via relabeling of spokes. Hence, an s-unit cannot represent an arbitrary piecewise linear decision surface. This issue is closely related to the important concepts of *capacity* and *VC dimension* from statistical learning theory. Despite its importance, it is beyond the scope of this work to cover all aspects of structural neural learning machines.

### 12.2.2 The Multi-Category Case

Structural linear discriminants can be easily extended to multi-category problems in more than one way [77]. Here, we consider a simple approach to illustrate how standard techniques for the multi-category case can be combined with structural discriminants.

Suppose that we are given $m$ classes $\mathcal{C}_1, \ldots, \mathcal{C}_m$. We consider a single layer network composed of $m$ threshold s-units, where s-unit $i$ responds to class $\mathcal{C}_i$. Each s-unit $i$ determines a structural linear discriminant function

$$f_i(X) = W^i \bullet X + h^i.$$

A new input graph $X$ is assigned to class $\mathcal{C}_i$ if $f_i(X) > f_j(X)$ for all $j \neq i$. The decision rule divides the input space $\mathcal{X}$ into $m$ decision regions $\mathcal{R}_i$, with $f_i(X)$ being the largest discriminant if $X$ lies in $\mathcal{R}_i$. If the decision regions $\mathcal{R}_i$ and $\mathcal{R}_j$ are adjacent, we have

$$\left(W^i \bullet X - W^j \bullet X\right) + (h^i - h^j) = 0.$$

Since the structural dot product is not bilinear, we cannot simply factor out the term $X$. But what we can do is consider the sectors into which $X$ falls. Suppose that $\tilde{\boldsymbol{w}}_i \in \mathcal{V}_{[W^i]}$ and $\tilde{\boldsymbol{w}}_j \in \mathcal{V}_{[W^j]}$ are spokes closest to $X$. Then $X$ is in the intersection

$$S_{ij} = S_{\tilde{\boldsymbol{w}}_i} \cap S_{\tilde{\boldsymbol{w}}_j}.$$

From

$$\left(\tilde{\boldsymbol{w}}_i^{\mathsf{T}} \boldsymbol{x} - \tilde{\boldsymbol{w}}_j^{\mathsf{T}} \boldsymbol{x}\right) + (h^i - h^j) = (\tilde{\boldsymbol{w}}_i - \tilde{\boldsymbol{w}}_j)^{\mathsf{T}} \boldsymbol{x} + (h^i - h^j) = 0,$$

it follows that the vector $\tilde{\boldsymbol{w}}_{ij} = \tilde{\boldsymbol{w}}_i - \tilde{\boldsymbol{w}}_j$ and the difference $h^i - h^j$ determine a hyperplane $H_{ij}$ dividing the intersection $S_{ij}$ into local decision regions for class $\mathcal{C}_i$ and $\mathcal{C}_j$. The signed distance from $X$ to $H_{ij}$ is given by

$$\frac{f_i(X) - f_j(X)}{\|\tilde{\boldsymbol{w}}_{ij}\|}.$$

It is important to note that not the spokes, but rather their differences are important. The difference $\tilde{\boldsymbol{w}}_{ij}$ determines a graph $\widetilde{W}^{ij}$ with

$$\tilde{\boldsymbol{w}}_{ij}^{\mathsf{T}}\boldsymbol{x} \leq \widetilde{W}^{ij} \bullet X.$$

Hence, $\tilde{\boldsymbol{w}}_{ij}$ needs not be a spoke closest to $X$.

## 12.3 Structural Learning

So far in this chapter we have discussed what s-units are and what they represent. The remainder of this chapter is concerned with the training of single layer s-networks to learn an unknown functional relationship from a given sample of labeled data. This section serves to provide the general setting of supervised structural learning with single layer s-networks.

In the following, we integrate the bias into the weight graph as described in Notation 12.1. To unclutter the notation, we drop the $\widehat{\ }$-accent and simple write, by abuse of notation, $W$ instead of $\widehat{W}$, $X$ instead of $\widehat{X}$, and $\Pi^n$ instead of $\Pi_1^{n+1}$.

Now suppose that we are given a training sample

$$\mathcal{Z} = \big\{(X_1, y_1), \ldots, (X_p, y_p)\big\} \subseteq \mathcal{X} \times \mathcal{Y}$$

consisting of $p$ training graphs $X_i \in \mathcal{X} \subseteq \mathcal{G}_{\mathcal{A}}$ drawn from some input space $\mathcal{X}$ together with corresponding output values $y_i \in \mathcal{Y} \subseteq \mathbb{R}$. Without loss of generality, we assume that the outputs are real-valued. Extension to vector-valued outputs is straightforward.

Following the inductive principle of empirical risk minimization, the goal is to find a weight graph $W \in \mathbb{G}_{\mathcal{A}}$ such that the hypothesis $H_W(X) = g(W \bullet X)$ minimizes an error function (empirical risk functional) of the form

$$E : \mathbb{G}_{\mathcal{A}} \to \mathbb{R}, \quad W \mapsto E(W) = \sum_{i=1}^{p} E_i(W) \tag{12.3}$$

constrained over the training sample $\mathcal{Z}$. The quantity

$$E_i(W) = l(x_i, y_i)$$

is the error (loss), which measures the discrepancy between the actual output $x_i = g(W \bullet X_i)$ predicted by hypothesis $H_W$ and the *correct* label $y_i$ of the $i$-

th training graph $X_i$. To emphasize functional dependence of the the loss $l(x_i, y_i)$ on $W$ via $x_i = g(W \bullet X_i)$, we use the more intuitive notation $E_i(W)$.

We now derive an incremental subgradient algorithm to minimize (12.3). Since the structural function $S_X(W) = W \bullet X$ is nonsmooth, we cannot expect that the error criterion

$$E(W) = E\big(S_X(W)\big)$$

is smooth. Hence, standard techniques that minimize smooth versions of (12.3) are inapplicable. But under the assumption that $E(W)$ is a regular function, we may adopt incremental subgradient methods as described by Algorithm 12 in Section 11.4.

A precondition to ensure that $E(W)$ is a regular function is that the transfer function $g(u)$ and the loss $l(x, y)$ are smooth in $u$ and $x$, respectively, for all $u, x \in \mathbb{R}$. Here, $u = W \bullet X$ refers to the activation and $x = g(u)$ to the output of an s-unit. Thus, by the chain rule of elementary calculus, $E$ is smooth as a function of $u = S_X(W)$. According to Theorem 11.4, the composition $E(W) = E\big(S_X(W)\big)$ of a smooth function $E$ in $u$ and a regular function $S_X$ in $W$ is regular. From Theorem 11.6, it follows that the generalized gradient $\partial E(W)$ of $E$ at $W$ is of the form

$$\partial E(W) = E'(S_X(W))\partial S_X(W).$$

We introduce a notation

$$\delta = E'(S_X(W)), \tag{12.4}$$

where $\delta$ is the *sensitivity* of an s-unit [77]. Evaluation of $\delta$ is straightforward. From definition (12.4) we have

$$\delta = g'(u) \cdot E'(x).$$

It remains to choose a subgradient from the generalized gradient $\partial S_X(W)$. For any optimal relabeling $\pi$ of $X$ to $W$, the relabeled graph $X^\pi$ is a subgradient from $\partial S_X(W)$. Combining the results yields

$$\delta X^\pi = g'(u) \cdot E'(x) \cdot X^\pi \in \partial E(W).$$

Hence, $\delta X^\pi$ is a subgradient from $\partial E(W)$ and we obtain the subgradient learning rule

$$W \leftarrow W - \eta \cdot \delta \cdot X^\pi. \tag{12.5}$$

Plugging the subgradient learning rule (12.5) into the general Incremental Subgradient Algorithm 12 ( Section 11.4) yields a learning procedure to minimize the error criterion (12.3). The procedure is shown in Algorithm 13.

**Algorithm 13** (Incremental Subgradient Learning)

**Input:**
$\mathcal{Z}$ – training sample $\{(X_1, y_1), \ldots, (X_p, y_p)\} \in \mathcal{X} \times \mathcal{Y}$
**Initialization:**
set $W_0 \in \mathbb{G}_{\mathcal{A}}$
**Procedure:**
**repeat**
**for** all $i \in [1{:}p]$ **do**
compute sensitivity $\delta$
choose learning rate $\eta$
choose optimal relabeling $\pi \in \mathcal{R}_{X_i,W}$
update $W = W - \eta\,\delta X_i^{\pi}$
**end for**
**until** some criterion is satisfied
**Output:** $H_W : X \mapsto W \bullet X$

Note that a regular error criterion is locally Lipschitz continuous. Hence, $E$ is smooth almost everywhere and the chosen subgradient coincides with the gradient of $E$.

## 12.4 Structural Perceptrons

This section extends Rosenblatt's perceptron from the domain of feature vectors to the domain of attributed graphs. Rosenblatt [302, 303] studied a single threshold unit with adjustable weights and bias for two-category classification problems. To appropriately adjust the weights and bias, he proposed a learning algorithm and showed that it converges to a correct solution, provided that the feature vectors are drawn from two *linearly separable* classes (i.e. the decision boundary of both classes forms a hyperplane). The proof of convergence is known as the *perceptron convergence theorem*. To extend Rosenblatt's perceptron to attributed graphs, we adapt the learning algorithm to the structural domain. In addition, we prove a weaker version of the perceptron convergence theorem for structurally separable problems.

An *s-perceptron* is a threshold s-unit with a threshold transfer function of the form

$$\Theta(u_i) = \begin{cases} +1 & : \text{ if } u_i > 0 \\ -1 & : \text{ if } u_i \leq 0 \end{cases} .$$

For mathematical convenience, we consider the bipolar threshold function with outputs from $\{\pm 1\}$ instead of the binary threshold function with outputs from

$\{0,1\}$.[2]

Suppose now that we are given a training sample

$$\mathcal{Z} = \big\{(X_1, y_1), \ldots, (X_p, y_p)\big\} \subseteq \mathcal{X} \times \{\pm 1\}$$

consisting of $p$ training graphs $X_i \in \mathcal{X} \subseteq \mathcal{G}_\mathcal{A}$ drawn from some input space $\mathcal{X}$ together with corresponding labels $y_i \in \{\pm 1\}$. Label $y = -1$ encodes class $\mathcal{C}_1$, and label $y = +1$ class $\mathcal{C}_2$. A training sample $\mathcal{Z}$ is called *structural linearly separable* if there exists a discriminant function $H_{W^*}$ with

$$H_{W^*}(X) = \Theta\big(W^* \bullet X\big) = y$$

for all training examples $(X, y) \in \mathcal{Z}$. The weight graph $W^*$ is called *separating graph* or *solution graph*.

### 12.4.1 The Structural Perceptron Criterion

The goal is to find a weight graph $W$ so that an s-perceptron minimizes the following *structural perceptron criterion*

$$E^{perc}(W) = \sum_{X_i \in \mathcal{E}(W)} -y_i\big(W \bullet X_i\big), \tag{12.6}$$

where $\mathcal{E}(W)$ is the set of training graphs $X_i$ misclassified by the hypothesis $H_W(X_i) = \Theta(W \bullet X_i)$. Since $y_i\big(W \bullet X_i\big) < 0$ if $X_i$ is misclassified, $E^{perc}$ is nonnegative, being zero only if $W$ is a separating graph or if $X$ is on the decision boundary.

The error criterion $E^{perc}(W)$ is continuous and piecewise linear as a function of $u = S_X(W) = W \bullet X$ with discontinuities in its gradient. Hence, $E^{perc}(W)$ is locally Lipschitz continuous, since $E^{perc}(u)$ and $S_X(W)$ are both locally Lipschitz continuous.

The standard perceptron criterion is proportional to the sum of the absolute distances from the misclassified training patterns to the decision hyperplane. The situation is different for the structural perceptron criterion (12.6). Let $\hat{r}(W)$ denote the sum of the absolute distances from the misclassified training graphs to the decision surface. According to the discussion in Section 12.2.1, we have

$$r(W) = \frac{E^{perc}(W)}{\|W\|} \leq \hat{r}(W).$$

This poses the question as to whether there exists a weight graph $W$ with

$$0 = r(W) < \hat{r}(W).$$

2. Note that both threshold functions are obtainable from one another via $2\Theta(u_i) - 1$.

With a similar argumentation as in Section 12.2.1, page 257, item 3, we have

$$r(W) = 0 \;\Leftrightarrow\; \hat{r}(W) = 0.$$

Thus, extension of the standard perceptron criterion to its structural counterpart is feasible.

***Remark 12.2***
*In general, the functions*

$$-S_W(X) = -\big(W \bullet X\big)$$
$$S_{-W}(X) = (-W) \bullet X = W \bullet (-X)$$

*are unequal. The function $-S_W$ is a pointwise minimizer, which first maximizes the matrix dot product $\boldsymbol{W_P} \bullet \boldsymbol{X}$ over all $\boldsymbol{P} \in \Pi^n$ and then takes the additive inverse of that maximum. The second function $S_{-W}$ is the usual pointwise maximizer of the support functions $(-\boldsymbol{W_P}) \bullet \boldsymbol{X}$ for all $\boldsymbol{P} \in \Pi^n$. Therefore, we cannot move the class label $y_i$ inside the structural dot product $W \bullet X_i$ to bring all training graphs $X_i$ into the same decision region. This normalization of training examples is commonly done for feature vectors to simplify the treatment.*

### 12.4.2 Minimizing the Structural Perceptron Criterion

We now want to present an incremental procedure to minimize the structural perceptron criterion (12.6). Since $E^{perc}(W)$ is locally Lipschitz continuous but not regular, we cannot inconsiderately adapt the Incremental Subgradient Learning Algorithm 13.

Suppose that example $(X_i, y_i) \in \mathcal{Z}$ is presented to the s-perceptron at iteration step $t$. Then an incremental learning procedure is concerned with the $i$-th component function $E_i^{perc} = -y_i(W \bullet X_i)$. Clearly, $E_i^{perc}$ is a regular function. But the function

$$f_i(W) = \begin{cases} E_i^{perc} & : \quad X_i \notin \mathcal{E}(W_t) \\ 0 & : \quad X_i \in \mathcal{E}(W_t) \end{cases}$$

is only regular at $W$ whenever $E_i^{perc}(W) \neq 0$. Thus, care must be taken at irregular points of $E_i^{perc}$.

Since the s-perceptron criterion is locally Lipschitz, it has a generalized gradient at any point from $\mathbb{G}_{\mathcal{A}}$. Hence, the following approach for the *s-perceptron learning rule* makes sense

$$W_{t+1} \leftarrow W_t - \eta_t G_t, \tag{12.7}$$

where $G_t \in \partial f_i(W)$ is a subgradient from the generalized gradient $\partial f_i(W)$. Hence, everything comes down to selecting an appropriate subgradient $G_t$. We distinguish the following cases:

1. $E_i^{perc}(W_t) > 0$: In this case, $X_i$ is misclassified and by definition of $f_i$, we have $f_i(W) = -y_i(W_t \bullet X_i)$. Let $\pi$ be an optimal relabeling of $X_i$ towards $W_t$. Then

$$\boldsymbol{G}_t = -y_i X_i^{\pi}$$

is a subgradient from $\partial f_i(W_t)$.

2. $E_i^{perc}(W_t) < 0$: In this case, $X_i$ is correctly classified and by definition of $f_i$, we have $f_i(W_t) = 0$. From continuity of $E_i^{perc}(W_t)$, it follows that $f_i(W) = 0$ for all $W$ from a sufficiently small $\rho$-neighborhood $\mathcal{B}_F(W_t, \rho)$ of $W_t$. Hence, $f_i$ is constant on $\mathcal{B}_F(W_t, \rho)$ and therefore smooth at $W_t$. In addition, the generalized gradient $\partial f_i(W_t)$ coincides with the gradient $\nabla f_i(W_t) = 0$, where 0 denotes the null graph. Thus,

$$G_t = 0$$

is a subgradient from $\partial f_i(W_t)$.

3. $E_i^{perc}(W_t) = 0$: We distinguish two further cases.

   (a) $X_i \in \mathcal{E}(W_t)$: In this case, $X_i$ lies on the decision surface but is misclassified. This occurs when $X_i$ is from class $\mathcal{C}_1$. Our goal is to adjust $W_t$ such that $X_i$ is correctly classified. We have $f_i(W) = -y_i(W_t \bullet X_i) = 0$. From the first case, it follows that

$$G_t = -y_i X_i^{\pi}$$

   is a subgradient from $\partial f_i(W_t)$, where $\pi$ is an optimal relabeling of $X_i$ to $W_t$. Now let us check whether $X_i$ is correctly classified by the new hypothesis $H_{W_{t+1}}(X_i)$. Let $\boldsymbol{P}$ be the matrix representation of the optimal relabeling $\pi$ of $X_i$ to $W_t$, and let $\boldsymbol{Q}$ be an optimal rotation of $X_i$ towards $W_{t+1}$. We have

$$\begin{aligned} H_{W_{t+1}}(X_i) &= W_{t+1} \bullet X_i = \big(W_t - \eta_t G_t\big) \bullet X_i \\ &= \big(\boldsymbol{W}_t + \eta_t y_i \boldsymbol{P}^{\mathsf{T}} \boldsymbol{X}_i \boldsymbol{P}\big) \bullet \boldsymbol{Q}^{\mathsf{T}} \boldsymbol{X}_i \boldsymbol{Q}. \end{aligned}$$

   From the equation in the last line it is straightforward to show that $\boldsymbol{Q} = \boldsymbol{P}$ is an optimal rotation of $X_i$ towards $W_{t+1}$. Thus with $\boldsymbol{X}_{\boldsymbol{P}} = \boldsymbol{P}^{\mathsf{T}} \boldsymbol{X}_i \boldsymbol{P}$, we get

$$\begin{aligned} H_{W_{t+1}}(X_i) &= \big(\boldsymbol{W}_t + \eta_t y_i \boldsymbol{X}_{\boldsymbol{P}}\big) \bullet \boldsymbol{X}_{\boldsymbol{P}} \\ &= \boldsymbol{W}_t \bullet \boldsymbol{X}_{\boldsymbol{P}} + \eta_t y_i \boldsymbol{X}_{\boldsymbol{P}} \bullet \boldsymbol{X}_{\boldsymbol{P}} \\ &= H_{W_t}(X) + \eta_t y_i \boldsymbol{X}_{\boldsymbol{P}} \bullet \boldsymbol{X}_{\boldsymbol{P}}. \end{aligned}$$

   Since $X_i$ is from class $\mathcal{C}_1$, we have $y = 1$. From $\|X\|^2 > 0$ for $X \neq 0$ follows the desired improvement

$$H_{W_{t+1}}(X_i) > H_{W_t}(X_i) = 0.$$

   (b) $X_i \notin \mathcal{E}(W_t)$: In this case, $X_i$ lies on the decision surface, but is correctly

classified. This occurs when $X_i$ is from class $\mathcal{C}_2$. Since there is no mistake, no update is necessary. This can be derived by choosing

$$G_t = 0$$

as a subgradient from $\partial f_i(W_t)$. This is feasible, because $f_i(W + \alpha D)$ is zero along any direction graph $D$ pointing into region $\mathcal{R}_2$.

To summarize, the subgradient $G_t$ of the structural perceptron learning rule (12.7) is of the form

$$G_t = \begin{cases} -y_i \boldsymbol{X}_i^{\pi} & : \quad X_i \in \mathcal{E}(W_t) \\ 0 & : \quad X_i \notin \mathcal{E}(W_t) \end{cases},$$

where $\pi$ is an optimal relabeling of $X_i$ to $W_t$.

Since $G_t = 0$ whenever an input is correctly classified, the s-perceptron algorithm is a *mistake-driven* incremental subgradient method (see Algorithm 14). Given a misclassified example $(X, y)$, updating of the weight graph involves the following steps: first, select an optimal relabeling $\pi$ of $X$ to $W$; second, relabel $X$ to $X^{\pi}$; third, update the weight graph $W$ by adding $\eta\, y X^{\pi}$.

Figure 12.2 illustrates learning rule (12.7). In geometrical terms, adjusting the bias stretches or shrinks the decision wheel, and adjusting the weight graph shears the spokes. It is possible that updating the weight graph collapses two spokes to a single spoke. Similarly, collapsed spokes may be sheared apart by applying update rule (12.7).

---

**Algorithm 14** (Structural Perceptron Algorithm)

---

**Input:**

$\mathcal{Z}$ – training sample consisting of $p$ examples $(X_1, y_1), \ldots, (X_p, y_p) \in \mathcal{X} \times \{\pm 1\}$

**Initialization:**

set $W = 0$

**Procedure:**

**repeat**
  **for** all $i \in [1{:}p]$ **do**
    $x_i = \Theta(W \bullet X_i)$
    **if** $x_i \neq y_i$ **then**
      choose learning rate $\eta$
      choose optimal relabeling $\pi \in \mathcal{R}_{X_i, W}$
      update $W = W + \eta\, y_i X_i^{\pi}$
  **end for**
**until** some criterion is satisfied

**Output:** $H_W : X \mapsto \Theta(W \bullet X)$

---

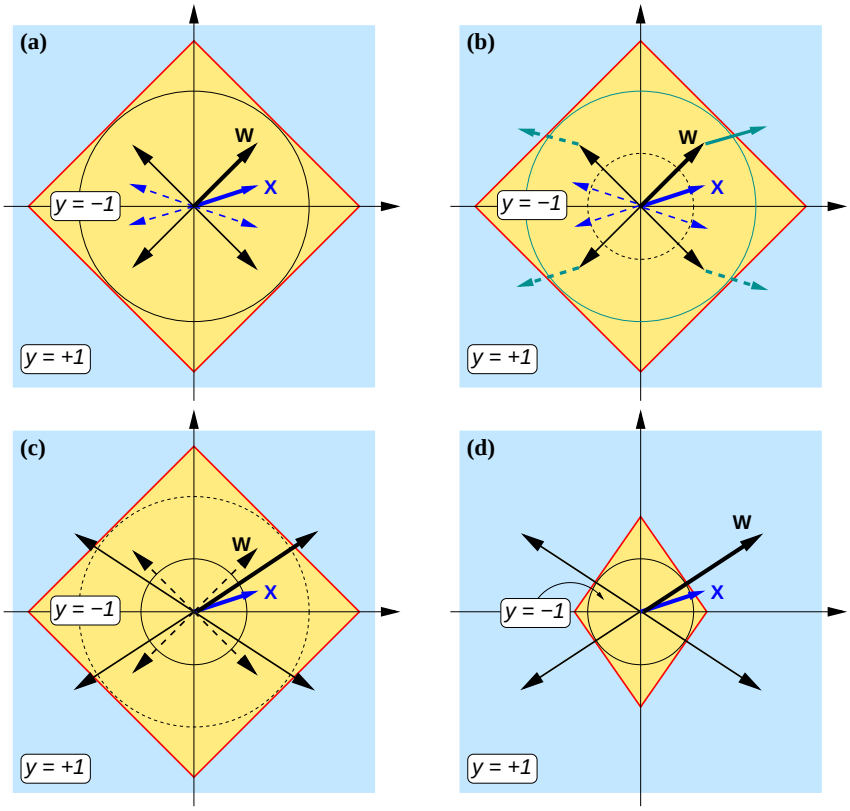



**Figure 12.2** Geometrical illustration of update rule (12.7). For convenience, we uncouple the bias $h$ from the extended weight graph and consider the usual weight graph $W$ and bias $h$ separately. Suppose that example $X$ labeled with $y = 1$ is misclassified by hypothesis $H(X) = W \bullet X + h$ as $y = -1$. In (a), we consider the vector representations $\mathcal{V}_{[X]}$ and $\mathcal{V}_{[W]}$ of the unlabeled graphs $[X]$ and $[W]$. We choose an arbitrary labeled representative $X$ of $[X]$ and determine the spoke $W$ closest to $X$. Both labeled graphs, $X$ and $W$, are highlighted. Since the misclassified example $X$ has label $y = 1$, we add $X$ to $W$. The effect of $W + X$ to all labeled representatives of $[W + X]$ is indicated in (b) and (c). Finally, (d) shows the updated weight graph and the resulting decision regions.

Since the s-perceptron criterion $E^{perc}(W)$ is locally Lipschitz, it is differentiable almost everywhere. Thus, from the perspective of differentiability, we have a similar situation as in the case of the standard perceptron criterion.

### 12.4.3 Structural Perceptron Convergence Theorem

Given a structural linearly separable training sample, we want to present a convergence result for Algorithm 14.

The standard *perceptron convergence theorem* states that the perceptron algo-

rithm finds a solution that correctly classifies the examples of a linearly separable training sample after a finite number of iteration steps. In addition, the theorem is shown for a constant learning rate $\eta_t = 1$ for all $t \geq 0$.

The main problem in extending the standard perceptron convergence theorem to Algorithm 14 is that known proofs exploit the bilinearity property of the inner product. Since the structural dot product is not bilinear, it is unclear how to derive a strong convergence result similar to the one of the standard case. Weaker convergence results can be obtained by bounding the behavior of the average error $E^{perc}$ over time. In particular, we show that the structural perceptron converges in the limit to a solution graph when we decrease the learning rate appropriately at each iteration step.

We use the following notation: let $j = (t \mod p) + 1$ for all $t \geq 0$, where $p$ is the cardinality of the training sample $\mathcal{Z}$. The subscript $t$ in $E_t^{perc}$, $X_t$, and $y_t$ refers to the $j$-th component function $E_j^{perc}$ of $E^{perc}$ and the $j$-th training example $(X_j, y_j) \in \mathcal{Z}$. With this notation at hand, we formulate the following important result.

***Lemma 12.1***
*Consider the structural perceptron criterion (12.6) for a structural linearly separable training sample $\mathcal{Z}$ with solution graph $W^*$. Let $\chi = \max_{i\in[1:p]} \|X_i\|$, and let $W_0$ be the initial weight graph. Then there are positive constants $\varepsilon_0, \ldots, \varepsilon_t$ such that*

$$\frac{1}{t}\sum_{i=0}^{t} E_i^{perc}(W_i) \leq \frac{\delta\big(W_0, W^*\big)^2 + 2\chi^2 \sum_{i=0}^{t} \eta_t^2}{2t \min_{i\in[0:t]} \eta_i} \tag{12.8}$$

*whenever $0 < \eta_i \leq \varepsilon_i$ for all $i \in [0:t]$.*

***Proof*** Suppose that $X_t$ is misclassified. Without loss of generality, we assume that the identity matrix $\boldsymbol{I} = \boldsymbol{I}_n$ is an optimal rotation of $X_t$ towards the current weight graph $W_t$. The weight graph is then updated according to

$$W_{t+1} = W_t + \eta_t y_t X_t.$$

First we show that there exists a constant $\varepsilon_t > 0$ such that the intersection

$$\mathcal{R}(W^*, W_{t+1}) \cap \mathcal{R}(W^*, W_t)$$

of the sets of optimal rotations of $W^*$ towards $W_{t+1}$ and towards $W_t$ is non-empty, provided that $0 < \eta_t \leq \varepsilon_t$. Let

$$f(W) = \max_{\boldsymbol{P}\in\Pi^n} f_{\boldsymbol{P}}(W) = \max_{\boldsymbol{P}\in\Pi^n} \boldsymbol{P}^{\mathsf{T}}\boldsymbol{W}^*\boldsymbol{P} \bullet \boldsymbol{W}$$

be the pointwise maximizer with linear support functions determined by $W^*$. The directional derivative of $f$ at $W_t$ in direction $y_t X_t$ is of the form

$$f'(W_t, y_t X_t) = \max\big\{\nabla f_{\boldsymbol{P}}(W_t) \bullet y_t X_t \,:\, f_{\boldsymbol{P}} \in \operatorname{supp}(f, W_t)\big\}.$$

Without loss of generality, we assume that $f_{\boldsymbol{I}} \in \operatorname{supp}(f, W_t)$ with $f'(W_t, y_t X_t) =$

$\nabla f_{\boldsymbol{I}}(W_t) \bullet y_t X_t$. Note that such a $f_{\boldsymbol{I}}$ exists, because $f$ is directionally differentiable at $W_t$. We have

$$\begin{aligned} f'(W_t, y_t X_t) &= \lim_{\alpha \downarrow 0} \frac{f(W_t + \alpha y_t X_t) - f(W_t)}{\alpha} \\ &= \lim_{\alpha \downarrow 0} \frac{\max_{\boldsymbol{P} \in \Pi^n} \boldsymbol{P}^{\mathsf{T}} \boldsymbol{W}^* \boldsymbol{P} \bullet \big(\boldsymbol{W}_t + \alpha y_t \boldsymbol{X}_t\big) - \boldsymbol{W}^* \bullet \boldsymbol{W}_t}{\alpha} \\ &= \nabla f_{\boldsymbol{I}}(W_t) \bullet y_t \boldsymbol{X}_t = \boldsymbol{W}^* \bullet y_t \boldsymbol{X}_t. \end{aligned}$$

Hence, there is an $\varepsilon_t > 0$ such that

$$f(W_t + \alpha y_t X_t) = f_{\boldsymbol{I}}(W_t + \alpha y_t X_t)$$

for all $\alpha \in [0, \varepsilon_t]$. This implies that

$$\boldsymbol{I} \in \mathcal{R}(W^*, W_{t+1}) \cap \mathcal{R}(W^*, W_t).$$

Next, we examine the distance from the updated weight graph $W_{t+1}$ to the separating graph $W^*$. We have

$$\begin{aligned} \big\|\boldsymbol{W}_{t+1} - \boldsymbol{W}^*\big\|^2 &= \big\|\boldsymbol{W}_t + \eta_t y_t \boldsymbol{X}_t - \boldsymbol{W}^*\big\|^2 \\ &= \big\|\boldsymbol{W}_t - \boldsymbol{W}^*\big\|^2 + 2\eta_t y_t \boldsymbol{X}_t \bullet \big(\boldsymbol{W}_t - \boldsymbol{W}^*\big) + \eta_t^2 \big\|\boldsymbol{X}_t\big\|^2, \end{aligned}$$

where we cancelled $y_t^2 = 1$ in the third term of the last line. Since $\boldsymbol{W}_t$ is a spoke closest to $\boldsymbol{X}_t$, we have

$$E_t^{perc}(W_t) = -y_t \boldsymbol{X}_t \bullet \boldsymbol{W}_t \geq 0.$$

Although $\boldsymbol{I}$ is an optimal rotation of $W^*$ towards $W_t + \eta_t y_t X_t$, it need not be an optimal rotation towards $X_t$. This observation implies that

$$-y_t \boldsymbol{X}_t \bullet \boldsymbol{W}^* \leq E_t^{perc}(W^*) = 0.$$

The equality $E_t^{perc}(W^*) = 0$ follows from the fact that $W^*$ is a separating graph. From

$$\begin{aligned} 2\eta_t y_t \boldsymbol{X}_t \bullet \big(\boldsymbol{W}_t - \boldsymbol{W}^*\big) &\leq -2\eta_t \Big(E_t^{perc}(W_t) - E_t^{perc}(W^*)\Big) \\ &= -2\eta_t E_t^{perc}(W_t) \end{aligned}$$

follows

$$\begin{aligned} \big\|\boldsymbol{W}_{t+1} - \boldsymbol{W}^*\big\|^2 &\leq \big\|\boldsymbol{W}_t - \boldsymbol{W}^*\big\|^2 - 2\eta_t E_t^{perc}(W_t) + \eta_t^2 \big\|\boldsymbol{X}_t\big\|^2 \\ &\leq \big\|\boldsymbol{W}_t - \boldsymbol{W}^*\big\|^2 + 2\eta_t E_t^{perc}(W_t) + \eta_t^2 \chi^2, \end{aligned}$$

where $\chi = \max_{i\in[1:p]} \|X_i\|$. Applying the above inequality recursively, we obtain

$$\begin{aligned}\left\|\boldsymbol{W}_{t+1} - \boldsymbol{W}^*\right\|^2 &\leq \left\|\boldsymbol{W}_0 - \boldsymbol{W}^*\right\|^2 - 2\sum_{i=0}^{t} \eta_t E_t^{perc}(W_t) + 2\chi^2 \sum_{i=0}^{t} \eta_t^2 \\ &\leq \left\|\boldsymbol{W}_0 - \boldsymbol{W}^*\right\|^2 - 2\eta_* \sum_{i=1}^{t} E_i^{perc}(W_i) + 2\chi^2 \sum_{i=0}^{t} \eta_t^2,\end{aligned}$$

where $\eta_* = \min_{i\in[0;t]} \eta_t$ denotes the minimal learning rate. Note that $\eta_i \leq \varepsilon_i$ for all $i \in [0:t]$, where $\varepsilon_i$ is chosen in the same way as described in the first part of this proof. Let

$$\mu_t = \frac{1}{t}\sum_{i=0}^{t} E_i^{perc}(W_i)$$

be the average error. We have

$$2t\eta_*\mu_t \leq \left\|\boldsymbol{W}_0 - \boldsymbol{W}^*\right\|^2 + 2\chi^2 \sum_{i=0}^{t} \eta_t^2.$$

Dividing both sides by $2t\eta_*$ yields the assertion. ■

From equation (12.8), we can directly derive convergence results. For $t > 0$ let

$$\mu_t^{perc} = \frac{1}{t}\sum_{i=0}^{t} E_i^{perc}(W_i)$$

denote the average error of the component functions $E_1^{perc}, \ldots, E_p^{perc}$. Given the assumptions in Lemma 12.1, we consider two types of rules for the learning rate $\eta_t$.

1. *Constant Learning Rate.* If $\eta_t = \eta$, the average error is bounded by

$$\mu_t^{perc} \leq \frac{\delta\left(W_0, W^*\right)^2 + 2\chi^2 t\eta^2}{2t\eta}.$$

Taking the limit, we have

$$\lim_{t\to\infty} \mu_t^{perc} \leq \lim_{t\to\infty} \frac{\delta\left(W_0, W^*\right)^2 + 2\chi^2 t\eta^2}{2t\eta} = \chi^2\eta.$$

This shows that for a constant learning rate $\eta$ the average error converges to a value bounded above by $\chi^2\eta$. The immediate implication of this result is twofold: first, we cannot guarantee convergence to a separation graph, but we can bound the expected error; and second, we can control the bound of the expected error by the magnitude of the learning rate.

2. *Stochastic Approximation.* Suppose that the learning rate satisfies

$$\|\boldsymbol{\eta}\|^2 = \sum_{i=0}^{\infty} \eta_i^2 < \infty, \qquad \sum_{i=0}^{\infty} \eta_i = \infty.$$

A typical example is $\eta_t = \alpha/(\beta + t)$, where $\alpha > 0$ and $\beta > 0$. Then we have

$$\frac{1}{t}\mu_t^{perc} \leq \frac{\delta\big(W_0, W^*\big)^2 + 2\chi^2\|\boldsymbol{\eta}\|^2}{2\sum_{i=0}^{t}\eta_i} \leq \frac{C}{2\sum_{i=0}^{t}\eta_i},$$

where $C$ is a positive constant bounding the nominator of the second term in the above inequality. Taking the limit $t \to \infty$ shows that the expected error converges to zero. Since the component functions $E_1^{perc}, \ldots, E_p^{perc}$ are nonnegative, the Structural Perceptron Algorithm 14 converges to a separating graph.

As opposed to the standard perceptron, an s-perceptron ensures convergence to a solution graph for structurally separable patterns only in the limit, provided that the learning rate is decreased according to the principle of stochastic approximation. Note that this result does not imply that an s-perceptron cannot find a solution graph after a finite number of iteration steps. Theorem 12.1 summarizes the second convergence result.

***Theorem 12.1***
*Consider the structural perceptron criterion (12.6) for a structural linearly separable training sample $\mathcal{Z}$ with solution graph $W^*$. Suppose that the learning rate decreases according to the rule*

$$\sum_{i=0}^{\infty}\eta_i^2 < \infty \quad \textit{and} \quad \sum_{i=0}^{\infty}\eta_i = \infty.$$

*Then we have*

$$\lim_{t\to\infty}\frac{1}{t}\mu_t^{perc} = 0.$$

## 12.5 Conclusion

In this chapter, we presented single layer feedforward s-networks for adaptive processing of attributed graphs. For classification problems, a single s-unit represents a structural linear decision surface composed of hyperplane segments, which are interrelated via relabeling of the spokes. For a structurally separable training sample, we showed that an s-perceptron can, at least in the limit, learn what it represents. In addition, we presented an incremental subgradient learning algorithm for classification and function approximation problems.

We conclude with a twofold outlook. The foundation has been established for structural neural learning machines. It is beyond the scope of this work to provide an exhaustive compilation of theoretical results, including results on statistical learning theory as available for standard neural networks.

The limitations of single layer s-networks in terms of the range of functions they can represent shifts the focus of interest to multi layer s-networks. We examine multi layer s-networks in the next chapter.

# 13 Structural Multi-Layer Networks

As argued in the previous chapter, s-networks having a single layer of adaptive weights are limited in terms of the class of structural functions they can represent. This chapter is devoted to multi layer networks for attributed graphs that implement a broader range of structural functions. In Section 13.1, we propose a structural model of multi layer feedforward networks. Section 13.2 provides some results on representational capabilities of structural multi layer networks. To learn what a structural network can represent, Section 13.3 extends the error back-propagation algorithm to structural networks for finding the generalized gradients of an error function with respect to the weights and biases.

## 13.1 Structural Model of Multi Layer Neural Networks

A multi layer neural network for attributed graphs is either a single layer s-network, or a single layer s-network coupled with a standard multi layer neural network for feature vectors. In a multi layer s-network, the s-units in the first layer provide the input signals for the conventional units in the second layer.

Figure 13.1 shows an example of a three layer s-network. It has an input terminal whose role is to feed attributed graphs into the network. The first layer of units are s-units. The second and third layer of units are conventional units. The outputs of s-units in the first layer form a vector, which provides the input of the second layer of units. Units in the last layer are the *output units* of the s-network. All other units, regardless whether of conventional or structural type, are called *hidden units*, because their activations and outputs are invisible for the external environment.

Note that in the literature on standard multi layer networks, different conventions are used for counting layers. Here, we count layers of units in a feedforward direction, beginning with the layer of s-units as the first layer. Layers of weights refer to the layers of adaptive weights. Thus, the first layer of weights are the weight graphs associated with the s-units. Following this convention, each layer of weights corresponds to a layer of units. Hence, we may speak loosely about layers or the $r$-th layer when referring to the units, weights or both components in that layer.

Consider an s-network consisting of $\ell$ layers. Suppose that layer $r$ consists of $m_r$ units for $r \in [1:\ell]$, where output layer $\ell$ has $m_\ell = m$ units. The first layer

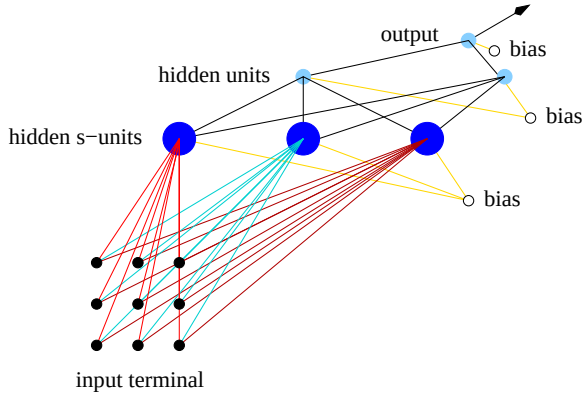



**Figure 13.1** Example of a feedforward s-network consisting of three layers of adjustable weights. Input units are depicted by black circles, s-units by large blue circles, and conventional units by light blue circles. The input terminal has matrix form. To feed an input graph $X$ into the network, the adjacency matrix of $X$ is applied to the terminal. Diagonal elements of the terminal receive input signals from vertex attributes and off-diagonal elements from non-vertex attributes of $X$. Similarly, connections emanating from diagonal elements represent vertex attributes, and connections emanating from off-diagonal elements represent non-vertex attributes of a weight graph. For each s-unit, the terminal performs an appropriate relabeling of $X$ to maximize the structural dot product with the associated weight graph. The output of the s-units forms a three-dimensional vector and can therefore be regarded as input of a standard multi layer network. The biases are shown as extra input and hidden units, respectively, with fixed output 1.

implements a function

$$H^{(1)} : \mathcal{G}_{\mathcal{A}} \to \mathbb{R}^{m_1}, \quad X \mapsto \boldsymbol{x}^{(1)} = \left(g\left(u_1^{(1)}\right), \ldots, g\left(u_{m_1}^{(1)}\right)\right)^{\mathsf{T}},$$

where the vector $\boldsymbol{x}^{(1)} \in \mathbb{R}^{m_1}$ collects the output values

$$x_i^{(1)} = g\left(u_i^{(1)}\right) = g\left(W^i \bullet X + h^i\right)$$

of all s-units $i$, and $g$ represents the transfer function. As usual, $W^i$ and $h^i$ denote the weight graph and bias associated with s-unit $i$. Indices referring to the first layer are enclosed in parentheses. We will use this convention for layers of higher rank as well. We dropped the layer index for $W^i = W^{i(1)}$ and $h^i = h^{i(1)}$, because weight graphs and biases in the first layer can be identified by the superscript index $i$ referring to the associated s-unit. For weights and biases in the following layers, reference to their associated units is indicated by a subscript index $i$.

Similarly, each layer $r \in [2\!:\!\ell]$ implements a function

$$H^{(r)} : \mathbb{R}^{m_{r-1}} \to \mathbb{R}^{m_r}, \quad \boldsymbol{x}^{(r-1)} \mapsto \boldsymbol{x}^{(r)} = \left(g\left(u_1^{(r)}\right), \ldots, g\left(u_{m_l}^{(r)}\right)\right)^{\mathsf{T}}.$$

The output vector of layer $r \in [2:\ell]$ is of the form

$$\boldsymbol{x}^{(r)} = g\left(\boldsymbol{u}^{(r)}\right) = g\left(\boldsymbol{W}^{(r)}\boldsymbol{x}^{(r-1)} + \boldsymbol{h}^{(r)}\right),$$

where $\boldsymbol{W}^{(r)} \in \mathbb{R}^{m_r \times m_{r-1}}$ denotes the weight matrix describing the connections from layer $r-1$ to layer $r$. Each row $\boldsymbol{w}_{i:}^{(r)}$ of $\boldsymbol{W}^{(r)}$ is composed of $m_{r-1}$ weights $w_{ij}^{(r)}$ connecting unit $j$ from layer $r-1$ with unit $i$ from layer $r$. The vector $\boldsymbol{h}^{(r)} \in \mathbb{R}^{m_r}$ collects the biases of all the $m_r$ units in layer $r$.We assume that the transfer function $g$ is identical for all units within the same layer, but may vary from layer to layer. The particular form of $g$ can be recovered by the layer index $r$ of its argument $\boldsymbol{u}^{(r)}$.

Merging the modules of the first and the following layers gives an s-network that implements a functional relationship of the form

$$H : \mathcal{G}_{\mathcal{A}} \to \mathbb{R}^m, \quad X \mapsto \boldsymbol{x}_\ell = H^{(\ell)} \circ \cdots \circ H^{(2)} \circ H^{(1)}(X). \tag{13.1}$$

## 13.2 Representational Capabilities

This section discusses which classes of functions can be represented by multi layer s-networks. For the same reason as in the standard case, the constructive proofs presented here are of little practical value [32]. Their purpose is to characterize the hypothesis space covered by different s-network architectures. In what follows, we assume that arbitrary large networks can be constructed if needed.

Let

$$\widehat{f} : [U] \to \mathbb{R}^m$$

be a function defined on a subset of the space $\left[\mathbb{G}_{\mathcal{A}}\right]$ of unlabeled graphs. What we want to know is whether there is a multi layer s-network that implements $\widehat{f}$ at least to arbitrary accuracy. To respond to this question, we consider the space $\mathbb{G}_{\mathcal{A}}$ of labeled graphs. The function $\widehat{f}$ gives rise to a uniquely determined structural function $f$ on a subset $U$ of $\mathbb{G}_{\mathcal{A}}$ with $f(X) = f([X])$. Since the space $\mathbb{G}_{\mathcal{A}}$ is isomorphic to a Euclidean space $\mathcal{A}^N$ $(N = n^2)$, the following diagram commutes:

$$\begin{array}{ccccc}
\mathcal{A}^N & \xleftarrow{\text{vec}} & \mathbb{G}_{\mathcal{A}} & \xrightarrow{[.]} & \left[\mathbb{G}_{\mathcal{A}}\right] \\
\downarrow{\scriptstyle f^{\text{vec}}} & & \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle \widehat{f}} \\
\mathbb{R}^m & \xrightarrow[\text{id}]{} & \mathbb{R}^m & \xleftarrow[\text{id}]{} & \mathbb{R}^m
\end{array}$$

The vec-operation transforms a labeled graph $W$ from $\mathbb{G}_{\mathcal{A}}$ to its vector representation $\boldsymbol{w} = \text{vec}(W)$ by concatenating the columns of its adjacency matrix $\boldsymbol{W}$. By abuse of notation, we apply the vec-operation on labeled graphs rather than on their adjacency matrices.

With the commutative diagram in mind, we first state a conjecture for which we

neither know a proof, nor a counterexample.

***Conjecture 13.1***
*Let* $[U] \subseteq \left[\mathbb{G}_{\mathcal{A}}\right]$ *be a subset of the space of unlabeled graphs. A function*

$$\hat{f} : \left[U\right] \to \mathbb{R}^m$$

*can be represented by an s-network (to arbitrary accuracy) if* $f^{\text{vec}}$ *can be represented by a conventional neural network architecture (to arbitrary accuracy).*

Suppose that $f^{\text{vec}}$ is a function, which can be implemented by a conventional multi layer network $\mathfrak{N}$. It is unclear how to show the existence of an s-network $\mathfrak{N}_s$ that implements $\widehat{f}$. Converting $\mathfrak{N}$ to $\mathfrak{N}_s$ is intricate for two reasons: first, it is unclear how to construct an s-network $\mathfrak{N}_{\text{vec}}$ that implements the vec-operation. If we know how to construct $\mathfrak{N}_{\text{vec}}$, all we have to do to obtain the desired s-network $\mathfrak{N}_s$ is join the output of $\mathfrak{N}_{\text{vec}}$ with the input of the conventional network $\mathfrak{N}$. Second, if there is no s-network $\mathfrak{N}_{\text{vec}}$, we have to substitute each unit $i$ in the first layer of $\mathfrak{N}$ by one or more s-units. Suppose that $\boldsymbol{w}_i$ is the weight vector and $h_i$ the bias associated with unit $i$. Then for any input pattern $\boldsymbol{x} \in \mathcal{A}^N$, the activation $u_i$ is defined by

$$u(\boldsymbol{x}) = \boldsymbol{w}_i^{\mathsf{T}} \boldsymbol{x} + h_i.$$

Let $X$ and $X'$ be isomorphic graphs with $\boldsymbol{x} = \text{vec}(X) \neq \text{vec}(X') = \boldsymbol{x'}$. Then the problem we face is that the activation of unit $i$ may possibly differ for different input patterns $\boldsymbol{x}$ and $\boldsymbol{x'}$, i.e. $u_i(\boldsymbol{x}) \neq u_i(\boldsymbol{x'})$. The situation for s-units is completely different. Since the structural dot product is a structural function, the activation of an s-unit $i$ is of the form

$$u_i(X) = W^i \bullet X + h^i = W^i \bullet X' + h^i = u_i(X').$$

It is unclear how to synthesize the different behavior of conventional and s-units.

Validity of Conjecture 13.1 would provide an answer to our question raised above for the broad class of functions that can be represented by conventional multi layer neural networks. Unless there is no proof to support the conjecture, we have to construct s-networks for particular classes of functions. In the following, we present constructive proofs for the representational capabilities of s-networks for selected examples of structural functions.

The first example shows that any structural boolean function on binary graphs can be represented by a two layer s-network consisting of threshold s-units in the hidden layer, and a single conventional threshold unit in the output layer.

***Theorem 13.1***
*Let $\mathcal{G}_{\mathbb{B}}$ be the set of binary graphs with attributes from $\mathbb{B} = \{0, 1\}$, and let*

$$\widehat{f} : [\mathcal{G}_{\mathbb{B}}] \to \mathbb{B}$$

*be a structural dichotomy. Suppose that weight graphs associated with s-units are elements of the set $\mathcal{G}_{\mathbb{B}'}$ of complex graphs with attributes from $\mathbb{B}' = \{\pm 1\}$. Then there exists a two layer s-network that implements a function $H$ with $H(X) = f(X)$ for all graphs $X \in \mathcal{G}_{\mathbb{B}}$.*

***Proof*** The null attributes of $\mathcal{G}_{\mathbb{B}}$ and $\mathcal{G}_{\mathbb{B}'}$ are $\epsilon_{\mathbb{B}} = 0$ and $\epsilon_{\mathbb{B}'} = -1$. We define the mapping

$$\mathrm{id}_{\mathbb{B}\mathbb{B}'} : \mathcal{G}_{\mathbb{B}} \to \mathcal{G}_{\mathbb{B}'}, \quad X \mapsto X_{\mathbb{B}'}$$

with

$$X_{\mathbb{B}'} \bullet X = X \bullet X. \tag{13.2}$$

The mapping $\mathrm{id}_{\mathbb{B}\mathbb{B}'}$ exists and satisfies the permutation invariant property. To show existence of $X_{\mathbb{B}'}$, we replace the null attribute $\epsilon_{\mathbb{B}}$ of non-edges in $X$ by the null attribute $\epsilon_{\mathbb{B}'}$. Clearly, $X_{\mathbb{B}'}$ satisfies (13.2). Hence, by construction, $\mathrm{id}_{\mathbb{B}\mathbb{B}'}$ is well-defined and bijective with inverse mapping $\mathrm{id}_{\mathbb{B}'\mathbb{B}}$. We show that $\mathrm{id}_{\mathbb{B}\mathbb{B}'}$ satisfies the permutation invariant property. Let $Y$ be isomorphic to $X$. Then $Y_{\mathbb{B}'}$ is isomorphic to $X_{\mathbb{B}'}$ and

$$X_{\mathbb{B}'} \bullet X = X \bullet X = X \bullet Y = X_{\mathbb{B}'} \bullet Y = Y_{\mathbb{B}'} \bullet Y.$$

Next, we show that

$$X_{\mathbb{B}'} \bullet Y < X \bullet X \tag{13.3}$$

for all binary graphs $Y \in \mathcal{G}_{\mathbb{B}}$ not isomorphic to $X$. Since $X$ and $Y$ are both binary graphs, it is easy to see that

$$X \bullet Y \leq X \bullet X.$$

We distinguish three cases:

*Case 1:* Let $|Y| < |X|$. Then from $d = |X| - |Y| > 0$, it follows that

$$X \bullet Y \leq X \bullet X - d < X \bullet X.$$

Since $\epsilon_{\mathbb{B}'} < \epsilon_{\mathbb{B}}$, we have

$$X_{\mathbb{B}'} \bullet Y \leq X \bullet Y < X \bullet X.$$

*Case 2:* Let $|Y| = |X|$. Since $Y$ is not isomorphic to $X$, there is at least one

mismatch between non-vertex items. Hence, we have

$$X \bullet Y \leq X \bullet X - 1 < X \bullet X.$$

With the same argumentation as in the first case, we find that $X_{\mathbb{B}'} \bullet Y < X \bullet X$.

*Case 3:* Let $|Y| > |X|$. We have $d = |Y| - |X| > 0$. Since $X_{\mathbb{B}'}$ has attributes from $\mathbb{B}'$ with null attribute $\epsilon_{\mathbb{B}'} = -1$, alignment of $X_{\mathbb{B}'}$ to the size of $Y$ adds dummy items labeled with $\epsilon_{\mathbb{B}'}$. Hence, we have

$$X_{\mathbb{B}'} \bullet Y \leq X \bullet Y - 2d < X \bullet X.$$

Combining the results of the three case shows (13.3).

Now we turn to the actual part of the proof. Let $f$ be a structural function $\mathcal{G}_{\mathbb{B}}$ with $f(X) = \widehat{f}([X])$ for all $X \in \mathcal{G}_{\mathbb{B}}$. The number of binary graphs of bounded order is finite. Hence, the sets $\mathcal{G}_{\mathbb{B}}$ and $\left[\mathcal{G}_{\mathbb{B}}\right]$ are finite, because we assumed that all graphs are of bounded order $n$. We take one hidden threshold s-unit $i_{[X]}$ for every unlabeled graph $[X] \in \left[\mathcal{G}_{\mathbb{B}}\right]$ with $\widehat{f}([X]) = 1$. The weight graph $W^{i_{[X]}}$ associated with s-unit $i_{[X]}$ is of the form

$$W^{i_{[X]}} = X_{\mathbb{B}'}.$$

We set the bias of $i_{[X]}$ to $h^{i_{[X]}} = 1 - X \bullet X$. Then the activation $u_{i_{[X]}}$ of s-unit $i_{[X]}$ is given by

$$u_{i_{[X]}} = W^{i_{[X]}} \bullet Y + h^{i_{[X]}} = X_{\mathbb{B}'} \bullet Y + h^{i_{[X]}}.$$

Using $h^{i_{[X]}} = 1 - X \bullet X$, we find from the first part of the proof that

$$x_{i_{[X]}} = \Theta\left(u_{i_{[X]}}\right) = \begin{cases} 1 & : \quad X' \simeq X \\ 0 & : \quad \text{otherwise} \end{cases}$$

for any input graph $X'$. Hence, the proof is complete if we connect each hidden s-unit to a single output threshold unit with weight $+1$ and set its bias to 0. ■

The proof of Theorem 13.1 allows a generalization to dichotomies on the set $\mathcal{G}_{\mathbb{B}^d}$ of attributed graphs colored with attributes from $\mathbb{B}^d$.

***Corollary 13.1***
*Any structural dichotomy*

$$\widehat{f} : \left[\mathcal{G}_{\mathbb{B}^d}\right] \to \mathbb{B}$$

*can be represented by a two layer s-network consisting of threshold units.*

***Proof*** Let $\mathcal{A} = \mathbb{B}^d$, and let $\mathcal{A}' = \{\pm 1\}^d$. The null attributes of $\mathcal{G}_{\mathcal{A}}$ and $\mathcal{G}_{\mathcal{A}'}$ are $\epsilon_{\mathcal{A}} = \mathbf{0}_d$ and $\epsilon_{\mathcal{A}'} = -\boldsymbol{e}_d$. We define the mapping

$$\mathrm{id}_{\mathcal{A}\mathcal{A}'} : \mathcal{G}_{\mathcal{A}} \to \mathcal{G}_{\mathcal{A}'}, \quad X \mapsto X_{\mathcal{A}'}$$

with

$$X_{\mathcal{A}'} \bullet X = X \bullet X.$$

The assertion follows from a similar argumentation as in the proof of Theorem 13.1. ■

The constructions in Theorem 13.1 and Corollary 13.1 are of little practical use, because we merely store the input graphs and abandon the capability of the s-network to generalize. But as in the standard case, the construction illustrates the concept of a *structural template*. Each hidden s-unit serves as a structural template for the corresponding input graph in the sense that it only fires when the input matches the template graph. As an immediate consequence, we may state that any classification problem on finite graphs colored with symbolic attributes from a finite set of symbols can be solved by a two layer s-network.

Now we turn from binary to continuous attributes. The next result shows that an s-network can approximate any decision boundary to arbitrary accuracy.

***Theorem 13.2***
*Let $\mathcal{A} = \mathbb{R}^d$ be the set of vertex and edge attributes. Suppose that*

$$\widehat{f} : [\mathbb{G}_{\mathcal{A}}] \to \mathbb{B}$$

*is a dichotomy. Let $\mathcal{R} = \widehat{f}^{-1}(\{0\})$ denote the region of all attributed graphs $X \in \mathcal{G}_{\mathcal{A}}$ with $\widehat{f}(X) = 0$. Then a three layer s-network can approximate region $\mathcal{R}$ to arbitrary accuracy.*

***Proof*** We embed $\mathbb{G}_{\mathcal{A}}$ into the vector space $\mathcal{A}^N$ ($N = n^2$), where each labeled graph $X$ is represented by the vector $\boldsymbol{x} = \mathrm{vec}(X) \in \mathcal{V}_{[X]}$. Since $f$ is a structural function, the function $f^{\mathrm{vec}}$ is constant on the set $\mathcal{V}_{[X]}$. It is sufficient to show that a two layer s-network with one output unit can generate a hypercube of any size and position in $\mathcal{A}^N$. To this end, we consider the vectors

$$\boldsymbol{w}_1 = \boldsymbol{u}_1, \quad \boldsymbol{w}^2 = \boldsymbol{u}_{n+1}, \quad \boldsymbol{w}_3 = -\boldsymbol{u}_1, \quad \boldsymbol{w}_4 = -\boldsymbol{u}_{n+1},$$

where $\boldsymbol{u}_i$ denotes the $i$-th unit vector of $\mathcal{A}^N$. Thus, $\boldsymbol{w}_1$ is the vector representation of the labeled complex graph $W^1$ consisting of a single isolated vertex colored with attribute $\boldsymbol{e}_d$. Note that $\boldsymbol{e}_d$ is the $d$-dimensional vector of all ones. All other items of $W^1$ are colored with the null attribute $\epsilon = \boldsymbol{0}_d$. Similarly, $\boldsymbol{w}_2$ is the vector representation of the labeled complex graph $W^2$ consisting of an imaginary edge colored with $\boldsymbol{e}_d$; all other items of $W^2$ are colored with $\epsilon$. The structure of the labeled graphs $W^3$ and $W^4$ represented by $\boldsymbol{w}_3$ and $\boldsymbol{w_4}$ directly follows from $W^1$ and $W^2$.

Let $h^i < 0$ for all $i = [1:4]$. Each function

$$H_i(X) = W^i \bullet X + h^i$$

determines a decision surface in $\mathcal{A}^N$ composed of hyperplane segments parallel to the coordinate axes with distance $h^i$ to the origin (note that $\|W^i\| = 1$). Assume that $h^i$ is identical to $h$ for all $i = [1:4]$. Then the region $\mathcal{R}'$ of all graphs $X$ satisfying

$$H_i(X) \leq 0$$

for all $i = [1:4]$ is a hypercube with center $\mathbf{0}_d$ and diagonal $\sqrt{2h}$. By varying $h^1, \ldots, h^4$, we can control the size and position of the hypercube. Hence, the following network architecture generates an arbitrary hypercube as a decision region for all graphs $X$ with $f(X) = 0$. The first layer consists of four structural threshold units, each of which is associated with one of the weight graphs $W^i$ defined above, and has a bias $h^i$. The output layer consists of a conventional threshold unit. The connections from s-units to the output unit have weight 1. The bias of the output unit is set to $-3$. Hence, the output unit fires a 1 if, and only if, all hidden s-units have output 1. In turn, the latter occurs if, and only if, a graph from the hypercube $\mathcal{R}'$ is presented to the network.

Now everything comes down to the case of feature vectors and the assertion follows by applying the construction proposed by [231]. ■

For the same reasons as in the case of feature vectors, the constructive proof of Theorem 13.2 is of little practical value. The construction requires specification of the decision boundary in advance and typically results in s-networks with large number of connections. But we obtain a notable result when comparing the complexity of s-networks with that of conventional networks. The existence proof of networks to approximate a decision boundary to arbitrary accuracy in the case of feature vectors follows the same line as the proof of Theorem 13.2. Suppose that we are given feature vectors of dimension $n$. Then we need $2n$ units in the first layer to determine a single hypercube as a decision region. Each group of $2n$ units in the first layer is connected to a single unit in the second layer. In addition, each of the $2n + 1$ units has a bias. This gives $2n^2 + 4n + 1$ adjustable weights and biases for representing a single hypercube in $\mathbb{R}^n$.

Now let us consider the complexity of s-networks. Suppose that the weight graphs are all of order $n$. Regardless of the magnitude of $n$, we always need four s-units in the first layer and one conventional unit in the second layer to determine a single hypercube. Since the weight graphs are supposed to be undirected, the number of distinct weights associated with an s-unit is $n(n+1)/2$. This gives a total of $2n(n+1)$ adjustable weights in the first layer. Including the biases, we have $2n^2 + 2n + 4$ free parameters in the first layer. Adding the weights to the conventional unit in the second layer and its bias finally gives a total of $2n^2 + 2n + 9$ adjustable weights and parameters to respond to a single hypercube.

Provided that $n > 4$, an s-network with weight graphs of order $n$ requires less weights and biases to represent a hypercube in a quadratically higher dimensional space $\mathbb{R}^N$ than a conventional network in the $n$-dimensional space $\mathbb{R}^n$. Intuitively,

this result is somewhat surprising, because the number of weights of an s-unit depends quadratically on its order. But the permutation invariance of the structural dot product compensates the high number of weights associated with an s-unit by allowing construction of more complex decision regions as in the case of conventional units.

In our last example, we show that any smooth structural function on a compact subset of $\mathbb{G}_{\mathcal{A}}$ can be approximated by an s-network to arbitrary accuracy.

***Theorem 13.3***
*Suppose that*

$$f : U \to \mathbb{R}^m$$

*is a smooth structural function on a compact subset $U$ of $\mathbb{G}_{\mathcal{A}}$. Then a three layer s-network can approximate $f$ to any desired accuracy.*

***Proof*** We use the same architecture as in the proof of Theorem 13.2, but replace the threshold transfer function by the logistic activation function in the hidden layers and by a linear transfer function in the output layer. Then the proof follows from [223]. ■

The results on the expressive power of s-networks indicate what we can potentially expect, but they provide no useful means for obtaining appropriate weights to solve practical pattern recognition problems. The main reason is that in a practical setting, the function to be approximated by an s-network is unknown. Hence, we are unable to specify the weights in advance. The next subsection shows how we can obtain suitable weights to approximate a given, but unknown, functional relationship.

## 13.3 Subgradient Backpropagation Algorithm

We now turn to the problem of how an s-network can learn a given functional relationship. As previously mentioned, the learning problem is posed as a minimization problem of a suitable error criterion as a function of the weights.

In the following, we consider extended weight and input graphs as described in Notation 12.1. To unclutter the notation, we drop the $\widehat{\ }$-accent.

Assume that we are given a training sample

$$\mathcal{Z} = \big\{(X_1, \boldsymbol{y}_1), \ldots, (X_p, \boldsymbol{y}_p)\big\} \subseteq \mathcal{X} \times \mathcal{Y}$$

consisting of $p$ *training graphs* $X_q \in \mathcal{X} \subseteq \mathcal{G}_{\mathcal{A}}$ drawn from some *input space* $\mathcal{X}$ together with corresponding output values $\boldsymbol{y}_q \in \mathcal{Y} \subseteq \mathbb{R}^m$. The problem is to

estimate an unknown functional relation

$$f : \mathcal{X} \to \mathcal{Y}$$

given the training sample $\mathcal{Z}$ and a hypothesis space $\mathcal{H}$ of functions $H : \mathcal{X} \to \mathcal{Y}$. Here we are concerned with a hypothesis space $\mathcal{H}$ consisting of functions of the form (13.1) that can be implemented by a multi layer s-network. In addition, we assume that all transfer functions occuring in (13.1) are continuously Fréchet differentiable.

Following the principle of empirical risk minimization, the basic approach in learning defines a suitable error criterion as a function of the weights and biases that is then minimized by matching the network outputs with the desired outputs. Here we shall consider error functions of the general form

$$E(\mathcal{W}) = \sum_{q=1}^{p} E_q(\mathcal{W}),$$

where $E_q(\mathcal{W}) = l(\boldsymbol{x}, \boldsymbol{y})$ expresses the error (loss) of the network's output $\boldsymbol{x}_q = \boldsymbol{x}_q^{(\ell)}$ against the desired output $\boldsymbol{y}$ when presented with input graph $X_q$. The set $\mathcal{W}$ represents all the adjustable weights (including biases) in the s-network. It is useful to distinguish between weights from the first layer and weights from the remaining layers. We write

$$\mathcal{W} = \mathcal{W}_s \cup \mathcal{W}_c,$$

where $\mathcal{W}_s$ represents all the weights associated with s-units in the first layer and $\mathcal{W}_c$ represents all the weights associated with conventional units in the subsequent layers.

If the loss $l$ is continuously Fréchet differentiable as a function of $\boldsymbol{x}$, the error $E$ is continuously Fréchet differentiable as a function of the weights from $\mathcal{W}_c$. Since the structural dot product $W \bullet X$ is regular as a function of $W$ and all other functions applied on $W \bullet X$ are continuously Fréchet differentiable, the error $E$ is regular as a function of the weights from $\mathcal{W}_s$. Hence, we can apply the standard back-propagation rule for layers $r \in [2 : \ell]$ followed by a *subgradient back-propagation rule* for the first layer. Note that the generalized gradient of a continuously Fréchet differentiable function consists of a single subgradient, namely the gradient of that function. Hence, we may denote the whole back-propagation pass from the output layer to the first layer as a *subgradient back-propagation algorithm.*

Before describing the interaction of the back-propagation pass for the standard and the structural part of the s-network, we first outline how incremental learning of an s-network proceeds:

1. *Feedforward Pass.* Present the next training example $(X_q, y_q) \in \mathcal{Z}$ to the s-network. Calculate the activation vectors $\boldsymbol{u}^{(r)}$ and output vectors $\boldsymbol{x}^{(r)}$ for all $\ell$ layers in the s-network by application of (13.1).
2. *Error Back-Propagation to Layer* $r = 2$. Starting from the output layer $r = \ell$,

successively evaluate the partial derivatives

$$\frac{\partial E_q}{\partial \boldsymbol{W}^{(r)}}$$

for all adjustable weights $\boldsymbol{W}^{(r)}$ in each layer $r$ until the second layer $r = 2$.

3. *Error Back-Propagation to Layer* $r = 1$. Evaluate subgradients

$$G_k \in \partial E_q \left(W^k\right)$$

for all s-units $k$ in the first layer.

4. *Updating.* Adjust all weights from $\mathcal{W}$ using the partial derivatives and subgradients provided by the back-propagation pass.

We discussed the feedforward pass in Section 13.1. Hence, it is sufficient to focus on the remaining three steps 2-4.

***Step 2: Error Back-Propagation to Layer*** $r = 2$

For $r \in [2 : \ell]$, the partial derivatives of $E_q$ with respect to the weights $w_{ij}^{(r)}$ are evaluated using the standard back-propagation procedure. According to the chain rule, we have

$$\frac{\partial E_q}{\partial w_{ij}^{(r)}} = \frac{\partial E_q}{\partial u_i^{(r)}} \frac{\partial u_i^{(r)}}{\partial w_{ij}^{(r)}} = \delta_i^{(r)} x_j^{(r-1)},$$

where the *sensitivity*

$$\delta_i^{(r)} = \frac{\partial E_q}{\partial u_i^{(r)}} \tag{13.4}$$

of unit $i$ in layer $r$ describes the overall error changes within the unit's activation. Differentiation of (13.4) for the output units $k$ yields

$$\delta_k^{(\ell)} = g'\left(u_k^{(\ell)}\right) l'(x_k, y_k),$$

where $y_k$ is the desired output of unit $k$ in layer $\ell$ and $x_k$ is the actual output of that unit when the network is presented input graph $X_q$. To evaluate $\delta_k^{(\ell)}$, we substitute appropriate expressions for $g'\left(u_k^{(\ell)}\right)$ and $l'(x_k, y_k)$, which are known once we have specified the architecture of the s-network and the loss function.

To evaluate the sensitivity $\delta_j^{(r)}$ of conventional units $j$ in hidden layers $r \geq 2$, application of the chain rule gives

$$\delta_j^{(r)} = g'\left(u_j^{(r)}\right) \sum_k w_{kj}^{(r+1)} \delta_k^{(r+1)}. \tag{13.5}$$

The sensitivity $\delta_j^{(r)}$ of a hidden unit can be obtained by propagating the sensitivities $\delta_k^{(r+1)}$ backwards from units of the next higher layer. Since we know how to

determine the sensitivities of the output units, it follows that by recursively applying the back-propagation formula (13.5), we can evaluate the sensitivities for all conventional units.

***Step 3: Error Back-Propagation to Layer*** $r = 1$

We now want to determine an arbitrary subgradient from the generalized gradient $\partial E_q\left(W^i\right)$ for each s-unit $i$ in the first layer. The component function $E_q$ is continuously Fréchet differentiable as a function of the activation $u_i^{(1)}$ and the activation $u_i^{(1)} = S_{X_q}\left(W^i\right)$ is regular as a function of $W^i$. Then $E_q$ is regular at $W^i$ and we can adapt the subgradient learning rule (12.5) presented in Section 12.3. The generalized gradient of $E_q$ at $W^i$ is of the form

$$\partial E_q\left(W^i\right) = \frac{\partial E_q}{\partial u_i^{(1)}} \circ \partial S_{X_q}\left(W^i\right).$$

The partial derivative of $E_q$ with respect to $u_i^{(1)}$ is the sensitivity $\delta_i^{(1)}$ of s-unit $i$ and can be computed with the back-propagation formula (13.5) using the sensitivities $\delta_j^{(2)}$ of the units in the second layer. Hence,

$$\delta_i^{(1)} X_q^\pi \in \partial E_q\left(W^i\right)$$

is a subgradient for any optimal relabeling $\pi \in \mathcal{R}_{X_q,W^i}$.

***Step 4: Updating***

From the preceding discussion, we arrive at two update rules. The first update rule adjusts weights from layers $r \geq 2$ based on gradient descent and the second update rule adjusts weights from the first layer based on subgradient methods. The compound update rule for incremental learning is of the form

$$\begin{aligned} w_{ij}^{(r)} &\leftarrow w_{ij}^{(r)} + \eta\, \delta_i^{(r)} x_j^{(r-1)}, \quad (2 \leq r \leq \ell) \\ W^i &\leftarrow W^i + \eta\, \delta_i^{(1)} X_q^\pi, \end{aligned}$$

where $\pi \in \mathcal{R}_{X_q,W^i}$ is an optimal relabeling.

## 13.4 Conclusion

In this chapter, we presented multi layer feedforward s-networks for adaptive processing of attributed graphs. We showed that a two layer s-network with threshold s-units in the first layer and conventional threshold units in the second layer can represent any structural dichotomy to arbitrary accuracy. In addition, a three layer s-network can approximate any structural smooth function on a compact subset of $\mathbb{G}_{\mathcal{A}}$. To learn what an s-network can represent, we proposed an incremental subgra-

dient backpropagation algorithm, which combines standard backpropagation with subgradient methods.

Since the central goal was to provide basic principles and mechanisms of structural neural learning machines, it is beyond the scope of this work to explore all the important results related to standard neural learning machines. Therefore, further research objectives of interest are concerned with the approximation properties of multi layer s-networks, improved learning techniques, generalization ability, regularization, and extension of other supervised neural learning approaches to the structural domain.

# 14 Structural Competitive Learning

In the preceding chapters, we studied supervised structural neural learning machines. This chapter is devoted to unsupervised structural learning. We extend competitive learning algorithms to attributed graphs. The aim of competitive networks is to cluster or categorize the input data. In Section 14.1, we first review simple competitive learning for feature vectors. Then Section 14.2 extends simple competitive learning to attributed graphs. Finally, in Section 14.3, we extend improved techniques of competitive learning to structural data as Self-Organizing Maps, Vector Quantization, and Adaptive Resonance Theory.

## 14.1 Simple Competitive Learning for Feature Vectors

Simple competitive learning constitutes the simplest connectionist model for unsupervised learning and may be regarded as a building block for advanced techniques including, for example, *Self-organizing Maps* [213, 214, 215], *Vector Quantization* [214], and *Adaptive Resonance Theory* [129].

In this section, we review simple competitive learning for feature vectors. A more detailed presentation including a discussion on its well-known limitations can be found in [166].

In simple competitive learning, the pattern space $\mathcal{Z}$ is usually a subset of a Euclidean space $\mathbb{R}^d$. Given the training sample

$$\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p\} \subseteq \mathcal{Z},$$

the goal of competitive learning is to find $k$ models

$$\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k\} \subseteq \mathcal{Z}$$

such that the average distortion $E(M, \mathcal{Y}; \mathcal{X})$ is minimized using gradient descent in sequential or incremental mode.

A *winner-take-all* (WTA) network is the simplest form of a competitive neural network. It consists of a single layer of $k$ inhibitory connected output units. Each output unit $i$ is associated with a model $\boldsymbol{y}_i \in \mathcal{Y}$ in the sense that its activation is determined by $\boldsymbol{y}_i^\mathsf{T}\boldsymbol{x}$ for a given input pattern $\boldsymbol{x} \in \mathbb{R}^n$. Note that without their associated models, the units of the WTA network form a MAXNET as described

in Section 7.3.

The competitive learning algorithm aims at moving the models $\boldsymbol{y} \in \mathcal{Y}$ to centers of the clusters in the input data $\mathcal{X}$ by using a *competitive learning rule*. Once the learning algorithm has converged, it can be applied as a pattern classifier where an input pattern is assigned to the class represented by the winner of the competition.

To find cluster structures in the input data and choose the cluster centers accordingly, competitive learning proceeds as follows. First initialize the models $\boldsymbol{y} \in \mathcal{Y}$. Then reiterate the following steps over the available set of input patterns from $\mathcal{Z}$ until convergence:

1. *Sampling.* Select the next training pattern $\boldsymbol{x} \in \mathcal{X}$ in turn or in random order. Alternatively, if the training sample $\mathcal{X}$ is not available a priori, draw an input pattern $\boldsymbol{x}$ from the pattern space $\mathcal{Z}$ with a certain probability. Apply the input pattern $\boldsymbol{x}$ to the network.

2. *Similarity Matching.* Find the *best matching* or *winning unit* $i^*$ for which the associated model $\boldsymbol{y}_{i^*}$ is closest to the current input pattern $\boldsymbol{x}$ with respect to the Euclidean distance. Formally, the winning unit $i^*$ is given by

$$i^* = \arg\min_{i \in [1:k]} ||\boldsymbol{x} - \boldsymbol{y_i}||.$$

If the models $\boldsymbol{y}_i$ are normalized, then minimizing the Euclidean distance of input pattern $\boldsymbol{x}$ and model $\boldsymbol{y}_i$ is equivalent to maximizing the activation $u_i = \boldsymbol{y}_i^\mathsf{T}\boldsymbol{x}$ of unit $i$. The activations $u_i$ provide the basis for competition among the units $i$, where the particular unit with the largest activation is declared as the winner of the competition.

3. *Updating.* Adjust the model $\boldsymbol{y}_{i^*}$ associated with the best matching unit $i^*$ by using the *standard competitive learning rule*

$$\boldsymbol{y}_{i^*} \leftarrow \boldsymbol{y}_{i^*} + \eta(\boldsymbol{x} - \boldsymbol{y}_{i^*}), \tag{14.1}$$

where $\eta$ is a time decreasing learning rate. Rule (14.1) moves model $\boldsymbol{y}_{i^*}$ closer to the current input pattern $\boldsymbol{x}$. This makes the winning unit $i^*$ more likely to win the competition on that pattern in the future. Gradually decreasing the learning rate to zero freezes the learned models.

Appropriate initialization and training of the network will gradually shift the models to the centers of the clusters. Algorithm 15 summarizes the simple competitive learning algorithm.

**Algorithm 15** (Simple Competitive Learning)

**Input:**

$\mathcal{X}$ – training sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p \in \mathcal{Z}$

$k$ – size of cluster set

**Initialize:**

set $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots \boldsymbol{y}_k\}$

**Procedure:**

**repeat**

randomly select $\boldsymbol{x} \in \mathcal{X}$

determine $i^* = \arg\min_i ||\boldsymbol{x} - \boldsymbol{y}_i||$

update $\boldsymbol{y}_{i^*} = \boldsymbol{y}_{i^*} + \eta(\boldsymbol{x} - \boldsymbol{y}_{i^*})$

decrease $\eta$

**until** no noticeable changes in $\mathcal{Y}$ are observed

**Output:** $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots \boldsymbol{y}_k\}$

Simple competitive learning aims at minimizing the average distortion [300]

$$E(M, \mathcal{Y}; \mathcal{X}) = \frac{1}{p} \sum_{j=1}^{k} \sum_{i=1}^{p} m_{ij} \|\boldsymbol{x}_i - \boldsymbol{y}_j\|^2,$$

where the gradient

$$\frac{\partial E}{\partial \boldsymbol{y}_j} = -\sum_{i=1}^{p} m_{ij}(\boldsymbol{x}_i - \boldsymbol{y}_j)$$

is just the sum of the changes $\Delta \boldsymbol{y}_j = \boldsymbol{x}_i - \boldsymbol{y}_j$ of the standard rule (14.1) over all input patterns $\boldsymbol{x}_i$ for which $j$ is the winning unit.

## 14.2 Structural Competitive Learning

Structural competitive learning operates on a subset $\mathcal{Z}$ of the set of attributed graphs $\mathcal{G}_\mathcal{A}$. Consider the training sample

$$\mathcal{X} = \{X_1, \ldots, X_p\} \subseteq \mathcal{Z},$$

consisting of attributed graphs from $\mathcal{Z}$. The goal of structural competitive learning is to find $k$ model graphs

$$\mathcal{Y} = \{Y_1, \ldots, Y_k\} \subseteq \mathcal{Z}$$

such that the average distortion $E(M, \mathcal{Y}; \mathcal{X})$ is minimized using generalized gradient descent in sequential mode.

Structural competitive learning differs from simple competitive learning for fea-

ture vectors in the following points:

1. *Structural Units.* Standard units of the WTA architecture are replaced by structural units (s-units). We associate a model graph $Y_i$ with each s-unit $i$.

2. *Similarity matching.* Given an input graph $X$, the best matching or winning s-unit is defined by

$$i^* = \arg\min_{i \in [1:k]} \delta\big(X, Y_i\big), \tag{14.2}$$

where $\delta$ is the structural Frobenius metric induced by the structural dot product. If the models are normalized, that is, $\|Y_i\| = 1$ for all $i$, then the best matching criterion given in (14.2) is equivalent to

$$i^* = \arg\max_{i \in [1:k]} X \bullet Y_i.$$

3. *Update rule.* Following the standard competitive learning rule (14.1), the structural competitive learning rule is of the form

$$Y_{i^*} \;\leftarrow\; Y_{i^*} + \eta\big(X^\pi - Y_{i^*}\big), \tag{14.3}$$

where $\pi$ is an optimal relabeling of $X$ to $Y_{i^*}$. By reorganizing the terms on the right hand side of (14.3), we obtain

$$Y_{i^*} \;\leftarrow\; \eta X^\pi + (1-\eta) Y_{i^*}.$$

According to Theorem 9.5, the updated model is a weighted mean of the current input graph $X$ and model $Y_{i^*}$. Hence, rule (14.3) moves model $Y_{i^*}$ closer to $X$. Since the function $X \bullet Y_{i^*}$ is Fréchet differentiable almost everywhere, the direction $X^\pi$ of movement is uniquely determined. At singular points, which only exceptionally occur, several directions of movement exist.

Figure 14.1 illustrates an example of the architecture of an inhibitory WTA s-network for competitive learning. Note that the architecture is closely related to the architecture of the structural WTA classifier presented in Section 7.4. The structural competitive learning procedure is summarized in Algorithm 16.

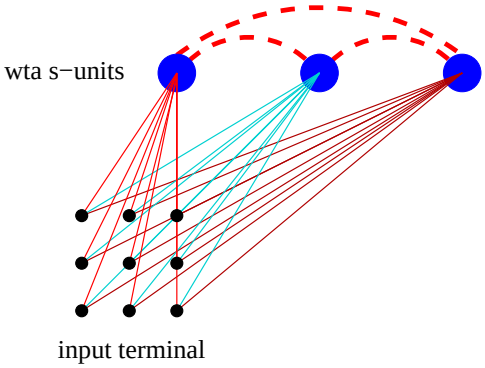



**Figure 14.1** Example of a winner-takes-all s-network consisting of three inhibitory connected s-units. Input units are depicted by black circles and s-units by large blue circles. The input terminal relabels applied input graphs $X$ appropriately for each s-unit. The highlighted dashed connections are inhibitory; the rest are excitatory.

---

**Algorithm 16** (Structural Competitive Learning)

---

**Input:**

$\mathcal{X}$ – training sample consisting of $p$ input graphs $X_1, \ldots, X_p \in \mathcal{Z}$

$k$ – size of cluster set

**Initialize:**

set $\mathcal{Y} = \{Y_1, \ldots Y_k\}$

**Procedure:**

**repeat**

randomly select $X \in \mathcal{X}$

determine $i^* = \arg\min_i \delta(X, Y_i)$

select $\pi \in \mathcal{R}_{X,Y_{i^*}}$

update $Y_{i^*} = Y_{i^*} + \eta\,(X^\pi - Y_{i^*})$

decrease $\eta$

**until** no noticeable changes in $\mathcal{Y}$ are observed

**Output:** $\mathcal{Y} = \{Y_1, \ldots Y_k\}$

---

Note that the learned cluster centers are attributed graphs from $\mathcal{G}_\mathcal{A}$, provided that the training sample $\mathcal{X}$ and the initial models are drawn from $\mathcal{G}_\mathcal{A}$. This contrasts with supervised neural learning for structures, where weight graphs may be adjusted to complex attributed graphs with real and imaginary part. Hence, the learned cluster centers are proper graphs in a graph-theoretical sense.

Structural competitive learning performs an incremental subgradient descent of the average distortion

$$E(M, \mathcal{Y}; \mathcal{X}) = \frac{1}{2} \sum_{j=1}^{k} \sum_{i=1}^{p} m_{ij} \delta(X_i, Y_j)^2.$$

The component function of $E(M, \mathcal{Y}; \mathcal{X})$ for the $i$-th example $X_i$ can be written as

$$E_i(M, \mathcal{Y}) = \frac{1}{2}\delta(X_i, Y_{i^*})^2,$$

where $Y_{i^*}$ is the winning s-unit when $X_i$ is presented to the network. According to Lemma 9.2, we have

$$\delta(X_i, Y_{i^*})^2 = \|X_i\|^2 - 2X_i \bullet Y_{i^*} + \|Y_{i^*}\|^2.$$

The term $\|X_i\|^2$ is constant as a function of $Y_{i^*}$, and therefore has the null graph as its gradient. The term $\|Y_{i^*}\|^2 = Y_{i^*} \bullet Y_{i^*}$ is continuously Fréchet differentiable with gradient $2Y_{i^*}$. This follows from Theorem 9.1. Finally, the term $S_{X_i}(Y_{i^*}) = X_i \bullet Y_{i^*}$ is generalized differentiable. Hence, for any optimal relabeling $\pi$ of $X_i$ to $Y_{i^*}$ the relabeled graph $X_i^\pi$ is a subgradient from $S_{X_i}(Y_{i^*})$. From the rules of calculus for locally Lipschitz continuous functions (see Theorem 11.5 and 11.6), we obtain

$$Y_{i^*} - X_i^\pi \in \partial E_i(M, \mathcal{Y}).$$

Thus, the resulting structural learning rule

$$Y_{i^*} = Y_{i^*} + \eta\,(X^\pi - Y_{i^*})$$

is in exact accordance with the structural learning rule given in Algorithm 16.

***Applying the Principle Elimination of Competition***

After learning, we may use the trained WTA network as a structural pattern classifier for unseen data. Provided that the learned models are pairwise dissimilar, we can improve utility of the structural classifier by applying the principle *elimination of competition* as proposed in Section 7.4. In addition, we may use elimination of competition during learning in the finetuning phase. Given an input $X$, the WTA classifier rapidly identifies the winning unit $i_*$ without completing computation of any of the $k$ distances $\delta(X, Y_1), \ldots, \delta(X, Y_k)$. Next, the distortion $\delta(X, Y_{i_*})$ and an optimal relabeling $\pi$ of $X$ to $Y_{i_*}$ is determined. Finally, the model $Y_{i_*}$ is updated according to the structural competitive learning rule.

## 14.3 Advanced Models of Structural Competitive Learning

In this section, we sketch how three standard models of unsupervised neural learning machines, namely *Adaptive Resonance Theory*, *Kohonen Maps*, and *Vector Quantization*, can be extended to the domain of attributed graphs.

### 14.3.1 Structural Adaptive Resonance Theory

One shortcoming of simple competitive learning for feature vectors is that convergence to a stable set of cluster centers is not guaranteed. The winning unit may continue to change even when the same input patterns are continuously presented. One way to enforce convergence is to freeze the learned cluster centers by gradually lowering the learning rate $\eta$. But then the system loses its *plasticity*, that is, its capability to react appropriately to new input data. This phenomenon is known as the *Grossberg's stability and plasticity dilemma* [129].

For structural competitive learning, Grossberg's stability and plasticity dilemma holds a fortiori. We present a structural version for the ART1 algorithm and show that the stability and plasticity dilemma is also resolved in the domain of binary graphs.

#### *The Structural ART1 Algorithm*

Output s-units $i$ can be *enabled* or *disabled*. Initially, each model $Y_i$ associated with $i$ is a fully connected binary graph. We call s-unit $i$ *uncommitted* unless $Y_i$ has been updated. For a new input graph $X$ from $\mathcal{G}_{\mathbb{B}}$, the algorithm proceeds as follows:

1. *Enabling.* Enable all the output s-unit.

2. *Similarity Matching.* Determine the winner $i^*$ among the enabled output s-units according to

$$i^* = \arg\max_{i \in [1:k]} \frac{X \bullet Y_j}{\rho_j + \left\| \boldsymbol{Y}_j \right\|_1},$$

where $\left\| \boldsymbol{Y}_j \right\|_1$ denotes the number of bits set in $\boldsymbol{Y}_j$. The quantity $\rho_j > 0$ models small random noise to break ties. Exit if all output s-units are disabled.

3. *Resonate* Test whether $X$ and $Y_{i^*}$ are sufficiently similar by evaluating the boolean expression

$$r = \left[ \frac{X \bullet Y_{i^*}}{\left\| \boldsymbol{X} \right\|_1} > v \right],$$

where $v$ is the *vigilance parameter*. If $r$ is true, there is resonance; go to step 4. Otherwise, model $Y_{i^*}$ is rejected; disable s-unit $i^*$ and go to step 2.

4. *Update rule.* Adjust $Y_{i^*}$ according to the rule

$$\boldsymbol{Y}_{i^*} = \boldsymbol{Y}_{i^*} \wedge \boldsymbol{P}^{\mathsf{T}} \boldsymbol{X} \boldsymbol{P},$$

where $\boldsymbol{P}$ is an optimal rotation from $X$ towards $Y_{i^*}$. The logical operator $\wedge$ deletes all bits in the adjacency matrix $\boldsymbol{Y}_{i^*}$ that are not also in the adjacency matrix $\boldsymbol{P}^{\mathsf{T}} \boldsymbol{X} \boldsymbol{P}$.

As the standard procedure, the structural ART1 either terminates in step 4 and outputs $i^*$, or in step 2 without providing an output. The next results shows that the structural ART1 algorithm solves Grossberg's stability and plasticity dilemma.

***Theorem 14.1***
*Let $\mathcal{X}$ be a finite training sample. After repeated presentations of $\mathcal{X}$, structural ART1 stabilizes to a set of fixed cluster centers. After convergence, we have*

1. $Y_i \in \mathcal{G}_{\mathbb{B}}$
2. $Y_i \neq 0$
3. $i \neq j \;\Rightarrow\; Y_i \neq Y_j$

***Proof*** We embed the problem into the Euclidean space $\mathcal{A}^N$, where $N = n^2$. The vectorized training set is given by

$$\mathcal{X}_V = \bigcup_{X \in \mathcal{X}} \mathcal{V}_{[X]},$$

where $\mathcal{V}_{[X]}$ is the vector representation of the unlabeled graph $[X]$. It is sufficient to consider an arbitrary labeling of each model such that

$$\mathcal{Y}_V = \left\{ \boldsymbol{y}_i \,:\, \boldsymbol{y}_i = \mathrm{vec}(Y_i),\, Y_i \in \mathcal{Y} \right\}$$

is the set of vectorized models. The structural ART1 algorithm turns into the standard ART1 procedure for feature vectors, where the current input vector presented to the network depends on the chosen input graph $X$ and the chosen optimal rotation $\boldsymbol{P}$ in step 4. Stability of standard ART1 follows from [53, 266] and implies stability of structural ART1.

It remains to show properties (1)-(3).

1. Consider binary graphs $X$ and $Y$ without imaginary parts. Clearly, the conjunction $Z = X \wedge Y$ is a binary complex graph. Suppose that $(i, j) \in E(Z)$ is an imaginary edge. Then at least one of the vertices $i$ and $j$ of $Z$ is imaginary. Without loss of generality, we assume that $i \in V(Z)$ is imaginary. Then from $z_{ii} = x_{ii} \wedge y_{ii} = 0$ follows $x_{ii} = 0$ or $y_{ii} = 0$. Again, we may assume without loss of generality that $x_{ii} = 0$. Since $X$ is a real graph without an imaginary part, we have $x_{ij} = 0$ for all $j$. This implies $z_{ij} = x_{ij} \wedge y_{ij} = 0$. This contradicts the assumption that $(i, j)$ is imaginary.

2. Since none of the graphs from $\mathcal{X}$ is zero, an optimal rotation from an input towards a model maps at least real vertices on one another.

3. Follows from [53, 266] using the conversion of structural to standard ART1.

■

### 14.3.2 Structural Kohonen Maps

A *topographic map* is a neighborhood relation-preserving map from the input space to the output space. The input space is a subset $\mathcal{Z}$ of attributed graphs $\mathcal{G}_A$, represented by the training sample $\mathcal{X}$. The output space is usually a set of competitive s-units, geometrically arranged in a one-dimensional line or two-dimensional plane. In a topographic map, structures from a neighborhood in the input space with respect to some metric are mapped to s-units located within a corresponding neighborhood in the output space according to its geometrical arrangement.

For inputs that are represented by feature vectors, different types of topographic maps have been considered. The most common type are *Kohonen maps* [215].

In structural competitive learning, only the models associated with the winner are updated. In addition, updating of the models does not consider the spatial relations among the s-units in the WTA network. Kohonen maps modify structural competitive learning in three ways to enable spatial organization:

1. *Geometrical arrangement of structural units.* Structural units are generally arranged in a one or two-dimensional array. Each s-unit can be thought of being connected to its immediate neighbors by an edge. Typical examples of two-dimensional arrangements of output s-units are lattices or hexagons.

2. *Neighborhood function.* A neighborhood function $\Lambda_i(j)$ with s-unit $i$ as its center indicates the distance from any s-unit $j$ in the array to the center $i$. A typical choice is

$$\Lambda_i(j) = \exp\left(-\frac{d(i,j)}{2\sigma_t^2}\right),$$

where $d$ is a distance measure defined on the output space with a given geometrical layout, and $\sigma_t$ is a time-dependent width parameter that is gradually decreased.

3. *Learning rule*: Suppose that $i^*$ is the winning s-unit of the competition for a given input $X$. Note that the winner $i^*$ of Kohonen competition is defined in the same way as for structural competitive learning. The learning rule is then of the form

$$Y_j \leftarrow Y_j + \eta\Lambda_{i^*}(j)\,(X^{\pi_j} - Y_j)\,,$$

where $\pi_i \in \mathcal{R}_{X,Y_j}$ is an optimal relabeling of $X$ to $Y_j$. The idea behind this rule is to make the winner and s-units from the neighborhood of the winner more likely to respond to inputs that are similar to the current input $X$.

### 14.3.3 Learning Structure Quantization

Vector quantization exploits the structure of the input vectors for data compression [111]. Learning vector quantization is a *supervised* version of vector quantization

proposed by Kohonen [215] to define class regions of the input space.[1] We discuss both techniques of competitive learning in the context of attributed graphs.

In structure quantization, an input space $\mathcal{Z}$ is divided into a number of regions. When presenting a new input graph $X$, the quantizer first determines in which region the graph $X$ lies. Next, $X$ is encoded by a *code word* associated with that region. A code word can simply be an index referring to an appropriate region. Thus, we can store or transmit the much smaller code word than the original input graph at the expense of some distortion. The code words can be decoded to a *quantized* structure (model), which can be viewed as a representation of any input graph associated with the code word. A structure quantizer with minimum encoding distortion is called a *nearest neighbor quantizer*.

The relation to structural competitive learning networks is immediate. The winner $i^*$ defined by

$$i^* = \arg\min_{i\in[1:k]} D\big(X, Y_i\big) \tag{14.4}$$

determines the code word of a given input $X$. Decoding $X$ results in a quantized structure $Y_{i^*}$.

To find an appropriate set of quantized structures in an unsupervised way, we can, for example, apply any structural clustering method that is capable of minimizing the average distortion defined in (8.2).

Learning structure quantization is a supervised technique that uses label information to define class regions. The algorithm differs from structural competitive learning in the way the quantized structures are updated and, optionally, in the choice of the distortion measure $D$. Let $i^*$ denote the winning s-unit for a given input $X$ of class $C_X$ according to the best matching criterion (14.4). The quantized structure $Y_{i^*}$ is adjusted as follows:

$$Y_{i^*} \leftarrow \begin{cases} Y_{i^*} + \eta\big(X^\pi - Y_{i^*}\big) & : \quad C_{i^*} = C_X \\ Y_{i^*} - \eta\big(X^\pi - Y_{i^*}\big) & : \quad C_{i^*} \neq C_X \end{cases},$$

where $\pi \in \mathcal{R}_{X,Y}$ is an optimal relabeling of $X$ towards $Y_{i^*}$ and $C_{i^*}$ is the label of the region represented by $Y_{i^*}$. No other quantized structures are modified.

## 14.4 Conclusion

In this chapter, we extended unsupervised competitive learning to the domain of attributed graphs and discussed structural versions of Adaptive Resonance Theory, Kohonen maps, and Vector Quantization. As in the previous chapters, the focus

1. Although learning vector quantization is a supervised learning technique, we consider it in league with unsupervised neural learning machines, because of its relation to the best matching criterion realized by WTA networks.

was on basic principles rather than on an exhaustive mathematical analysis of the proposed systems.

We conclude the last of three chapters on structural neural learning machines with an outlook for further research directions. Of striking interest are the convergence properties and overall dynamics of structural competitive learning. This might be a challenging problem since even for the standard Kohonen map, convergence properties have only been proved for limited settings. Another objective is concerned with the extent to which a structural Kohonen map preserves statistical properties of the data as a nonlinear approximation of principal component analysis. A third objective is to construct other unsupervised structural learning machines based on subgradient techniques. A first example is the structural $K$-means algorithm [183].

# 15 Experiments

This chapter serves to illustrate that structural neural learning machines can – at least in principle – generalize over a given training sample. The focus is on checking the theoretical framework for its potential applicability, rather than providing a comprehensive study. Consequently, results are presented for a selection of more or less simple problems with an emphasis on issues related to learnability, rather than experimental exhaustion. Sections 15.1 and 15.2 apply a structural neural learning machine to function approximation problems, Section 15.3 to classification, and Section 15.4 to cluster analysis.

We used the ACS algorithm to approximate the structural inner dot products. The algorithms were implemented in Java using `JDK` 1.2. All experiments were run on a multi-server `Sparc SUNW Ultra-4`.

## 15.1 Function Approximation I

The aim of our first experiment is to check whether the theory developed in Chapters 9-13 also works in a practical setting. Moreover, we want to know whether an s-network can generalize. To this end, we tested a multi layer s-network on a function approximation problem.

***Problem Formulation***

Let $X$ be a random weighted graph. The task is to learn the structural function

$$f : \mathcal{G}_{\mathbb{U}}^{20} \to [0,1], \quad X \mapsto \frac{\sum_{(i,j)\in E(X)} |\mu_X(i,j)|}{|X|\big(|X|-1\big)},$$

where $\mathcal{G}_{\mathbb{R}}^{20}$ denotes the set of all random weighted graphs $X$ of order $|X| \leq 20$ with weights drawn from a $N(0,1)$ Gaussian distribution with zero mean and standard deviation one. Null attribute is $\epsilon = 0$. Evaluation of $f$ at $X$ measures the weighted edge density of a graph $X$.

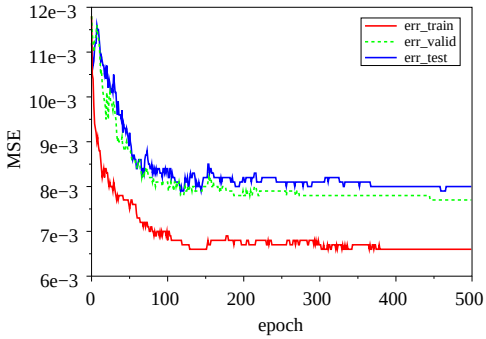



**Figure 15.1** Result for estimating the weighted edge density of random weighted graphs. Shown are the training (solid red line), validation (dashed green line), and test error (solid blue line) as a function of the number of epochs.

### *Evaluation Procedure*

1. *Setting of s-network*: We used a two layer s-network consisting of 10 s-units with *tanh*-activation function and one linear output unit. Each weight graph in the first hidden layer was of order 3. We set the initial learning rate $\eta$ and the momentum term $\alpha$ to 0.1. The learning rate was slowly decreased at each iteration step according to the rule $\eta \leftarrow 0.99 \cdot \eta$. We trained the network using the standard mean sum-of-squares error (`MSE`) function.

2. *Training sample*: To compile the data, we generated $1,250$ graphs from a uniform distribution over the set $\mathcal{G}_{\mathbb{R}}^{20}$. The sample was divided into a training, validation, and test set composed of 500, 250, and 500 weighted graphs, respectively.

### *Numerical Results*

Figure 15.1 plots the training, validation and test error against the number of passes through the training set. The result shows that

- the s-network is robust against approximations of the structural dot product. Since determining the structural dot product is NP-complete, this observation is important for practical issues;
- the theory developed in Chapters 9-13 can be applied to practical problems. As shown by the curve of the training error, approximating the subgradient learning rule minimizes the structural MSE function;
- the s-network generalizes over the training sample.

In addition, all three error rates have converged by maintaining a small oscillation around $E_{train} = 0.0066$, $E_{valid} = 0.008$, and $E_{test} = 0.0077$. Since $\eta \rightarrow 0$, the oscillations are due to the randomness of the approximate solution of the structural dot product. From the plot we see that training, validation, and test error all

converge at approximately the same time, that is, no over-fitting occurs. We assume that approximating the structural dot products introduces random noise, which prevents over-fitting of the training data.

## 15.2 Function Approximation II

In this experiment, we applied a multi layer s-network to a regression problem in *quantitative structure-property relationship* (QSPR) analysis. This experiment has been conducted in a diploma thesis by P. Bönke [39].

***Problem Formulation***

The QSPR problem consists in predicting the boiling points of saturated hydrocarbons up to decanes. Boiling points are in the range of $[-161.5°C, 219°C]$. The dataset is described in [306]. It consists of all saturated hydrocarbon structures up to 10 $C$ atoms with known boiling points giving a total of 531 structures.

***Graph-Based Representation***

Each hydrocarbon structure was represented by an attributed graph. Vertices represented $C$ atoms. Bonds between $C$ atoms were represented by edges. We assigned the attribute value 1 to all edges and the value 0 to all other items.

***Evaluation Procedure***

1. *Setting of s-network*: We used a two layer s-network (SN) consisting of seven structural units and a single output unit. The weight graphs associated with the s-units were of order $n = 10$. The s-units used the hyperbolic function $g(u) = \tanh(u)$ as activation function and the output unit the identity function. The learning rate was initially set to 0.1 and slowly decreased at each iteration step according to the rule $\eta \leftarrow 0.99 \cdot \eta$. We set the momentum term to 0.1. Learning was terminated after 500 epochs. The SN minimized the squared error function. In contrast to all other experiments, we used the Graduated Assignment algorithm [116] for approximating the structural dot product.
   Note that the goal was to show that SN can generalize reasonably well. Hence, neither model selection nor other systematic experimentation with varying data representation and graph matching algorithms has been conducted.
2. *Selected learning algorithms*: We compared the SN with the following four learning algorithms:

| | |
|---|---|
| RecNN | Recursive cascade correlation neural network |
| $STK_1$ | Support vector regression using a string kernel |
| $TK_1$ | $STK_1$ with subsequent application of an RBF kernel |
| $TK_2$ | Support vector regression using a tree kernel |
| $TK_2$ | $TK_2$ with subsequent application of an RBF kernel |

For a detailed description of the recursive neural network and the support vector machines, we refer to [260].

3. *Protocol*: To provide a fair comparison, we used the same protocol as in [260]. We focused on a subset of our dataset consisting of all 150 acyclic alcanes, and performed a 10-fold cross validation protocol to estimate the prediction error.

***Numerical Results***

Table 15.2 summarizes the results. As in our first experiment, we see that structural neural learning machines are able to generalize even when using approximate solutions to the structural dot products. Although structural neural learning machines are in its infancy, generalization of SN performs on an average level.

As expected, the results indicate that the tree kernels better capture the relevant structural information for predicting boiling points of acyclic alcanes as both string kernels. On the other hand, SN worked in the more complex domain of attributed graphs, rather than exploiting the acyclic structure of alcanes. Approximate solutions for the more complex problem resulted in a loss of relevant structural information. Note that the loss is more controlled than for string kernels. Since acyclic alcanes are simple structures, which can be represented by unlabeled trees, RecNN and tree kernel methods are most appropriate for predicting boiling points.

We emphasize that SN becomes an interesting choice for noisy structures where real-valued attribute vectors are assigned to the vertices and the edges. For those type of graphs, recurrent neural networks are not directly applicable, and fully general graph kernels are computationally inefficient. One problem with support vector regression in this experiment is that at least 1/3 of the structures from the training set are support vectors. This is computationally inacceptable for more complex problems with large training sets consisting of arbitrary graphs.

| | SN | RecNN | $STK_1$ | $STK_2$ | $TK_1$ | $TK_2$ |
|---|---|---|---|---|---|---|
| $E_{train}$ | $3.39 \pm 0.49$ | $2.15 \pm 0.12$ | $2.52 \pm 0.41$ | $2.16 \pm 0.32$ | $3.70 \pm 0.21$ | $2.43 \pm 0.24$ |
| $E_{test}$ | $4.47 \pm 1.28$ | $2.86 \pm 0.74$ | $7.03 \pm 1.84$ | $5.49 \pm 1.64$ | $4.70 \pm 1.06$ | $2.93 \pm 0.92$ |
| $N$ | 7.0 | 140.1 | 85.4 | 110.4 | 113.2 | 52.5 |

**Table 15.1** Results for predicting boiling points of 150 acyclic alcanes up to 10 $C$ atoms. The quantities $E_{train}$ and $E_{test}$ refer to the average absolute training and generalization error. By $N$ we denote the average number of hidden units for SN, RecNN and of support vectors for $STK_1$, $STK_2$, $TK_1$, $TK_2$.

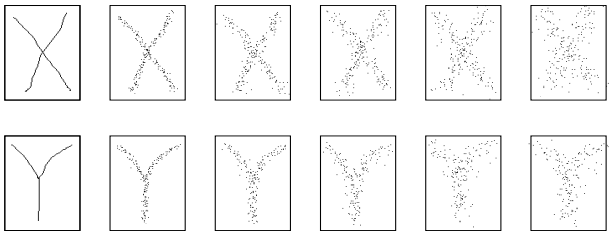


**Figure 15.2** Images of handwritten characters $'X'$ and $'Y'$. The first column shows the model images. Column 2-6 are samples of corrupted images with increasing noise level $\sigma = 2, 4, 6, 8, 10$. For the sake of presentation no rotation is shown.

## 15.3 Classification

In this test series, we investigated how generalization of a single s-unit depends on structural variation and noisy attributes. For this, we considered the problem of classifying noisy copies of handwritten characters $'X'$ and $'Y'$.

### *Problem Formulation*

Consider the two handwritten characters $\mathcal{C} = \{C_X, C_Y\}$ shown in the first column of Figure 15.2. The model characters were drawn using an X windows interface. The black pixels of each model were expressed as a set of points in the 2D plane.

Given a corrupted version $X$ of a model character $C \in \mathcal{C}$, the goal was to identify the model $C$ that generates the data $X$. Corrupted inputs were generated from the handwritten characters by a similar perturbance model as in Section 7.5.2:

1. Select model character $C \in \mathcal{C}$ and copy to $X$.
2. Randomly rotate $X$ around its barycenter.
3. Delete each point of $X$ with 10% probability.
4. Add Gaussian noise $N(0, \sigma)$ to the coordinates of each point of $X$.

Figure 15.2 shows examples of corrupted data images generated by the above procedure for standard deviations $\sigma \in \{2, 4, 6, 8, 10\}$.

### *Graph-Based Representation*

Given an input image as a point set of black pixels, we randomly selected points so that the pairwise normalized distances between the chosen points were larger than a given threshold $\theta$.[1] The resulting point set $\mathcal{P}$ was transformed to a fully

1. The value $\theta$ depends on the implementation. To reproduce the experiments, $\theta$ should be chosen so that the estimated expected cardinality of the point sets equals the values in Table 15.2.

| | 2.0 | 4.0 | 6.0 | 8.0 | 10.0 |
|---|---|---|---|---|---|
| mean | 16.8/20.5 | 21.7/25.9 | 27.2/32.9 | 31.6/38.2 | 36.0/43.3 |
| var | 0.9/1.1 | 3.7/4.1 | 4.8/6.4 | 6.8/7.4 | 7.4/9.4 |
| $\Delta$ | 6.0/6.0 | 11.0/12.0 | 12.0/16.0 | 17.0/14.0 | 15.0/19.0 |

**Table 15.2** Structural variations of the character dataset. Shown are the mean, variance, and the maximal difference $\Delta = \max - \min$ of vertices for varying noise levels $\sigma$. For each entry $x/y$ the numbers $x$ and $y$ refer to character $'X'$ and $'Y'$, respectively.

connected attributed graph. Vertices $v(\boldsymbol{p})$ represent the points $\boldsymbol{p} \in \mathcal{P}$, and edges an abstract line between the corresponding points. To each vertex $v(\boldsymbol{p})$, we assigned a three-dimensional attribute vector $\boldsymbol{a} = (a_1, a_2, a_3)$. Attribute $a_1$ is the normalized distance of point $\boldsymbol{p}$ to the barycenter of the input image, attribute $a_2$ is the average distance from point $p$ to all other points $\boldsymbol{q} \in \mathcal{P}$, and attribute $a_3$ is the variance of the distances from point $p$ to all the other points $\boldsymbol{q} \in \mathcal{P}$. Each edge was colored by the normalized distance between the corresponding points. To each edge connecting vertex $v(\boldsymbol{p})$ and $v(\boldsymbol{q})$, we assigned a two-dimensional attribute vector $\boldsymbol{b} = (b_1, b_2)$. The first attribute is the normalized distance between $\boldsymbol{p}$ and $\boldsymbol{q}$. The second attribute measures the normalized distance from the abstract line passing through $\boldsymbol{p}$ and $\boldsymbol{q}$ to the barycenter of the input image.

By construction, each graph represents an input image invariant to rotation, translation, and scaling. Table 15.2 summarizes the structural variation of a random dataset consisting of 50 corrupted input images from each category giving a total of 100 examples.

***Evaluation Procedure***

1. *Setting of s-network*: For evaluation, we used a single s-unit (SU) with logistic sigmoid activation function. The weight graph $W$ was of order $|W| = 20$. The learning rate was initially set to 0.1 and slowly decreased at each iteration step according to the rule $\eta \leftarrow 0.99 \cdot \eta$. We set the momentum term to 0.2. The s-network minimized the cross-entropy error function.

2. *Selected classifiers*: We compared the s-unit classifier with the following four support vector (`SV`) classifiers:

   $(SV_1)$ Pairwise proximities classifier proposed by [140].

   $(SV_2)$ $SV_1$-classifier using an RBF kernel.

   $(SV_3)$ support vector classifier using the structural dot product [170].

   $(SV_4)$ $SV_3$-classifier with RBF kernel applied on structural Frobenius norm.

   The pairwise proximity classifiers $SV_1$, $SV_2$, and $SV_4$ operated on approximations of the structural Frobenius metric as a proximity measure. The $SV_3$-classifier used approximations of the structural dot product.

| $\sigma$ | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|
| $SU$ | 0 | 0 | 1 | 2 | 2 |
| $SV_1$ | 0 | 0 | 6 | 8 | 29 |
| $SV_2$ | 0 | 0 | 4 | 6 | 23 |
| $SV_3$ | 0 | 0 | 5 | 4 | 29 |
| $SV_4$ | 0 | 0 | 5 | 4 | 20 |

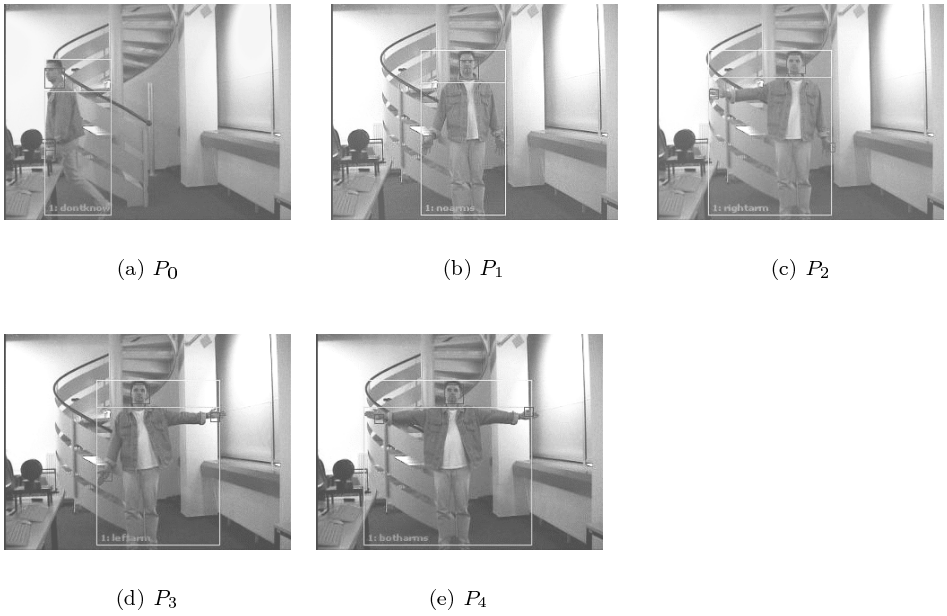
**Table 15.3** Classification results for synthetic characters. Shown are the estimated predictive errors (in %) for the s-unit ($SU$) and the four $SV_i$-classifiers for varying noise levels $\sigma$. Error estimates are from a 10-fold cross-validation protocol. Results of the $SV_i$ classifiers were taken from [170]. Standard deviation from the estimated prediction errors for the proposed $SU$-classifier was 1.5% at most in all five experiments. Standard deviations for the other classifiers have not been reported.

3. *Training sample*: To provide a fair comparison, we used the same setup as in [170]. For each noise level $\sigma \in \{2, 4, 6, 8, 10\}$, we generated a well-balanced training sample consisting of 50 examples from each category, and performed a 10-fold cross validation protocol to estimate the prediction error.

***Numerical Results***

Table 15.3 presents the estimated prediction errors of all five classifiers. As expected, the estimated predictive error increased for all classifiers with increasing noise level. Prediction of the proposed $SU$-classifier was robust to strong noise and structural variation.

To point to an attractive feature of s-networks, consider the classification problem at noise level $\sigma = 10$. As shown by Table 15.2, the average order of all graphs in the training sample is about 40. In order to classify an unseen image, the $SV_i$-classifiers approximated a number of structural proximities where both graphs under consideration were composed of about 40 vertices. In contrast, the $SU$-classifier approximated exactly one structural dot product of an input graph of expected order 40 and a weight graph of order 20. Hence, we may expect that the $SU$-classifier is much faster than the $SV_i$-classifiers. In addition, the $SU$-classifier performs *order reduction* to relevant structural parts of the data.



(a) $P_0$ (b) $P_1$ (c) $P_2$

(d) $P_3$ (e) $P_4$

**Figure 15.3** Sample images of arm postures.

## 15.4 Clustering

One important problem in computer vision and human-machine interaction is to sense gestures and postures of people for directing computers or robots. This problem arises in many areas such as e.g. a policemen giving signs to regulate the traffic, directing a crane, navigating a vehicle into a parking lot, or medical monitoring of patients in a hospital or nursing home.

In this experiment, we applied the structural competitive learning algorithm to learn the class structure of arm postures as shown in Figure 15.3.

### *Dataset*

We considered five different categories of 235 arm postures:

($P_0$) Unknown
($P_1$) NoArms
($P_2$) RightArm
($P_3$) LeftArm
($P_4$) BothArms

Each category refers to the lifted arms of a person. Table 15.4 provides a description of the five categories, where 0 denotes an arm pointing downwards and 1 denotes an

| category | right arm | left arm | category distribution |
|---|---|---|---|
| $P_0$ | $*$ | $*$ | 37 |
| $P_1$ | 0 | 0 | 50 |
| $P_2$ | 1 | 0 | 50 |
| $P_3$ | 0 | 1 | 50 |
| $P_4$ | 1 | 1 | 48 |

**Table 15.4** Description of arm postures and their distributions.

arm pointing straight out from the shoulders parallel to the floor. By $*$ we denote the *don't care* symbol.

***Graph-Based Representation***

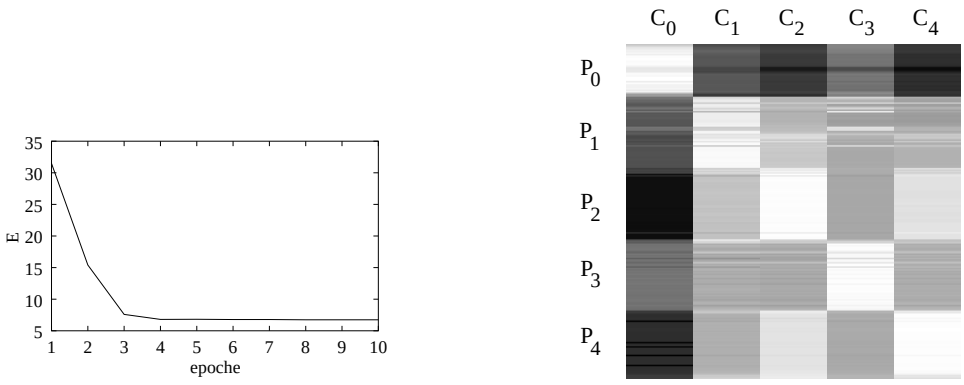
The graph-based representations of the images was provided by [30]. Each image was obtained by automatically localizing a person in video data from a camera. The localized person was enclosed in a bounding box. Position of body parts, like head and hands of the person, were identified by skin color. We transformed each image to a fully connected attributed graph. The vertices represent the upper left and right corner of the bounding box and the identified body parts (head and hands). The order of graphs varied between 3 and 5. Variation of the order was caused by segmentation errors. Vertices were assigned a three-dimensional binary attribute vector $\boldsymbol{a} = (a_1, a_2, a_3) \in \mathbb{B}^3$. Attribute vector $(1, 0, 0)$ represents the left corner, $(0, 1, 0)$ the right corner, and $(0, 0, 1)$ a body part. Edge attributes were the distances between the corresponding components in the image.

***Evaluation Procedure***

1. *Setting of s-network*: We used a structural competitive learning network consisting of 5 inhibitory connected s-units. The weight graphs associated with the s-unit were of order 5. The learning rate was initially set to 0.1 and slowly decreased at each iteration step according to the rule $\eta \leftarrow 0.99 \cdot \eta$.
2. *Initialization*: Since the performance of simple competitive learning depends on a proper initialization of the models, we may expect the same behavior from the structural competitive network. Therefore, we randomly selected five images from the dataset, one from each category to initialize the models.

***Numerical Results***

Figure 15.4 presents the results. From subfigure (a) we see that the structural competitive learning rule minimizes the average distortion $E(M, \mathcal{Y}, \mathcal{X})$ until convergence after about 5 epochs. The quality of the cluster structure discovered by




(a) (b)

**Figure 15.4** Results of clustering arm postures. *Subfigure (a)*: Shown are the average distortion as a function of the number of epochs. *Subfigure (b)*: Shown is the pairwise distance matrix between the learned models representing the clusters $C_1, \ldots, C_4$ and the input samples ordered by arm posture $P_0, \ldots, P_4$. Brighter shadings indicate closer structural Frobenius distance.

structural competitive learning is visualized in subfigure (b). Table 15.5 shows the error rate of the nearest neighbor classifiers using the models after initialization ($\texttt{NN}_0$) and clustering ($\texttt{NN}_1$). The results presented in Table 15.5 indicate that learning shifts the initial models closer to the centers of the respective categories.

Table 15.6 shows the confusion matrices before and after clustering. The $i$th row of both confusion matrices shows the distribution of members of class $P_i$ into clusters $C_1, \ldots, C_4$ using the nearest neighbor rule applied on the structural Frobenius norm. Misclassification of images from category RightArm as images from category NoArms are caused by an initial model representing category NoArms with a slightly lifted right arm. The same holds for misclassification of images from category LeftArm as images from category BothArms. Here, the model representing category LeftArm is an image with the left arm lifted and a

| | $P_0$ (37) | $P_1$ (48) | $P_2$ 48) | $P_3$ (48) | $P_4$ (48) | $E$ [%] |
|---|---|---|---|---|---|---|
| $\texttt{NN}_0$ | 3 | 15 | 44 | 40 | 0 | 44.5 |
| $\texttt{NN}_1$ | 3 | 0 | 0 | 1 | 0 | 1.7 |

**Table 15.5** Classification results of arm postures. Shown are the number of misclassified samples by $\texttt{NN}_0$ and $\texttt{NN}_1$ for each class and the total percentage error rate $E$. The numbers in parenthesis indicate the number of images of the corresponding class.

| | $C_0$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_0$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $P_0$ | 30 | 5 | 2 | 0 | 0 | 35 | 0 | 0 | 2 | 0 |
| $P_1$ | 0 | 25 | 25 | 0 | 0 | 0 | 43 | 0 | 7 | 0 |
| $P_2$ | 0 | 0 | 50 | 0 | 0 | 0 | 0 | 49 | 0 | 1 |
| $P_3$ | 0 | 0 | 0 | 12 | 38 | 0 | 3 | 0 | 45 | 2 |
| $P_4$ | 0 | 0 | 0 | 0 | 48 | 0 | 0 | 5 | 0 | 43 |

**Table 15.6** Confusion matrices. Each row shows the distribution of all samples of a posture $P_i$ into clusters. *Left table*: Before clustering. *Right table*: After clustering.

slightly raised right arm. To summarize, as in supervised learning, the subgradient competitive learning rule minimizes the average structural distortion to recover the cluster structure of the data.

## 15.5 Conclusion

In this chapter, we presented and discussed experiments on structural learning problems. The key results are:

- Subgradient learning rules can minimize structural error functions.
- The learned s-networks can generalize.
- Learning and generalization is robust against approximations of the structural dot product.

The first result confirms the theory developed in Chapters 9-14. The second result justifies further research on structural neural learning machines. Finally, the third result makes structural neural learning machines amenable to practical problems.

The focus of this chapter was to provide an empirical justification of the theory developed in the second part of this work. Therefore, simple learning problems were selected to illustrate the behavior of structural neural learning machines. To establish structural neural learning machines as an alternative solution for conventional techniques in structural pattern recognition, extensive experimentation is imperative.

# 16 Conclusion

The main contribution of this thesis is the construction of structural neural learning machines for arbitrary attributed graphs. Its theoretical foundation is a chief mathematical result that dwarfs all other contributions of this thesis: We are now in the position to pull the powerful machinery of differential analysis to the domain of attributed graphs. As a consequence, we believe that structural neural learning machines are only a first example of potential applications that exploit gradient-like information. Basically, the theory applies to any problem that aims at minimizing a locally Lipschitz continuous structural function on attributed graphs. But the field of structural neural learning machines is already large enough to make an exhaustive treatment infeasible. Therefore, we provided basic principles and mechanisms of structural neural learning machines and hope that, in such, a solid theoretical ground has been laid for exploring advanced issues.

Construction of structural neural learning machines deals with both fundamental problems of structural pattern recognition, namely computational and analytical intractability. The first part of this thesis was concerned with the issue of computational intractability arising in the feedforward pass to determine the output of a structural unit. Part II was devoted to the issue of analytical intractability arising in the formulation of structural learning rules to appropriately adjust the weight graphs. The partial results that contribute to the overall thesis are also interesting on their own accord, and are summarized as follows:

***Part I***

- We established a generic view of graph matching problems with a $\mathfrak{p}$-closed domain in terms of a maximum weight clique problem in an association graph. This eliminates a common misconception that clique search is a special technique for solving a narrow class of graph matching problems. In addition, the scattered plethora of structural proximity measures, including the graph-edit distance, are now summarized under a pure graph-theoretical umbrella of maximum clique weight. Hence, any solution to the maximum weight clique problem is also a solution to $\mathfrak{p}$-graph matching problems.[1]

1. As opposed to the literature, the maximum weight clique problem asks for a clique with maximum total vertex *and* edge weight.

- In a mathematical analysis, we derived an equivalent continuous quadratic program of the maximum weight clique problem to enable continuous optimization methods. The attractive feature of the proposed quadratic formulation is that its constrained local minima are in one-to-one correspondence with the solutions of the original problem.
- To approximately solve the quadratic formulation of the maximum weight clique problem, we proposed a selective attention control system (ACS). The proposed system ensures convergence to feasible solutions, requires no tuning of system parameters, and adapts its system parameters optimally during run time. In particular, the latter feature makes the ACS algorithm extremely fast and suitable for large scaled problems.
- Existing neural network techniques failed as a practical alternative for solving the graph isomorphism problem. We proposed a two-stage neural network approach that outperformed the fastest solutions reported in the neural network literature by a factor of up to $10^5$ for standard isomorphism problems. In addition, the proposed system significantly outperformed known existing methods to inexact graph isomorphism problems.
- We presented a real-time neural network for combinatorial optimization problems that can be interrupted at any point in time and provide meaningful results. In addition, the quality of results improves with increasing computation time.
- To improve the utility of template matching procedures, we developed a self-organizing WTA classifier that combines techniques from anytime computing with the principle *elimination of competition*. The system dynamically allocates computing resources to promising input-model pairs in a self-organizing manner. This approach results in a fast classifier suitable for recognition problems with pairwise dissimilar models.

***Part II***

- As a counterpart of the standard dot product, we formulated the structural dot product. We showed that the structural dot product has the same geometrical properties as the standard dot product. The results lead to another geometry of structures.
- Based on the generalized and structural Frobenius metric induced by the structural dot product, we developed the theory of differential analysis for structural functions on attributed graphs. The most important result is that the gradient of a smooth structural function is a well-defined graph pointing in the direction of steepest increase.
- Since the structural dot product is nonsmooth, we accommodated techniques from nonsmooth analysis to minimize structural functions built upon the structural dot product. We showed that the structural dot product is regular and therefore smooth almost anywhere.

- For two-category problems, a single structural unit determines two decision regions, one for each class. We showed that the decision surface implemented by a structural unit is composed of hyperplane segments.
- We extended Rosenblatt's perceptron to attributed graphs and proved a weak form of the structural perceptron convergence theorem.
- Three results concerning the representational capabilities of structural multi layer feedforward networks have been presented. We showed that an s-network can represent any dichotomy on binary graphs, approximate any decision region to arbitrary accuracy, and approximate any smooth structural function on a compact subset to arbitrary accuracy.
- The back-propagation algorithm is extended to a subgradient back-propagation algorithm to determine the derivatives of a structural error criterion with respect to the weights.
- We extended WTA learning, Self-Organizing Maps, Vector Quantization, and Adaptive Resonance Theory to attributed graphs.

### Outlook

In the course of this thesis, we addressed many open questions. In the following, we will discuss four topics, which I consider to be most interesting for further research in the area of structural neural learning machines.

- *Forgettable ACS models*: Since the ACS procedure requires no tuning of system parameters and optimally adapts its parameters during run time, it might serve as a manageable and fast alternative to existing methods for clique problems. In particular, introducing intelligent schemes of a forgetting mechanism combined with selective attention may result in a useful solution for large-scaled problems, because forgetting reduces the problem size to relevant parts. In addition, transferring the theoretical results of ACS to other combinatorial optimization problems might be beneficial.
- *Hierarchical decomposition of graphs*: One approach to reduce the complexity of graph matching problems, to improve the solution quality, and to provide more meaningful results is to hierarchically decompose the graphs into concepts. This is a challenging problem, because even for sparse graphs like chemical structures, hierarchical decomposition is not necessarily unique. In addition, reduction with loss of structural information yields fast solutions with degraded solution quality [241, 364]. Conversely, reduction without loss of structural information can be computationally inefficient [319]. Satisfactory results have been obtained for simple toy examples [257] and network decompositions [258]. The components of network decompositions, however, are meaningless in the sense that they do not necessarily represent a semantic part of the object described by a graph. In computer vision, hierarchical decomposition by pyramidical structures is one direction of research [196]. Attempts to exploit hierarchical segmentations to

compare structural descriptions can be found in [30, 113].

- *Differential Analysis of Structural Functions*: The theory of differentiable structural functions is a key result of this thesis, which enables us to formulate and analyze structural neural learning machines. Expanding differential analysis of structural functions is not only interesting from a purely mathematical point of view, but may also provide powerful techniques for intelligent data analysis in the domain of graphs.
- *Structural Neural Learning Machines*: A number of problems concerning structural neural learning machines remains open.
  - *Theoretical issues*: Of particular interest are results for and related to the learning capability of structural networks. Other important issues are concerned with approximation properties of feedforward s-networks, regularization, and convergence of structural self-organizing maps. Note that this list is far from being complete.
  - *Optimization methods*: Since any optimization method for minimizing a structural error function evaluates the structural dot product, statements on practical issues about nonsmooth optimization methods do not directly transfer to the structural domain. Therefore, devising minimizers tailored to the specific needs of structural error functions and adopting sophisticated approaches like bundle methods play a striking role in structural learning.
  - *Experimental validation*: In the second part of this thesis, the emphasis was placed on theoretical issues. First experiments served to underpin that structural neural learning machines can learn. To justify and establish the proposed approach as an alternative tool in structural pattern recognition, extensive empirical validation is imperative.

# App. A Algorithms

This chapter presents pseudocode of known algorithms that have been implemented for the purpose of comparison or application, and their default parameter settings. Some of the algorithms have been slightly modified for solving the MWCP.

Unless otherwise stated, $X = (V, E, \boldsymbol{X})$ is a normalized graph of order $|X| = n$. The function to be minimized is of the form

$$\begin{aligned} &\text{maximize} && E(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{x} - \boldsymbol{h}^{\mathsf{T}}\boldsymbol{x} \\ &\text{subject to} && \boldsymbol{x} \in \mathbb{B}^n, \end{aligned}$$

where $\boldsymbol{W} = \lambda(\boldsymbol{O}_{\boldsymbol{X}} - \mu\boldsymbol{A})$ and $\boldsymbol{h} = \lambda\boldsymbol{d}_{\boldsymbol{X}}$. The matrix $\boldsymbol{O}_{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{D}_{\boldsymbol{X}}$ denotes the off-diagonal matrix of $\boldsymbol{X}$.

For all $\boldsymbol{x} \in \mathbb{U}^n$, we define

$$\omega(\boldsymbol{x}) = \begin{cases} \omega(C_{\boldsymbol{x}}) & : \quad \boldsymbol{x} \text{ represents a clique } C_{\boldsymbol{x}} \\ 0 & : \quad \text{otherwise.} \end{cases}$$

## A.1 Hopfield Models

In this section we describe the following algorithms and their parameter settings

1. Selective Attention Control System
2. Repeated Attention Control
3. Steepest Descent
4. Exterior Point Penalty Networks
5. Stochastic Steepest Descent
6. Mean-Field Annealing
7. Potts Mean-Field Annealing

For the Selective Attention Control System we only present the parameter settings. Algorithms (3)-(6) were originally proposed to solve the MCP and systematically investigated by Jagota [167]. We extend the networks for the MWCP. Parameter settings for algorithms (3)-(6) are largely taken from [167]. Potts Mean-Field Annealing replaces the dynamics of Mean-Field Annealing by the Potts dynamics

and uses a quenched annealing schedule.

### A.1.1 Selective Attention Control System

The algorithm is described in Section 4.5.2. The parameters were selected as follows: We set $\varepsilon = 10^{-10}$. The initial activation for each unit $i$ was of the form

$$u_i = \lambda(\deg_{\mathrm{w}}(i) + x_{ii}).$$

### A.1.2 Repeated Attention Control

Let $ACS(\boldsymbol{u})$ denote the clique weight returned by ACS given the initial activation $\boldsymbol{u}$. Algorithm 17 describes the RAC procedure.

**Algorithm 17** (RAC — Repeated Attention Control)

**Input:**

| | | |
|---|---|---|
| $X$ | – | normalized graph $X = (V, E, \boldsymbol{X})$ |
| $\varepsilon$ | – | small positive constant |
| $R$ | – | number of repeated runs of ACS |

**Initialization:**

set $\omega = 0$

**Procedure:**

**for** $i = [1 : R]$

set $\boldsymbol{u} \in \mathbb{U}^n$

$\omega = \max(\omega, ACS(\boldsymbol{u}))$

**Output:** $\omega$

**Parameter Settings:**

| $\varepsilon$ | $R$ |
|---|---|
| $10^{-10}$ | $n/10$ |

We initialized $\boldsymbol{u}$ in the $i$-th run as follows: Let $k$ denote the vertex with the $i$-th largest value $\deg_{\mathrm{w}}(k) + x_{kk}$. Then the initial activation of unit $j$ is given by

$$u_j = \begin{cases} 1 & : \quad j = k \\ (\deg_{\mathrm{w}}(k) + x_{kk})/n + \nu & : \quad j \in N(k) \\ -10 & : \quad \text{otherwise} \end{cases},$$

where $\nu$ is random noise drawn from a uniform distribution over the interval $[-0.005, +0.005]$.

Note that RAC uses the principle of selective attention from the very start of each run. The active units are the selected $k$ and its neighbors. All other units are passive. Thus, attention is shifted to the neighborhood of a selected unit.

### A.1.3 Steepest Descent

**Algorithm 18** (SD — Steepest Descent)

**Input:**

$X$ – normalized graph $X = (V, E, \boldsymbol{X})$

**Initialization:**

set $\boldsymbol{x} = \boldsymbol{x}_0 \in \mathbb{B}^n$

set $\mu > |X|$

set $\boldsymbol{W} = \boldsymbol{O}_{\boldsymbol{X}} - \mu \boldsymbol{A}$

set $\boldsymbol{h} = \boldsymbol{d}_{\boldsymbol{X}}$

**Procedure:**

**repeat**

**for** $i = [1 : n]$

$\Delta E_i = (1 - 2x_i)(\boldsymbol{W}\boldsymbol{x} + h_i)$

$i_* = \arg\min \Delta E_i$

**if** $\Delta E_{i_*} < 0$ **then** $x_{i_*} = 1 - x_{i_*}$

**until** $\Delta E_{i_*} \geq 0$

**Output:** $\omega(\boldsymbol{x})$

**Parameter Settings:**

| $\mu$ | $\boldsymbol{x}_0$ |
|---|---|
| $2n$ | $\boldsymbol{e}_n$ |

### A.1.4 Exterior Point Penalty Networks

**Algorithm 19** (EPP — Exterior Point Penalty Network)

**Input:**

$X$ – normalized graph $X = (V, E, \boldsymbol{X})$

$\mu_0$ – initial value of penalty parameter

$\alpha$ – rate at which the temperature is increased

**Initialization:**

set $\boldsymbol{x} = \boldsymbol{x}_0 \in \mathbb{B}^n$

set $\mu = \mu_0$

set $\boldsymbol{h} = \boldsymbol{d}_{\boldsymbol{X}}$

**Procedure:**

**repeat**

$\boldsymbol{W} = \boldsymbol{O}_{\boldsymbol{X}} - \mu \boldsymbol{A}$

run procedure of SD

$\mu = \alpha\mu$

**until** $\mu > |X|$

**Output:** $\omega(\boldsymbol{x})$

**Parameter Settings:**

| $\mu_0$ | $\alpha$ | $\boldsymbol{x}_0$ |
|---|---|---|
| $1/8$ | $2$ | $\boldsymbol{e}_n$ |

### A.1.5 Stochastic Steepest Descent

**Algorithm 20** (SSD — Stochastic Steepest Descent)

**Input:**

$X$ – normalized graph $X = (V, E, \boldsymbol{X})$

$R$ – number of repeated runs of stochastic SD

**Initialization:**

set $\mu > |X|$

set $\boldsymbol{W} = \boldsymbol{O_X} - \mu \boldsymbol{A}$

set $\boldsymbol{h} = \boldsymbol{d_X}$

set $\omega = 0$

**Procedure:**

**for** $r = [1 : R]$

set $\boldsymbol{x} = \boldsymbol{x}_0 \in \mathbb{B}^n$

**repeat** (*stochastic SD*)

**for** $i = [1 : n]$

$\Delta E_i = (1 - 2x_i)(\boldsymbol{W}\boldsymbol{x} + h_i)$

**if** $\Delta E_i > 0$ **then** $\Delta E_i = 0$

$Z_i = \Delta E_1 + \cdots + \Delta E_i$

**if** $Z_n > 0$ **then**

draw $\rho$ from a uniform distribution over $[0, 1[$

select $i_*$ with $Z_{i-1}/Z_n \leq \rho < Z_i/Z_n$

$x_{i_*} = 1 - x_{i_*}$

**until** $Z_n == 0$

$\omega = \max(\omega, \omega(\boldsymbol{x}))$

**Output:** $\omega$

**Parameter Settings:**

| $\mu$ | $\alpha$ | $\boldsymbol{x}_0$ |
|---|---|---|
| $n$ | $n/5$ | $\boldsymbol{e}_n$ |

### A.1.6 Mean-Field Annealing

**Algorithm 21** (MFA — Mean-Field Annealing)

**Input:**

| | | |
|---|---|---|
| $X$ | – | normalized graph $X = (V, E, \boldsymbol{X})$ |
| $\mu$ | – | penalty parameter |
| $\lambda$ | – | step size |
| $T$ | – | temperature |
| $T_0$ | – | initial value of temperature |
| $T_f$ | – | minimum value of temperature |
| $\alpha$ | – | rate at which the temperature is decreased |
| $I_T$ | – | maximum number of iterations at each value of $T$ |

**Initialization:**

set $\boldsymbol{x} = \boldsymbol{x}_0 \in \mathbb{U}^n$

set $\boldsymbol{W} = \boldsymbol{O_X} - \mu \boldsymbol{A}$

set $\boldsymbol{h} = \boldsymbol{d_X}$

set temperature $T = T_0$

set iteration $t = 0$

**Procedure:**

**repeat** (*anneal*)

  set $t_T = 0$

  **repeat** (*update*)

    $\boldsymbol{u} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{h}$

    $\boldsymbol{x} = (1 - \lambda)\boldsymbol{x} + \lambda\, g_{\frac{1}{T}}(\boldsymbol{u})$

    $t_T = t_T + 1$

  **until** $t_T = I_T$

  $T = \alpha T$

**until** $T \leq T_f$

$\boldsymbol{x} = \Theta(\boldsymbol{x})$

**Output:** $\omega(\boldsymbol{x})$

**Parameter Settings:**

| $\mu$ | $\lambda$ | $T_0$ | $T_f$ | $\alpha$ | $I_T$ | $\boldsymbol{x}_0$ |
|---|---|---|---|---|---|---|
| $4n$ | 0.1 | $\frac{3}{4}(1-p)n^2$ | 1.0 | s.b. | $n$ | $\left(\frac{1}{2} + \boldsymbol{\nu}\right) \boldsymbol{e}_n$ |

To define $T_0$, we use the density

$$p = \frac{2|E|}{n(n-1)}$$

of $X$. There are two values for $\alpha$:

1. $\alpha = 0.9$ for the first three annealing loops.
2. $\alpha = 0.5$ for the remaining time.

By $\boldsymbol{\nu}$ we denote a noise vector drawn from a uniform distribution on $[-0.05, +0.05]^n$.

### A.1.7 Potts Mean-Field Annealing

Potts Mean-Field Annealing is almost of the same form as Mean-Field Annealing. We merely replace the update rule of Mean-Field Annealing by the Potts dynamics (4.10) described in Section 4.3.5. Moreover, other parameters have been used:

**Parameter Settings:**

| $\mu$ | $\lambda$ | $T_0$ | $T_f$ | $\alpha$ | $I_T$ | $\boldsymbol{x}_0$ |
|---|---|---|---|---|---|---|
| $n$ | 0.1 | $n$ | 1.0 | s.b. | 1 | $\left(\frac{1}{2} + \boldsymbol{\nu}\right) \boldsymbol{e}_n$ |

The setting for $\alpha$ and $\boldsymbol{x}_0$ are explained in the previous section.

## A.2 Other Meta-Heuristics

In this section we describe

1. Replicator Equations [280] and
2. Simulated Annealing [146]

The default parameter settings of the Replicator equations slightly differ from [280]. The parameters of simulated annealing are taken from [146].

### A.2.1 Replicator Dynamics

In [280] Pelillo suggested the *Exponential Replicator Equations* (REP) to approximately solve a regularized Motzkin-Strauss formulation of the MCP [271, 35]. The Replicator Equations are a continuous dynamical system derived from evolutionary game theory [144]. The regularized Motzkin-Strauss formulation of the `MCP` is the quadratic program

$$\begin{aligned} \text{maximize} \quad & f(\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T} \widehat{\boldsymbol{X}} \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{x}^\mathsf{T} \boldsymbol{e} = 1 \\ & \boldsymbol{x} \geq 0, \end{aligned} \tag{A.1}$$

where $\widehat{\boldsymbol{X}} = \boldsymbol{X} + \frac{1}{2}\boldsymbol{I}_n$. It is shown in [35] that the local minima of (A.1) are in one-to-one correspondence with the maximal cliques of $X$. In addition, the minima

are strict and the optimum value of (A.1) is

$$\frac{1}{2(1-f(\mathbf{1}_C))},$$

where $\mathbf{1}_C$ is the characteristic vector of a maximum clique $C$ of $X$. For an extension of the Motzkin-Strauss formulation to the MVCP we refer to [37]. It is unclear how to generalize (A.1) to the MWCP in its most general form. Algorithm 22 describes the REP algorithm for solving program (A.1).

**Algorithm 22** (REP — Exponential Replicator Equations)

**Input:**
- $X$ – normalized graph $X = (V, E, \boldsymbol{X})$
- $\varepsilon$ – precision
- $\kappa$ – exponentiation constant

**Initialization:**

set $\boldsymbol{x}$ such that $\boldsymbol{x}^{\mathsf{T}}\boldsymbol{e} \approx 1$

set $\boldsymbol{W} = \boldsymbol{X} + \frac{1}{2}\boldsymbol{I}_n$

**Procedure:**

**repeat**

$\boldsymbol{\pi} = \exp(\kappa \boldsymbol{W}\boldsymbol{x})$

**for all** $i \in [1:n]$

$x_i = x_i \pi_i / \boldsymbol{x}^T \boldsymbol{\pi}$

perturb $\boldsymbol{x}$ if it is a saddle point

**until** $\|\boldsymbol{x} - \mathbf{1}_C\| \leq \varepsilon$ for some maximal clique $C$

**Output:** $\omega(\boldsymbol{x})$

**Parameter Settings:**

| $\varepsilon$ | $\kappa$ | $\boldsymbol{x}_0$ |
|---|---|---|
| $10^{-7}$ | 10 | $(1/n + \boldsymbol{\nu})\boldsymbol{e}_n$ |

By $\boldsymbol{\nu}$ we denote a noise vector drawn from a uniform distribution over $[-\varepsilon, +\varepsilon]^n$.

### A.2.2 Simulated Annealing

To solve the MCP, Homer and Peinado [146] combined simulated annealing and the exterior point penalty method using a quenched annealing schedule for both parameters, the temperature $T$, and the penalty $\mu$. We adopt this strategy for the MWCP.

**Algorithm 23** (SA — Simulated Annealing)

**Input:**

| | | |
|---|---|---|
| $X$ | – | normalized graph $X = (V, E, \boldsymbol{X})$ |
| $\mu_0$ | – | initial value of penalty parameter |
| $\mu_f$ | – | final value of penalty parameter |
| $\alpha_\mu$ | – | amount at which $\mu$ is increased |
| $T_0$ | – | initial value of temperature |
| $T_f$ | – | final value of temperature |
| $\alpha_T$ | – | amount at which $T$ is decreased |
| $I_*$ | – | maximal number of iterations |

**Initialization:**

set $\boldsymbol{x} = \boldsymbol{x}_0 \in \mathbb{B}^n$

set $\mu = \mu_0$

set $T = T_0$

set $\boldsymbol{h} = \boldsymbol{d_X}$

**Procedure:**

**for** $t = [1 : I_*]$

$\boldsymbol{W} = \boldsymbol{O_X} - \mu \boldsymbol{A}$

draw $i$ from a uniform distribution over $[1 : n]$

$\Delta E_i = (1 - 2x_i)(\boldsymbol{W}\boldsymbol{x} + h_i)$

draw $\rho$ from a uniform distribution over $[0, 1[$

**if** $\rho < \exp(-\Delta E_i / T)$ **then** $x_i = 1 - x_i$

$\mu = \mu + \mu_a$

$T = T - T_a$

**Output:** $\omega(\boldsymbol{x})$

**Parameter Settings:**

| $\mu_0$ | $\mu_f$ | $\alpha_\mu$ | $T_0$ | $T_f$ | $\alpha_T$ | $I_*$ | $\boldsymbol{x}_0$ |
|---|---|---|---|---|---|---|---|
| 0.7 | 1.2 | $0.5/I_*$ | 1.0 | 0.0 | s.b. | $n^2$ | $\boldsymbol{e}_n$ |

There are two values for $\alpha_T$:

1. $\alpha_T = \dfrac{2}{I_*}$ for $t \in [1 : I_*/4]$
2. $\alpha_T = \dfrac{2}{3I_*}$ for $t \in ]I_*/4 : I_*]$

The temperature $T$ is linearly reduced from 1 to 0.5 during the first 25% of the time and from 0.5 to 0 during the remaining 75%.

# App. B Experimental Settings

## B.1 Performance Evaluation

*Performance profiles* proposed by [75] are a useful tool for benchmarking and comparing optimization algorithms. What makes them so useful is that they combine the best features of other common tools for analyzing benchmarking results.

The performance profile of an algorithm for a performance metric is a cumulative distribution function. Assume that we are given a set $\mathcal{A}$ of algorithms and a set $\mathcal{P}$ of problem instances. Let us denote by the number of problems $n_p = |\mathcal{P}|$. For time measures, we use *performance ratios*, and for measures of solution quality, we use *performance deviations*.

### B.1.1 Performance Profile for Time Metrics

We first describe the performance profile of an algorithm for time metrics. For each problem $p \in \mathcal{P}$ and each algorithm $a \in \mathcal{A}$, we define

$$t_{p,a} = \text{time until termination of algorithm } a \text{ to solve problem } p.$$

Examples of time metrics are the clock time and the number of iterations. We compare the performance of algorithm $a$ on problem $p$ with the best performance by any algorithm from $\mathcal{A}$ on this problem. For this, we use the *performance ratio*

$$r_{p,a} = \frac{t_{p,a}}{\min\{t_{p,a} \,:\, a \in \mathcal{A}\}}.$$

The *performance profile* of algorithm $a$ for a time metric is then defined by

$$\rho_a(\tau) = \frac{\big|\{p \in \mathcal{P} \,:\, r_{p,a} \leq \tau\}\big|}{n_p}.$$

The quantity $\rho_a(\tau)$ estimates the probability that a performance ratio $r_{p,a}$ is within a factor $\tau$ of the best possible ratio. The function $\rho_a$ is the estimated cumulative distribution function for the performance ratio.

### B.1.2 Performance Profile for Metrics of Solution Quality

Next, we describe the performance profile of an algorithm for metrics of solution quality. Let

$$q_{p,a} = \text{solution quality of algorithm } a \text{ on problem } p.$$

An example of a metric of solution quality is a structural similarity of two given graphs or the weight of a clique. The *performance deviation* is of the form

$$d_{p,a} = 1 - \frac{q_{p,a}}{\max\{q_{p,a} : a \in \mathcal{A}\}}.$$

As for time metrics, the *performance profile* of algorithm $a$ for a metric of solution quality is defined by

$$\rho_a(\tau) = \frac{\big|\{p \in \mathcal{P} : d_{p,a} \leq \tau\}\big|}{n_p}.$$

The quantity $\rho_a(\tau)$ estimates the probability that performance deviation $d_{p,a}$ deviates $100 \cdot \tau$ percent from the best solution. The function $\rho_a$ is the estimated cumulative distribution function for the performance deviation.

### B.1.3 Properties of Performance Profiles

Performance profiles have the following properties:

1. Given a performance metric, the performance profile $\rho_a : \mathbb{R} \to [0, 1]$ for an algorithm $a$ is a non-decreasing step function.

2. For time metrics, the value $\rho_a(1)$ is the estimated probability that algorithm $a \in \mathcal{A}$ terminates first for a class of problems represented by a finite sample set $\mathcal{P}$. If we consider metrics of solution quality, the value $\rho_a(0)$ is the estimated probability that algorithm $a \in \mathcal{A}$ finds the best solution for the same class of problems.

3. An algorithm $a$ outperforms algorithm $a'$ with respect to a performance metric if $\rho_a \geq \rho_{a'}$ and $\rho_a \neq \rho_{a'}$. Thus, rapid increase of a performance profile to the value 1 indicates better performance rather than slow increase.

## B.2 Datasets

### B.2.1 Segmented Images

The segmented image dataset has been kindly provided by Stefan Bischoff of the Heinrich Hertz Institute, Berlin. The dataset consists of complete attributed graphs describing visual properties of 39 segmented images. Vertices of a graph represent

segments. Each vertex is labeled with a 166 dimensional feature vector (MPEG-7 descriptors) describing visual properties of a segment. Each pair of vertices is connected by an edge labeled by the Euclidean distance between the centers of the corresponding segments. The segmentation procedure and feature extraction process of the MPEG-7 descriptors is described in Bischoff's dissertation [30]. The images are shown in Figure 4.9 of Section 4.6.

Table B.1 summarizes some characteristics of the graphical representations with respect to the number of vertices.

| data | min | max | mean | dev |
|---|---|---|---|---|
| images | 20 | 50 | 31.6 | 7.2 |

**Table B.1** Shown are the minimum, maximum, mean, and standard deviation with respect to the number of segments.

### B.2.2 Mutagenesis Dataset

The MUTAGENESIS dataset is a benchmark dataset for relational learning originating from the ILP community. The dataset consists of 230 chemical compounds [342]. The mutagenicity of a chemical compound is closely related to its cancerogenicity. A particular problem lies in discovering rules to predict mutagenicity in a database of nitro-aromatic compounds (see [342]).

The MUTAGENESIS dataset is usually divided into two subsets containing 188 and 42 examples, respectively. Each compound in the dataset is described by its atoms and their bonds. An atom is characterized by its element symbol, its type (e.g. aromatic) and a partial electrical charge. Bonds are either simple, double, or aromatic. Attributes that describe global properties of a molecule are not considered.

The following table shows some characteristics of the MUTAGENESIS dataset with respect to the number of atoms in a compound.

| data | min | max | mean | dev |
|---|---|---|---|---|
| 188 | 14 | 40 | 26.03 | 6.3 |
| 42 | 13 | 39 | 23.83 | 8.2 |
| 230 | 13 | 40 | 25.63 | 6.73 |

**Table B.2** Shown are the minimum, maximum, mean, and standard deviation with respect to the number of atoms of the 188 dataset, the 42 dataset, and all 230 compounds.

## B.3 Synthetic Test Graphs

### B.3.1 Simple Graphs

1. *Random Graphs:* Let $V$ be a vertex set of a graph $X$ consisting of $n$ elements and $p \in [0,1]$ be a number called *edge probability*. An $n$-vertex $p$-random graph is a simple graph of order $n$ where each pair of vertices is connected by an edge with probability $p$.
We generate $n$-vertex $p$-random graphs as follows: For each pair $\{i,j\} \in V^{[2]}$ we decide by a random experiment whether or not $\{i,j\}$ shall be an edge in $X$. A random experiment for $\{i,j\}$ consists of drawing a number $\rho$ from a uniform distribution over $\mathbb{U}$. We accept $\{i,j\}$ as an edge for $X$ if, and only if, $\rho \leq p$.
Generating random graphs with varying edge probability $p$ is a simple method for obtaining graphs that *almost surely* exhibit different graph properties like (non)-connectivity, (non)-planarity, (non)-existence of Hamiltonians, etc. Note that the clique size of a random graph is relatively small compared to its density.

2. *$k$-Random Cliques Graphs:* A $k$-random cliques graph of order $n$ is the union of $k$ randomly generated cliques of various size. This type of graphs has a wider range of different clique sizes than random graphs. Therefore, the `MCP` for $k$-random cliques graphs is considered to be harder than for random graphs [167].
We generate a $k$-random cliques graph as follows:

**Algorithm 24** ($k$-Random Cliques Graph Generator)

**Input:**
$n$ – number of vertices
$k$ – number of cliques

**Initialization:**
set $V = [1:n]$
set $E = \emptyset$

**Procedure:**
**for** $i = [1:k]$
  draw $\rho$ from a uniform distribution over $[0, 0.5]$
  set $C = \emptyset$
  **for** $j = [1:n]$
    draw $\rho_j$ from a uniform distribution over $\mathbb{U}$
    **if** $\rho < \rho_j$ **then** $C = C \cup \{j\}$
  $E = E \cup C^{[2]}$

**Output:** $X = (V, E, \boldsymbol{X})$

### B.3.2 Weighted Graphs

Let $X = (V, E, \boldsymbol{X})$ be a simple graph. We consider the following sampling methods to randomly assign weights to the vertices and edges of $X$.

1. *IID Sampling:* Each vertex and each edge of $X$ is assigned an independent identically distributed value drawn from $]0, 1]$.

2. *IGD Sampling:* To assign a weight $x_{\boldsymbol{i}}$ for a vertex- or edge-item $\boldsymbol{i}$, we proceed as follows: First, we draw a mean $\mu$ and standard deviation $\sigma$ from a uniform distribution over $[0, 1]$. Next, we draw a value $w$ from a Gaussian distribution with mean $\mu$ and standard deviation $\sigma$. Finally, we assign the weight $x_{\boldsymbol{i}} = [w]_\beta$, where $[.]_\beta$ is the limiter function with gain $\beta = 1$. Note that we delete an edge whenever $[w]_\beta = 0$.

3. *Irregular Uniform Sampling:* In an irregular uniform weighting scheme, the vertex and edge weights are drawn from a non-independent uniform distribution over $]0, 1]$. Algorithm 25 describes the irregular uniform sampling scheme.

---

**Algorithm 25** (Irregular Uniform Sampling)

---

**Input:**

$X$ – simple graph $X = (V, E, \boldsymbol{X})$

**Initialization:**

set $\boldsymbol{X'} = \boldsymbol{0}_{n,n}$

**Procedure:**

**for** $i = [1 : n]$
  draw $\rho$ from a uniform distribution over $]0, 1]$
  draw $\rho_{ii}$ from a uniform distribution over $]0, 1]$
  set $x'_{ii} = \rho\rho_{ii}$
  **for** $j = ]i : n]$
    draw $\rho_{ij}$ from a uniform distribution over $]0, 1]$
    set $x'_{ij} = \rho\rho_{ij}$
    set $x'_{ji} = x'_{ij}$
  set $\boldsymbol{X} = \boldsymbol{X'}$

**Output:** $X = (V, E, \boldsymbol{X})$

---

# References


1. E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, 1989.
2. M. Abdulrahim and M. Misra. A graph isomorphism algorithm for object recognition. *Pattern Analysis and Application*, 1(3):189–201, 1998.
3. D.H. Ackley, G. Hinton, and T. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.
4. K. Aihara, T. Tabake, and M. Toyoda. Chaotic neural networks. *Physics Letters A*, 144(6/7):333–340, 1990.
5. S.V.B. Aiyer. Solving combinatorial optimization problems using neural networks. Technical Report CUED/F-IN-FENG/TR89, Engineering Department, Cambridge University, 1991.
6. S.V.B. Aiyer, M. Niranjan, and F. Fallside. A theoretical investigation into the performance of the hopfield model. *IEEE Transactions on Neural Networks*, 1(2):204–215, 1990.
7. Y. Akiyama, A. Yamashita, M. Kajiura, and H. Aiso. Combinatorial optimization with Gaussian machines. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 533–540, 1989.
8. H.A. Almohamad. A polynomial transform for matching pairs of weighted graphs. *Journal of Applied Mathematical Modeling*, 15(4), 1991.
9. H.A. Almohamad and S.0. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):522–525, 1993.
10. A.P. Ambler, H.G. Barrow, C.M. Brown, R.M. Burstall, and R. J. Popplestone. A versatile computer-controlled assembly system. In *International Joint Conference on Artificial Intelligence*, pages 298–307. Stanford University, California, 1973.
11. D.J. Amit. *Modeling Brain Function: The world of attractor neural networks.* Cambridge University Press, 1989.
12. A.F.R. Araújo and A. Barreto. Context in temporal sequence processing: A self-organizing approach and its application to robotics. *IEEE Transactions on Neural Networks*, 13(1):45–57, 2002.
13. R. Azam. *Studies on Quasi-Newton Methods for Nonsmooth Convex Opti-*

*mization*. PhD thesis, Dept. of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, 1998.

14. A. Baddeley. *Working memory*. Clarendon Press, 1986.
15. A.T. Balaban, editor. *Chemical Applications of Graph Theory*. Academic Press, London, 1976.
16. E. Balas and C.S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal of Computing*, 15(4):1054–1068, 1986.
17. D.H. Ballard, P.C. Gardner, and M.A. Srinivas. Graph problems and connectionist architectures. Technical Report TR 167, Dept. Computer Science, University of Rochester, 1987.
18. A. Barreto, A.F.R. Araújo, and S.C. Kremer. A taxonomy for spatiotemporal connectionist networks revisited: the unsupervised case. *Neural Computation*, 15:1255–1320, 2003.
19. H. Barrow and R. Burstall. Subgraph isomorphism, matching relational structures and maximal cliques. *Information Processing Letters*, 4:83–84, 1976.
20. H. Barrow and R.J. Popplestone. Relational descriptions in picture processing. *Machine Intelligence*, 6:377–396, 1971.
21. R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610–637, 2001.
22. S. Behnke, M. Pfister, and R. Rojas. A study on the combination of classifiers for handwritten digit recognition. In *Proceedings of Neural Networks in Applications*, pages 39–46, 1998.
23. R.E. Bellmann. *Dynamic Programming*. Princeton University Press, 1957.
24. E. Bengoetxea. *Inexact Graph Matching using Estimation of Distribution Algorithms*. PhD thesis, Ecole Nationale Supérieure des Télécommunications (Paris), Département Traitement du Signal et des Images, 2002.
25. L.T. Benjamin, J.R. Hopkins, and J.R. Nation. *Psychology*. Macmillan, New York, 3rd edition, 1994.
26. D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2 edition, 1999.
27. D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
28. D.P. Bertsekas and J.N. Tsitsiklis. Gradient convergence in gradient methods. *SIAM Journal on Optimization*, 10:627–642, 2000.
29. R. Bhattarcharya. *Transformation of Linear Control Algorithms into Operationally Optimal Real-Time Tasks*. PhD thesis, University of Minnesota, 2003.
30. S. Bischoff. *Konstruktion einer Suchmaschine für segmentierte Bilder, die speziell durch interpretierte Graphen beschrieben werden*. PhD thesis, Dept. of Electrical Engineering and Computer Science, Technical University of Berlin, 2005.
31. S. Bischoff, D. Reuss, and F. Wysotzki. Applied connectionist methods in

computer vision to compare segmented images. In A. Günter, R. Kruse, and B. Neumann, editors, *KI 2003: Advances in Artificial Intelligence. 26th Annual Conference on AI*, volume 2821 of *LNAI*, pages 312–326. Springer-Verlag, 2003.

32. C.M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.
33. M. Boddy and T.L. Dean. Solving time-dependent planning problems. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 979–984, 1989.
34. M. Boddy and T.L. Dean. Deliberation scheduling for problem solving in time constrained environments. *Artificial Intelligence*, 67(2):245–285, 1994.
35. I.M. Bomze. Evolution towards the maximum clique. *Journal of Global Optimization*, 10:143–164, 1997.
36. I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4, pages 1–74. Kluwer Academic Publishers, Boston, MA, 1999.
37. I.M. Bomze, M. Pelillo, and V. Stix. Approximating the maximum weight clique using replicator dynamics. *IEEE Transactions on Neural Networks*, 11:1228–1241, 2000.
38. D. Bonchev. *Chemical Graph Theory: Introduction and Fundamentals.* Taylor and Francis, 1991.
39. P. Böncke. Funktionslernen über strukturierten Objekten mit mehrschichtigen neuronalen netzen. Master's thesis, Dept. of Electrical Engineering and Computer Science, Technical University of Berlin, 2005.
40. N.K. Bose and P. Liang. *Neural network fundamentals with graphs, algorithms, and applications.* McGraw-Hill, Inc., 1996.
41. L.M. Brégman. Certain properties of nonnegative matrices and their permanents. *Dokl. Akad. Nauk SSSR*, 211:27–30, 1973.
42. D.P. Brown. Matrix tests for period 1 and 2 limit cycles in discrete threshold networks. *IEEE Transactions onSystems, Man and Cybernetics*, 22(3):552–554, 1992.
43. J. Bruck. On the convergence properties of the Hopfield model, 1990.
44. J. Buhmann and H. Kühnel. Complexity optimized data clustering by competitive neural networks. *Neural Computation*, 5(3):75–88, 1993.
45. J.M. Buhmann. *The Handbook of Brain Theory and Neural Networks*, chapter Data clustering and learning, pages 278–281. MIT Press, 1995.
46. H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.
47. H. Bunke. Recent developments in graph matching. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 2, pages 117–124,

Barcelona, Spain, 2000.

48. H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.
49. H. Bunke, X. Jiang, and A. Kandel. On the minimum common supergraph of two graphs. *Computing*, 1:13–25, 2000.
50. H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19, 1998.
51. E.R. Caianiello. Outline of a theory of thought and thinking machines. *Journal of Theoretical Biology*, 1:204–235, 1961.
52. M. Carcassoni and E.R. Hancock. Weighted graph-matching using modal clusters. In *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 260–269, 2001.
53. G. A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organising neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1987.
54. C.-W.K. Chen and D.Y.Y. Yun. Toward solving maximal overlap set problems. Technical Report TR-LIPSC&Y96a, Laboratory of Intelligent and Parallel Systems, University of Hawaii, 1996.
55. C.-W.K. Chen and D.Y.Y. Yun. Unifying graph-matching problem with a practical solution. In *Proceedings of International Conference on Systems, Signals, Control, Computers*, 1998.
56. L. Chen and K. Aihara. Chaotic simulated annealing by a neural network model with transient chaos. *Neural Networks*, 8(6):915–930, 1995.
57. W.J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):749–764, 1995.
58. F.H. Clarke. *Optimization and Nonsmooth Analysis*. John Wiley & Sons, NY, 1983.
59. M. Cone, R. Venkataraghavan, and F. McLafferty. Molecular structure comparison program for the identification of maximal common substructures. *Journal of the American Chemical Society*, 99:7668–7671, 1977.
60. D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.
61. D.J. Cook and L.B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.
62. P. Cooper. Structure recognition by connectionist relaxation: Formal analysis. In R. Goebel, editor, *Proceedings of the Seventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 148–155,

1988.

63. L.P. Cordella, P. Foggia, C. Sasone, and M. Vento. Evaluating performance of the VF graph matching algorithm. In *Proceedings of the 10th International Conference on Image Analysis and Processing*, pages 1172–1177, 1999.
64. D.G. Corneil and R.A. Mathon. Algorithmic techniques for the generation and analysis of strongly regular graphs and other combinatorial configurations. *Annals of Discrete Mathematics*, 1:1–32, 1978.
65. P. Crescenzi and V. Kann (*eds.*). A compendium of NP optimization problems. URL: http://www.nada.kth.se/∼viggo/wwwcompendium/, 1994-2004.
66. V. Crespi. *Structural and Computational Properties of Certain Permanents*. PhD thesis, University of Milan, 1997.
67. A.D.J. Cross, R.C. Wilson, and E.R. Hancock. Inexact graph matching with genetic search. *Pattern Recognition*, 30(6):953–970, 1997.
68. B. D'Ambrosio. Resource bounded-agents in an uncertain world. In *Proceedings of the Workshop on Real-Time Artificial Intelligence Problems (IJCAI'89)*, 1989.
69. N. Das, B.B. Bhattacharya, and J. Dattagupta. Isomorphism of conflict graphs in multistage interconnection networks and its application to optimal routing. *IEEE Transaction on Computers*, 42(6):665–677, 1993.
70. J.W. de Fockert, G. Rees, C.D. Frith, and N. Lavie. The role of working memory in visual selective attention. *Science*, 291:1803–1806, 20001.
71. T.L. Dean. Intractability and time-dependent planning. In M.P. Georgeff and A.L. Lansky, editors, *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, 1987.
72. T.L. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 49–54, 1988.
73. P. Deuflhard and A. Hohmann. *Numerische Mathematik I.* de Gruyter, 2002.
74. P. Deuflhard and A. Hohmann. *Numerische Mathematik II.* de Gruyter, 2002.
75. E.D. Dolan and J.J. More. Benchmarking optimization software with performance profiles. *Mathematical Programming*, pages 201–213, 2002.
76. G.M. Downs and P. Willett. Similarity searching in databases of chemical structures. In K.B. Lipkowitz and D.B. Boyd, editors, *Reviews in Computational Chemistry*, volume 7, pages 1–66. VCH Publishers, Inc., 1996.
77. R.O. Duda, P.E. Hart, and D.G. Storck. *Pattern Classification.* J. Wiley & Sons, 2nd edition, 2001.
78. R. Durbin and D. Willshaw. An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326:689–691, 1987.
79. S. Edelman. *Representation and Recognition in Vision.* MIT Press, 1999.
80. D. Ellis, J. Furner-Hines, and P. Willett. Measuring the degree of similarity

between objects in text retrieval systems. *Perspectives in Information Management*, 3:128–149, 1993.

81. E. Erkut, T. Baptie, and B. von Hohenbalken. The discrete p-maxian location problem. *Computers and Operations Research and their Application to Problems of World Concern*, 17:51–61, 1990.
82. M.A. Eshera and K.S Fu. A graph distance measure for image analysis. *IEEE Transactions on Systems Man and Cybernetics*, 14(3):353–363, 1984.
83. M.A. Eshera and K.S Fu. An image understanding system using attributed symbolic representation and inexact graph-matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(5):604–618, 1986.
84. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the SIGCOMM '99 Symposium on Communications Architectures and Protocols*, pages 251–262, 1999.
85. Y. Fang, M.A. Cohen, and T.G. Kincaid. Dynamics of a winner-take-all neural network. *Neural Networks*, 9(7):1141–1154, 1996.
86. O. Faugeras and M. Berthod. Improving consistency and reducing ambiguity in stochastic labeling: An optimization approach. *EEE Transactions on Pattern Analysis and Machine Intelligence*, 3(4):412–424, 1981.
87. O. Faugeras and K. Price. Semantic labelling of aerial images using stochastic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(6):633–642, 1981.
88. J.A. Feldmann and D.H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6:205–254, 1982.
89. M.-L. Fernandez and G. Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, pages 753–758, 2001.
90. A.M. Finch, R.C. Wilson, and E.R. Hancock. Matching delaunay triangulations by probabilistic relaxation. In *Proceddings of the 8th International Conference on Image Analysis and Processing*, pages 350–358, 1995.
91. A.M. Finch, R.C. Wilson, and E.R. Hancock. An energy function and continuous edit process for graph matching. *Neural Computation*, 10(7):1873–1894, 1998.
92. I. Fischer. *Enumeration of perfect matchings: Rhombus tilings and Pfaffian graphs*. PhD thesis, University Vienna, 2000.
93. M.A. Fischler and R.A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, 22(1):67–92, 1973.
94. R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, 1987.
95. P. Foggia, C. Sasone, and M. Vento. A performqance comparison of five algorithms for graph isomorphism. In *3rd IAPR -TC-15 Workshop on Graph-based Representations in Pattern Recognition*. CUEN ed, 2001.

96. S. Fortin. The Graph Isomorphism Problem. Technical Report TR 96-20, Dept. of Computing Science, University of Alberta, Canada, 1996.

97. P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9(5):768–786, 1998.

98. E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

99. N. Funabiki and S. Nishikawa. A neural network model for finding a near-maximal clique. *Journal of Parallel and Distributed Computing*, 14(3):340–344, 1992.

100. N. Funabiki, Y. Takefuji, and K.C. Lee. Comparisons of energy-descent optimization algorithms for maximum clique. *IEICE Trans. Fundamentals*, E79-A(4):452–460, 1996.

101. A.A. Gaivoronski. Convergence properties of backpropagation for neural networks via theory of stochastic gradient methods. *Optimization Methods and Software*, 4:117–134, 1994.

102. G. Gálan-Marín, E. Mérida-Casermeiro, and J. Muñoz-Pérez. Modelling competitive Hopfield networks for the maximum clique problem. *Computers & Operations Research*, 30:603–624, 2003.

103. J.M. Gallone and F. Charpillet. Hopfield neural network for scheduling. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI'1996)*, pages 223–227, 1996.

104. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

105. T. Gärtner. Exponential and geometric kernels for graphs. In *NIPS Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*, 2002.

106. T. Gärtner. A survey of kernels for structured data. *SIGKKD Explorations*, 5(1):49–58, 2003.

107. A. Garvey and V. Lesser. A survey of research in deliberative real-time artificial intelligence. *The Journal of Real-Time Systems*, 6:317–347, 1994.

108. P. Geibel, B.J. Jain, and F. Wysotzki. SVM learning with the SH inner product. In M. Verleysen, editor, *Proceedings of the 12th European Symposium on Artificial Neural Networks (ESANN) 2004*, pages 299–304. D-Facto, Brussels, 2004.

109. S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

110. D. Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983.

111. A. Gersho and R.M. Gray. *Vector Quantization and Signal Compression*.

Kluwer, 1992.

112. L.E. Gibbons, D.W. Hearn, P.M. Pardalos, and M.V. Ramana. Continuous characterizations of the maximum clique problem. *Mathematics of Operations Research*, 22:754–768, 1997.

113. R. Glantz, M. Pellilo, and W.G. Kropatsch. Matching segmentation hierarchies. *International Journal for Pattern Recognition and Artificial Intelligence*, 18(3):397–424, 2004.

114. F. Glover. Tabu search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

115. F. Glover. Tabu search – Part II. *ORSA Journal on Computing*, 2(1):4–32, 1989.

116. S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

117. S. Gold and A. Rangarajan. Softmax to softassign: Neural network algorithms for combinatorial optimization. *Journal of Artificial Neural Networks*, 2(4):381–399, 1996.

118. S. Gold, A. Rangarajan, and E. Mjolsness. Learning with preknowledge: Clustering with point and graph matching distance measures. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 713–720. MIT Press, 1995.

119. R.M. Golden. *Mathematical Methods for Neural Network Analysis and Design*. MIT Press, 1996.

120. L. Goldfarb. A new approach to pattern recognition. In L.N. Kanal and A. Rosenfeld, editors, *Progress in Machine Intelligence and Pattern Recognition*, pages 1–156. Elsevier Science Publishers, 1985.

121. L. Goldfarb, J. Abela, V. C. Bhavsar, and V. N. Kamat. Can a vector space based learning model discover inductive class generalization in a symbolic environment? *Pattern Recognition Letters*, 16:719–726, 1995.

122. R.L. Goldstone. Similarity, interactive activation, and mapping. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 20(3):3–28, 1994.

123. R.L. Goldstone. *MIT Encyclopedia of the Cognitive Sciences*, chapter Similarity. MIT Press, 1999.

124. E. Goles. Antisymmetrical neural networks. *Discrete Applied Mathematics*, 13(1), 1986.

125. C. Goller. *A connectionist approach for learning search control heuristics for automated deduction systems*. PhD thesis, Technical University of Munich, Germany, 1997.

126. T. Graepel, R. Herbrich, P. Bollmann-Sdorra, and K. Obermayer. Classification on pairwise proximity data. In *Advances in Neural Information Processing*

*Systems, NIPS 11*, pages 438–444, 1999.

127. W.E.L. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints*. The MIT Press, 1990.
128. L. Grippo. A class of unconstrained minimization methods for neural network training. *Optimization Methods and Software*, 4:135–150, 1994.
129. S. Grossberg. Adaptive resonance theory. Technical Report CAS/CNS-2000-024, Boston University, Center for Adaptive Systems and Department of Cognitive and Neural Systems, 2000.
130. S. Günter and H. Bunke. Adaptive self-organizing map in the graph domain. In H. Bunke and A. Kandel, editors, *Hybrid Methods in Pattern Recognition*, pages 61–74. World Scientific, 2002.
131. S. Günter and H. Bunke. Self-organizing map for clustering in the graph domain. *Pattern Recognition Letters*, 23:401–417, 2002.
132. R. Hahnloser. On the piecewise analysis of networks of linear threshold neurons. *Neural Networks*, 11:691–697, 1998.
133. B. Hammer. *Learning with recurrent neural networks*. Springer, 2000.
134. B. Hammer and B.J. Jain. Neural methods for non-standard data. In *Proceedings of the 12th European Symposium on Artificial Neural Networks (ESANN)*, pages 281–292. D-Facto, Brussels, 2004.
135. B. Hammer and J.J. Steil. Perspectives on learning with recurrent networks. In M. Verleysen, editor, *Proceedings of the 10th European Symposium on Artificial Neural Networks (ESANN) 2002*. D-Facto, Brussels, 2002.
136. D.A. Harville. *Matrix Algebra from a Statistican's Perspective*. Springer, 1997.
137. D. Haussler. Convolution kernels on discrete structures. Technical Report Tech. Rep. UCSC-CRL-99-10, Department of Computer Science, University of California at Santa Cruz, 1999.
138. S. Haykin. *Neural Networks*. Prentice Hall, Inc., 2nd edition, 1999.
139. L. Herault, R. Horaud, F. Veillon, and J.J. Niez. Symbolic image matching by simulated annealing. In *Proceedings of the British Machine Vision Conference*, pages 319–324, 1990.
140. R. Herbrich. *Learning Kernel Classifiers*. The MIT Press, Cambridge, MA, 2002.
141. A.V.M. Herz. *Models of Neural Networks III*, chapter Global Analysis of Recurrent Neural Networks. Springer, 1996.
142. D. Hidović and M. Pelillo. Metrics for attributed graphs based on the maximal similarity common subgraph. *International Journal on Pattern Recognition and Artificial Intelligence*, 18(3):299–313, 2004.
143. M.W. Hirsch. Convergent activation dynamics in continuous time networks. *Neural Networks*, 2:331–349, 1989.

144. J. Hofbauer and K. Sigmund. *Evolutionary Games and Population Dynamics*. Cambridge University Press, UK, 1998.

145. T. Hofmann and J.M. Buhmann. Pairwise data clustering by deterministic annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(1):1–14, 1997.

146. S. Homer and M. Peinado. On the performance of polynomial-time clique approximation algorithms. In D.S. Johnson and M. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.

147. J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings National Academy of Sciences*, 79:2554–2558, 1982.

148. J.J. Hopfield. Neurons with graded respose have collective computation properties like those of two-state neurons. *Proceedings National Academy of Sciences*, 81:3088–3092, 1984.

149. J.J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

150. J.J. Hopfield and D.W. Tank. Computing with neural circuits: A model. *Science*, 223:625–633, 1986.

151. R. Horaud and T. Skordas. Stereo correspondence through feature grouping and maximal cliques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1168–1180, 1989.

152. E.J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 3rd Workshop on Uncertainty in Artificial Intelligence (UAI'1987)*, pages 429–444, 1987.

153. E.J. Horvitz. *Computation and Action Under Bounded Resources*. PhD thesis, Stanford University, California, 1990.

154. E.J. Horvitz and J.S. Breese. Ideal partition of resources for metareasoning. Technical Report KSL-90-26, Stanford Knowledge Systems Laboratory, Stanford, California, 1990.

155. E.J. Horvitz, H.J. Suermondt, and G.F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of the 5th Conference on Uncertainty in Artificial Intelligence (UAI'89)*, pages 182–193, 1989.

156. M.W. Howard, D.S. Rizzuto, J.C. Caplan, J.R. Madsen, J. Lisman, R. Aschenbrenner-Scheibe, A. Schultze-Bonhage, and M.J. Kahana. Gamma oscillations increase with working memory load in humans. *Cerebral Cortex*, 13:1369–1374, 2003.

157. B. Huet and E.R. Hancock. Shape recognition from large image libraries by inexact graph matching. *Pattern Recognition Letters*, 20(11-13):1259–1269,

1999.

158. J.E. Hummel and K.J. Holyoak. Distributed representation of structure: A theory of analogical access and mapping. *Psychological Review*, 104(3):427–466, 1997.

159. R. Hummel and S. Zucker. On the foundations of relaxation labeling processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):267–287, 1983.

160. M. Hunting. *Relaxation Techniques for Discrete Optimization Problems*. PhD thesis, University of Twente, 1998.

161. M. Hunting, U. Faigle, and W. Kern. A lagrangian relaxation approach to the edgeweighted clique problem. *European Journal of Operational Research*, 131(1):119–131, 2001.

162. D. Hwang and F. Fatouhi. Modifications of discrete hopfield neural optimization in maximum clique problem. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1189–1194, 2002.

163. S. Ishii and M. Sato. Doubly constrained network model for combinatorial optimization problems. In *Proceedings of the 1995 International Symposium on Nonlinear Theory and Its Applications*, pages 371–374, 1995.

164. S. Ishii and M. Sato. Constrained neural approaches to quadratic assignment problems. *Neural Networks*, 10:941–963, 1998.

165. Y. Ishitani. Model matching based on association graph for form image understanding. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 287–292, 1995.

166. R.G. Palmer J. Hertz, A. Krogh. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA, 1991.

167. A. Jagota. Approximating maximum clique with a Hopfield network. *IEEE Transactions of Neural Networks*, 6:724–735, 1995.

168. A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

169. B.J. Jain, P. Geibel, and F. Wysotzki. Combining recurrent neural networks and support vector machines for classifying structured data. In *27th Annual German Conference on Artificial Intelligence (KI)*, pages 241–255. Springer, 2004.

170. B.J. Jain, P. Geibel, and F. Wysotzki. SVM learning with the SH inner product. *Neurocomputing*, 2005. In press.

171. B.J. Jain and F. Wysotzki. Distance-based classification of structures within a connectionist framework. In R. Klinkenberg et al., editor, *Proceedings Fachgruppentreffen Maschinelles Lernen*, 2001.

172. B.J. Jain and F. Wysotzki. Efficient pattern discrimination with inhibitory WTA nets. In *Proceedings of the 11th International Conference on Artificial*

*Neural Networks (ICANN) 2001*, LNCS 2130, pages 827–834. Springer, 2001.

173. B.J. Jain and F. Wysotzki. On the short-term-memory of WTA nets. In M. Verleysen, editor, *Proceedings of the 9th European Symposium on Artificial Neural Networks (ESANN) 2001*, pages 289–294. D-Facto, Brussels, 2001.
174. B.J. Jain and F. Wysotzki. Fast winner-takes-all networks for the maximum clique problem. In M. Jarke, J. Koehler, and G. Lakemeyer, editors, *KI 2002: Advances in Artificial Intelligence*, volume 2479 of *LNAI*, pages 163–173. Springer, 2002.
175. B.J. Jain and F. Wysotzki. Self-organizing recognition and classification of relational structures. In W. Gray and C. Schunn, editors, *The 24th Annual Meeting of the Cognitive Science Society (COGSCI) 2002*, pages 163–173. Springer, 2002.
176. B.J. Jain and F. Wysotzki. An associative memory for the automorphism group of structures. In M. Verleysen, editor, *Proceedings of the 11th European Symposium on Artificial Neural Networks (ESANN) 2003*, pages 107–112. D-Facto, Brussels, 2003.
177. B.J. Jain and F. Wysotzki. Automorphism partitioning with neural networks. *Neural Processing Letters*, 17:205–215, 2003.
178. B.J. Jain and F. Wysotzki. A competitive winner-takes-all architecture for classification and pattern recognition of structures. In E. Hancock and M. Vento, editors, *Graph Based Representations in Pattern Recognition*, LNCS 2726, pages 259–270. Springer, 2003.
179. B.J. Jain and F. Wysotzki. A k-winner-takes-all classifier for structured data. In A. Günter, R. Kruse, and B. Neumann, editors, *KI 2003: Advances in Artificial Intelligence.*, volume 2821 of *LNAI*, pages 342–354. Springer-Verlag, 2003.
180. B.J. Jain and F. Wysotzki. A neural graph isomorphism algorithm based on local invariants. In M. Verleysen, editor, *Proceedings of the 11th European Symposium on Artificial Neural Networks (ESANN) 2003*, pages 79–84. D-Facto, Brussels, 2003.
181. B.J. Jain and F. Wysotzki. A novel neural network approach to solve exact and inexact graph isomorphism problems. In *Proceedings of the 13th International Conference on Artificial Neural Networks (ICANN) 2003*, LNCS 2714, pages 299–306. Springer, 2003.
182. B.J. Jain and F. Wysotzki. Perceptron learning in the domain of graphs. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN) 2003*, 2003.
183. B.J. Jain and F. Wysotzki. Central clustering of attributed graphs. *Machine Learning Journal. Special Issue: Theoretical Advances in Data Clustering*, 56(1-3):169–207, 2004.
184. B.J. Jain and F. Wysotzki. Discrimination networks for maximum selection.

*Neural Networks*, 17(1):143–15, 2004.

185. B.J. Jain and F. Wysotzki. Learning with neural networks in the domain of graphs. In *Proceedings Fachgruppentreffen Maschinelles Lernen*, 2004.

186. B.J. Jain and F. Wysotzki. The maximum weighted clique problem and Hopfield networks. In M. Verleysen, editor, *Proceedings of the 12th European Symposium on Artificial Neural Networks (ESANN) 2004*, pages 331–336. D-Facto, Brussels, 2004.

187. B.J. Jain and F. Wysotzki. Multi-layer perceptron learning in the domain of attributed graphs. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN) 2004*, 2004.

188. B.J. Jain and F. Wysotzki. Structural perceptrons for attributed graphs. In *Joint IAPR International Workshops on Structural and Syntactical Pattern Recognition and Statistical Pattern Recognition. Accepted for publication*, pages 85–94, 2004.

189. B.J. Jain and F. Wysotzki. Solving inexact graph isomorphism problems using neural networks. *Neurocomputing*, 63:45–67, 2005.

190. H. Jeong, B. Tombor, R. Albert, Z.N. Oltvai, and A.-L. Barabasi. The large-scale organization of metabolic networks. *Nature*, 407:651–654, 2000.

191. D.S. Johnson and M. Trick, editors. *Cliques, Coloring, and Satisfiability: Second Dimacs Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

192. M. Johnson. Some constrains on embodied analogical understanding. In D.H. Helman, editor, *Analogical Reasoning. Perspectives of Artificial Intelligence, Cognitive Science, and Philosophy*. Kluwer Academic Publishers, 1988.

193. M.A. Johnson. Relating metrics, lines and variables defined on graphs to problems in medicinal chemistry. In Y. Alavi, G. Chartrand, L. Lesniak, D.R. Lick, and C.E. Wall, editors, *Graph Theory With Applications to Algorithms and Computer Science*, pages 457–470. Wiley, 1985.

194. M.A. Johnson, M. Naim, V. Nicholson, and C.C. Tsai. Unique mathematical features of the substructure metric approach to quantitative molecular similarity analysis. In R.B. King and D.H. Rouvray, editors, *Graph Theory and Topology in Chemistry*, pages 219–225. Elsevier Science, 1987.

195. J.M. Jolion. Graph matching: What are we really talking about? In *Graph-based Representations in Pattern Recognition, 3rd IAPR-TC-15 Workshop*, 2001.

196. J.M. Jolion and A. Rosenfeld. *A Pyramid Framework for Early Vision*. Kluwer, 1994.

197. F. Kaden. Graphmetriken und Distanzgraphen. In *Beiträge zur angewandten Graphentheorie*. ZKI-Informationen, Berlin, 1982.

198. E.R. Kandel, J.H. Schwartz, and T.M. Jessel. *The Principles of Neuroscience*.

McGraw-Hill, 2000.

199. V. Kann. On the approximability of NP-complete optimization problems. Master's thesis, Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1992.

200. R.M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. Plenum Press, New York, 1972.

201. H. Kashima and A. Inokuchi. Kernels for graph classification. In *ICDM Workshop on Active Mining*, 2002.

202. H. Kashima and T. Koyanagi. Kernels for semi-structured data. In *Proceedings of the 19th International Conference on Machine Learning*. Morgan Kaufmann, 2002.

203. P.W. Kasteleyn. Graph theory and crystal physics. In F. Harary, editor, *Graph Theory and Theoretical Physics*, pages 43–110. Academic Press, 1967.

204. M.T. Keane, T. Ledgeway, and S. Duff. Constraints on analogical mapping: A comparison of three models. *Cognitive Science*, 18:387–438, 1994.

205. J.L. Kelley. *General Topology*. D. van Nostrand Company, Inc., 1955.

206. V.M. Kibardin. Decomposition into functions in the minimization problem. *Automation and Remote Control*, 40:1311–1323, 1980.

207. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

208. H. Kitaoka, Y. Park, J. Tschirren, J.M. Reinhardt, M. Sonka, G. McLennan, and E.A. Hoffman. *Automated nomenclature labeling of the bronchial tree in 3D-CT lung images*, pages 1–11. LNCS 2489. Springer, 2002.

209. J. Kittler and E.R. Hancock. Combining evidence in probabilistic relaxation. *International Journal of Pattern Recognition and Artificial Intelligence*, 3:29–52, 1989.

210. J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

211. M. Klin, Ch. Rücker, G. Rücker, and G. Tinhofer. Algebraic Combinatorics in Mathematical Chemistry. Methods and Algorithms. I. Permutation Groups and Coherent (Cellular) Algebras. *Communications in mathematical and in computer chemistry*, pages 1–138, 1999.

212. J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem: its structural complexity*. Birkhauser, 1993.

213. T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

214. T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, 1984.

215. T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 1995.

216. S. Kosinov and T. Caelli. Inexact multisubgraph matching using graph eigenspace and clustering models. In T. Caelli, A. Amin, R. Duin, M. Kamel, and D. de Ritter, editors, *Proceedings of the Joint IAPR International Workshops Structural, Syntactic, and Statistical Pattern Recognition (SSPR and SPR) 2002*, LNCS 2396, pages 133–142. Springer, 2002.

217. K. Koutroumbas and N. Kalouptsidis. Qualitative analysis of the parallel and asynchronous modes of the hamming network. *IEEE Transactions on Neural Networks*, 15(3):380–391, 1994.

218. R. Kree and A. Zippelius. Recognition of topological features of graphs. *Journal of Physics A: Mathematical and General*, 21:813–818, 1988.

219. S.C. Kremer. Spatiotemporal connectionist networks: a taxonomy and review. *Neural Computation*, 13:249–306, 2001.

220. C.L. Krumhansl. Concerning the applicability of geometric models to similarity data: The interrelationship between similarity and spatial density. *Psychological Review*, 85(5):445–463, 1978.

221. M.J. Kuby. Programming models for facility dispersion: the p-dispersion and maxisum dispersion models. *Geographical Analysis*, 19:315–329, 1987.

222. P. Langley. Systematic and nonsystematic search strategies. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, pages 145–152, 1992.

223. A. Lapedes and R. Farber. How neural nets work. In D. Z. Anderson, editor, *Neural Information Processing Systems*, pages 442–456. American Institute of Physics, 1987.

224. J. LaSalle and S. Lefschetz. *Stability by Ljapunov's Direct Method*. Academic Press, 1961.

225. C. Lemaréchal. *Handbooks in Operations Research and Management Science*, volume 1, chapter Nondifferentiable Optimization, pages 529–572. Elsevier Science Publishers, 1989.

226. V. Lesser, J. Pavlin, and E. Durfee. Approximate processing in real-time problem-solving. *AI Magazine*, 9(1):49–61, 1988.

227. V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics-Doklady*, 10:707–710, 1966.

228. G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9:341–352, 1972.

229. J. Levy and H.E. Pashler. Target-distractor phase and selective attention: Idle resources do the devil's work? under review.

230. W. Li and N.M. Nasrabadi. Object recognition based on graph matching implemented by a hopfield–style neural network. In *International Joint Conference on Neural Network, (IJCNN) 1989*, pages 287–290. IEEE Press, 1989.

231. R.P. Lippman. An introduction to computing with neural nets. *IEEE ASSP*

*Magazine*, pages 4–22, April 1987.

232. R.P. Lippman, B. Gold, and M.L. Malpass. A comparison of hamming and hopfield neural nets for pattern classification. Technical Report TR-769, MIT Lincoln Laboratory Technical Report, 1988.

233. W.A. Little. The existence of persistent states in the brain. *Mathematical Bioscience*, 19:101–119, 1974.

234. J.W.S. Liu, K. Lin, W. Shih, and A.C. Yu. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5), 1991.

235. J. Llados, J. Lopez-Krahe, and E. Mart. Hand drawn document understanding using the straight line hough transform and graph matching. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 497–501, 1996.

236. B.C. Love. A computational level theory of similarity. In L.R. Gleitman and A.K. Joshi, editors, *The 22th Annual Meeting of the Cognitive Science Society (COGSCI) 2002*, pages 316–321. Springer, 2000.

237. M.A. Lozano and F. Escolano. ACM attributed graph clustering for learning classes of images. In E. Hancock and M. Vento, editors, *Graph Based Representations in Pattern Recognition. 4th IAPR International Workshop, GbRPR 2003*, LNCS 2726, pages 247–258. Springer-Verlag, 2003.

238. M.A. Lozano and F. Escolano. A significant improvement of softassign with diffusion kernels. In A. Fred, T. Caelli, R.P.W. Duin, A. Campilho, and D. de Ridder, editors, *Proceedings of the Joint IAPR International Workshops Structural, Syntactic, and Statistical Pattern Recognition (SSPR and SPR) 2004*, LNCS 3138, pages 76–84. Springer, 2004.

239. D.G. Luenberger. *Introduction to Linear and Nonlinear Programming*. Addison Wesley, 1973.

240. L. Luksan and J. Vlcek. Introduction to nonsmooth analysis. theory and algorithms. Technical Report DMSIA, Serie Didattica 1/2000, Universita degli Studi di Bergamo, Bergamo, 2000.

241. W. Lukutin and C. Vorwoold. Hierarchisches Graphmatching mit Neuronalen Netzen. Unpublished technical report, 2003.

242. A. Lumini, D. Maio, and D. Maltoni. Inexact graph matching for fingerprint classification. *Machine GRAPHICS and VISION, Special issue on Graph Transformations in Pattern Generation and CAD*, 8(2):231–248, 1999.

243. B. Luo and E.R. Hancock. Structural graph matching using the em algorithm and singular value decomposition. In *Asian Conference on Computer Vision 2002*, pages 75–80, 2002.

244. B. Luo, R.C. Wilson, and E.R. Hancock. Spectral embedding of graphs. *Pattern Recognition*, 36(10):2213–2223, 2003.

245. Z. Luo. On the convergence of the LMS algorithm with adaptive learning rate for feedforward network. *Neural Computing*, 3:226–245, 1991.

246. Z.Q. Luo and P. Tseng. Analysis of an approximate gradient projection method with applications to the backpropagation algorithm. *Optimization Methods and Software*, 4:85–101, 1994.

247. W. Maass. On the computational power of winner-take-all. *Neural Computation*, 12(11):2519–2536, 2000.

248. E.M. Macambira and C.C. de Souza. The edge-weighted clique problem: valid inequalities, facets, and polyhedral computations. Technical Report IC-97-14, Instituto Computacao, Universidade Estadual de Campinas, Brazil, 1996.

249. E. Majani, R. Erlanson, and Y. Abu-Mostafa. On the k-winner-take-all network. *Advances in Neural Information Processing Systems*, 1:634–642, 1989.

250. M.M. Mäkelä and P. Neittaanmäki. *Nonsmooth optimization: Analysis and algorithms with applications to optimal control*. World Scientific Publishing Company, Singapore, 1992.

251. O.L. Mangasarian and M.V. Solodov. Serial and parallel backpropagation convergence via nonmonotone perturbed minimization. *Optimization Methods and Software*, 4(2), 1994.

252. C.M. Marcus and R.M. Westervelt. Dynamics of iterated-map neural networks. *Physical Review A*, 40(1):501–504, 1989.

253. W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

254. J.J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, pages 23–34, 12.

255. B.D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

256. T. Meiers, I. Keller, and T. Sikora. Image visualization and navigation based on MPEG-7 descriptors. In *Proceedings of Conference on Augmented, Virtual Environments and 3D Imaging, EUROIMAGE*, 2001.

257. O. Meincke. Hierarchisches Strukturmatching mit Winner-Takes-All-Netzen. Master's thesis, Technical University of Berlin, 2002.

258. B. Messmer and H. Bunke. A network based approach to exact and inexact graph matching. Technical Report IAM-93-021, University of Bern, 1993.

259. P. Michel and J.P. Penot. Calcul sous-diff'erential pour des fonctions lipschitziennes et non lipschitziennes. *Comptes Rendues de l'Academie des Sciences Paris*, 298:269–27, 1984.

260. A. Micheli, F. Potera, and A. Sperduti. A preliminary empirical comparision of recursive neural networks and tree kernel methods on regression tasks for tree structured domains. *Neurocomputing*, 64:73–92, 2005.

261. A. Micheli, A. Sperduti, A. Starita, and A.M. Bianucci. Analysis of the internal representations developed by neural networks for structures applied to

quantitative structure-activity relationship studies of benzodiazepines. *Journal of Chemical Information and Computer Sciences (ACS Publications)*, 41, 2001.

262. A. Micheli, A. Sperduti, A. Starita, and A.M. Bianucci. *Soft Computing Approaches in Chemistry*, chapter A Novel Approach to QSPR/QSAR Based on Neural Networks for Structures, pages 265–296. Springer, 2003.
263. H. Minc. Upper bounds for permanents of (0,1)-matrices. *Bulletin of the American Mathematical Society*, pages 789–791, 1963.
264. M. Minoux. *Mathematical Programming: Theory and Algorithms*. John Wiley & Sons, 1986.
265. E. Mjolsness, G. Gindi, and P. Anandan. Optimization in model matching and perceptual organization. *Neural Computation*, 1:218–229, 1989.
266. B. Moore. ART 1 and pattern clustering. In G. Hinton, D. Touretzky, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School, Pittsburgh*, pages 174–185. Morgan Kaufmann, 1988.
267. B.S. Mordukhovich. Maximum principle in problems of time optimal control with nonsmooth constraints. *Journal of Applied Mathematics and Mechanics*, 40:960–969, 1976.
268. B.S. Mordukhovich. *Approximation Methods in Problems of Optimization and Control*. Nauka, Moscow, 1988.
269. H.L. Morgan. The generation of a unique machine description for chemical structures. *Journal of Chemical Documentation*, 5:107–112, 1965.
270. F. Moscheni, S. Bhattacharjee, and M. Kunt. Spatiotemporal segmentation based on region merging. *IEEE Transactions on Patern Analysis and Machine Intelligence*, 20(9):897–915, 1998.
271. T.S. Motzkin and E.G. Strauss. Maxima for graphs and a new proof of a Theorem of Turan. *Canadian Journal of Mathematics*, 17:533–540, 1965.
272. R. Myers, R.C. Wilson, and E.R. Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, 2000.
273. A. Nedić and D.P. Bertsekas. Convergence rate of incremental subgradient algorithms. *Stochastic Optimization: Algorithms and Applications*, pages 263–304, 2000.
274. R.M. Nosofsky. Stimulus bias, asymmetric similarity, and classification. *Cognitive Psychology*, 13:87–108, 1991.
275. G.B. Orr and K.R. Müller, editors. *Neural Networks: Tricks of the Trade*. Springer, 1998.
276. P.M. Pardalos and J. Xu. The maximum clique problem. *Journal of Global Optimization*, 4:301–328, 1994.
277. E. Pekalska. *Dissimilarity Representations in Pattern Recognition. Concepts, Theory and Applications*. PhD thesis, Delft University of Technology, 2005.

278. E. Pekalska, P. Paclik, and R.P.W. Duin. A generalized kernel approach to dissimilarity-based classification. *Journal of Machine Learning Research, Special Issue on Kernel Methods*, 2(2):175–211, 2002.

279. M. Pelillo. A unifying framework for relational structure matching. In *Proceedings of the 14th International Conference on Pattern Recognition (ICPR) 1998*, pages 1316–1319, 1989.

280. M. Pelillo. Replicator equations, maximal cliques, and graph isomorphism. *Neural Computation*, 11(8):1933–1955, 1999.

281. M. Pelillo. Replicator equations, maximal cliques, and graph isomorphism. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 550–556. MIT Press, Cambridge, MA, 1999.

282. M. Pelillo. Matching free trees, maximal cliques, and monotone game dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1535–1541, 2002.

283. M. Pelillo and A. Jagota. Feasible and infeasible maxima in a quadratic program for maximum clique. *Journal of Artificial Neural Networks*, 2:411–420, 1995.

284. M. Pelillo, K. Siddiqi, and S.W. Zucker. Matching hierarchical structures using association graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1105–1120, 1999.

285. C. Peterson and B. Soderberg. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 1:2–33, 1989.

286. C. Peterson and B. Soderberg. *The Handbook of Brain Research and Neural Networks*, chapter Neural Optimization. Bradford Books/The MIT Press, 1998.

287. E.G.M. Petrakis and C. Faloutsos. Similarity searching in medical image databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):435–447, 1997.

288. P. Pipenbacher, A. Schliep, S. Schneckener, A. Schnhuth, D. Schomburg, and R. Schrader. ProClust: Improved clustering of protein sequences with an extended graph-based approach. *Bioinformatics*, 1(1):1–10, 2002.

289. B.T. Polyak. *Introduction to Optimization*. Optimization Software Inc., NY, 1987.

290. A. Rangarajan, S. Gold, and E. Mjolsness. A novel optimizing network architecture with applications. *Neural Computation*, 8(5):1041–1060, 1996.

291. A. Rangarajan and E. Mjolsness. A Lagrangian relaxation network for graph matching. *IEEE Transactions on Neural Networks*, 7(6):1365–1381, 1996.

292. A.C. Rao and D.V. Raju. Application of the hamming number technique to detect isomorphism among kinematic chains and inversions. *Mechanism and*

*Machine Theory*, 26(1):55–75, 1991.

293. S.S. Ravi, D.J. Rosenkrantz, and G.K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.
294. J.W. Raymond, E.J. Gardiner, and P. Willett. Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm. *Journal of Chemical Information and Computer Sciences*, 42(2):305–316, 2002.
295. J.W. Raymond, E.J. Gardiner, and P. Willett. RASCAL: Calculation of graph similarity using maximum common edge subgraphs. *Computer Journal*, 45(6):631–644, 2002.
296. J.W. Raymond and P. Willett. Efectiveness of graph-based and fingerprint-based similarity measures for virtual screening of 2d chemical structure databases. *Journal of Computer-Aided Molecular Design*, 16:59–71, 2002.
297. R.C. Read and D.G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, pages 339–363, 1977.
298. M.M. Richter. Classification and learning of similarity measures. In *Proceedings of the Jahrestagung der Gesellschaft für Klassifikation: Studies in Classification, Data Analysis and Knowledge Organisation*. Springer, 1992.
299. H. Ritter. A spatial approach to feature linking. In *Proc. INNC'90 International Neural Network Conference Paris*, volume 2, pages 898–901, 1990.
300. H. Ritter and K. Schulten. Kohonen's Self-Organizing Maps: Explorating their computational capabilities. In *IEEE International Conference on Neural Networks*, pages 109–116. New York: IEEE, 1988.
301. R.T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, NJ, 1970.
302. F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.
303. F. Rosenblatt. *Principles of Neurodynamics*. Spartan Book, 1962.
304. A. Rosenfeld, R. Hummel, and S. Zucker. Scene labeling by relaxation operations. *IEEE Transactions on Systems, Man and Cybernetics*, 6:420–433, 1976.
305. N. Rouche, P. Habets, and M. Laloy. *Stability Theory by Liapunov's Direct Method*. Springer, 1977.
306. G. Rücker and C. Rücker. On topological indices, boiling points, and cycloalkanes. *Journal of Chemical Information and Computer Sciences*, 39(5):788–802, 1999.
307. S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.
308. J. Russell and S. Zilberstein. Composing real-time systems. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 212–217, 1991.

309. A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:353–362, 1983.

310. B. Le Saux and H. Bunke. Feature selection for graph-based image classifiers. In *IAPR Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA'05)*, 2005 (to appear).

311. K. Schädler. *Die Ermittlung struktureller Ähnlichkeit und struktureller Merkmale bei komplexen Objekten: Ein konnektionistischer Ansatz und seine Anwendungen.* PhD thesis, Dept. of Computer Science, Technical University of Berlin, 1999. In DISKI 228, infix, Akademische Verlagsgesellschaft Aka GmbH, Berlin, 2000.

312. K. Schädler and F. Wysotzki. Klassifizierungslernen mit Hilfe spezieller Hopfield-Netze. In W. Dilger, M. Schlosser, J. Zeidler, and A. Ittner, editors, *Procceedings of the Fachgruppentreffen Maschinelles Lernen der GI-Fachgruppe 1.1.3*, Chemnitzer Informatik-Berichte CSR-96-06. Technische Universität Chemnitz-Zwickau, 1996.

313. K. Schädler and F. Wysotzki. Theoretical foundations of a special neural net approach for graphmatching. Technical Report 96-26, Dept. of Computer Science, TU Berlin, 1996.

314. K. Schädler and F. Wysotzki. Comparing structures using a Hopfield-style neural network. *Applied Intelligence*, 11:15–30, 1999.

315. C. Schellewald and C. Schnorr. Subgraph matching with semidefinite programming. Technical report, Dept. of Mathematics and Computer Science, University of Mannheim, Germany, 2002.

316. A. Schenker, M. Last, H. Bunke, and A. Kandel. Comparison of distance measures for graph-based clustering of documents. In E. Hancock and M. Vento, editors, *Graph Based Representations in Pattern Recognition. 4th IAPR International Workshop, GbRPR 2003*, LNCS 2726, pages 203–213. Springer-Verlag, 2003.

317. A. Schenker, M. Last, H. Bunke, and A. Kandel. Comparison of algorithms for web document clustering using graph representaions of data. In A. Fred, T. Caelli, R.P.W. Duin, A. Campilho, and D. de Ridder, editors, *Proceedings of the Joint IAPR International Workshops Structural, Syntactic, and Statistical Pattern Recognition (SSPR and SPR) 2004*, LNCS 3138, pages 190–197. Springer, 2004.

318. S. Scholtes. Introduction to piecwise differentiable equations. Habilitation Thesis, Preprint No. 54/1994, Institut für Statistik und Mathematische Wirtschaftstheorie, Universität Karlsruhe, Germany, 1994.

319. H. Schülzke, M. Schade, and E. Chandravati. Hierarchisches Graphmatching mit Neuronalen Netzen. Unpublished technical report, 2005.

320. G. Scott and H. Longuet-Higgins. An algorithm for associating the features

of two patterns. In *Proceedings of the Royal Society London*, volume B244, pages 21–26, 1991.

321. O. G. Selfridge. Pandemonium: A paradigm for learning. In D.V. Blake and A.M. Uttley, editors, *Proceedings of the Symposium on Mechanisation of Thought Processes*, pages 511–529. H.M. Stationary Office, London, 1959.
322. S.M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.
323. L.G. Shapiro and J.M. Brady. Feature-based correspondence: An eigenvector approach. *Image and Vision Computing*, 10(5):283–288, 1992.
324. L.G. Shapiro and R.M. Haralick. Structural descriptions and inexact matching. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 3(5):514–519, 1981.
325. L.G. Shapiro and R.M. Haralick. A metric for comparing relational descriptions. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 7(1):90–94, 1985.
326. K. Shearer, H. Bunke, and S. Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition*, 34:1075–1091, 2001.
327. S. Shekhar and S. Dutta. inimizing response times in real time planning and search. In *Proceedings of 11th International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 238–242, 1989.
328. A. Shokoufandeh and S. Dickinson. *Graph-Theoretical Methods in Computer Vision*, pages 148–174. LNCS 2292. Springer, 2002.
329. N.Z. Shor. *Minimization Methods for Nondifferentiable Functions*. Springer, 1985.
330. K. Siddiqi, A. Shokoufandeh, S. Dickinson, and S. Zucker. Shock graphs and shape matching. *International Journal of Computer Vision*, 30:1–24, 1999.
331. T. Sikora. The MPEG-7 visual standard for content description — an overview. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6), 2001.
332. P.D. Simić. Constrained nets for graph matching and other quadratic assignment problems. *Neural Computation*, 3:268–281, 1991.
333. H.A. Simon. *Models of bounded rationality,*. MIT Press, 1982.
334. M. Singh, A. Chatterjee, and S. Chaudhury. Matching structural shape descriptions using genetic algorithms. *Pattern Recognition*, 30(9):1451–1462, 1997.
335. R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Annals of Mathematical Statistics*, 35:876–879, 1964.
336. K.A. Smith. Neural networks for combinatorial optimisation: A review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–

34, 1999.

337. F. Sobik. Graphmetriken und Klassifikation strukturierter Objekte. In *Beiträge zur angewandten Graphentheorie*. ZKI-Information. Berlin, 1982.

338. M.V. Solodov and S.K. Zavriev. Error stability properties of generalized gradient-type algorithms. *Journal of Optimization Theory and Applications*, 98(3):663–680, 1998.

339. H. Späth. Heuristically determining cliques of given cardinality and with minimal cost within weighted complete graphs. *Zeitschrift für Operations Research*, 29:125–131, 1985.

340. A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

341. A. Srinivasan, S. Muggleton, R. King, and M. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the Fourth International Workshop on Inductive Logic Programming*, number 237 in GMD Studien, pages 217–232, 1994.

342. A. Srinivasan, S. Muggleton, R. King, and M. Sternberg. Theories for mutagenicity: a study of first-order and feature based induction. *Journal of Artificial Intelligence*, 85(1,2):277–299, 1996.

343. D. Steinhausen and K. Langer. *Clusteranalyse: Einführung in Methoden und Verfahren der automatischen Klassifikation*. Walter de Gruyter, 1977.

344. P. Suganthan. Structural pattern recognition using genetic algorithms. *Pattern Recognition*, 35(9):1883–1893, 2002.

345. P. Suganthan, E. Teoh, and D. Mital. Pattern recognition by graph matching using potts mft networks. *Pattern Recognition*, 28:997–1009, 1995.

346. P. Suganthan and H. Yan. Recognition of handprinted chinese characters by constrained graph matching. *Image and Vision Computing*, 16(3):191–201, 1998.

347. J.P.F. Sum and P.K.S. Tam. Note on the maxnet dynamics. *Neural Computation*, 8(3):491–499, 1996.

348. H. Szu and R. Hartley. Fast simulated annealing. *Physics Letters A*, 122:157–162, 1987.

349. Y. Takahashi, Y. Satoh, H. Suzuki, and S. Sasaki. Recognition in largest common structural fragment among a variety of chemical structures. *Analytical Science*, 3:23–28, 1987.

350. Y. Takefuji and H. Szu. Design of parallel distributed Cauchy machines. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN) 1989*, volume 1, pages 529–532, 1989.

351. O.N. Temkin, A.V. Zeigarnik, and D. Bonchev. *Chemical Reaction Networks: A Graph-Theoretical Approach*. CRC Press, 1996.

352. A. Torsello. *Matching Hierarchical Structures for Shape Recognition*. PhD

thesis, Dept. of Computer Science, University of York, UK, 2004.

353. A. Torsello and E.R. Hancock. Efficiently computing weighted tree edit distance using relaxation labeling. In A.K. Jain M. Figueiredo, J. Zerubia, editor, *Energy Minimization Methods in Computer Vision and Pattern Recognition: Third International Workshop, EMMCVPR 2001*, volume 2134 of *LNCS*, pages 438–453. Springer, 2001.

354. A. Torsello, D. Hidovic, and M. Pelillo. Four metrics for efficiently comparing attributed trees. In *Proceedings of the 17th International Conference on Pattern Recognition*, volume 2, pages 467–470, 2004.

355. J.S. Treiman. Finite dimensional optimality conditions: B-gradients. *Journal of Optimization Theory and Applications*, 62:139–150, 1989.

356. J.S. Treiman. Optimal control with small generalized gradients. *SIAM Journal of Control and Optimization*, 28:720–732, 1990.

357. A.M. Treisman. Strategies and models of selective attention. *Psychological Review*, 76(3):282–299, 1969.

358. W.H. Tsai and K.S. Fu. Error-correcting isomorphism of attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 9:757–768, 1979.

359. W.H. Tsai and K.S. Fu. Subgraph error-correcting isomorphism for syntactic pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:48–62, 1983.

360. P. Tseng. An incremental gradient(-projection) method with momentum term and adaptive stepsize rule. *SIAM Journal on Optimization*, 2:506–531, 1998.

361. A. Tversky. Feature of similarity. *Psychological Review*, 84(4):327–350, 1977.

362. J.R. Ullman. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

363. S. Umeyama. An eigen decomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.

364. A. Üretmen. Hierarchisches Graph Matching mit Neuronalen Netzen. Master's thesis, Technical University of Berlin, 2004.

365. L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

366. P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, 1987.

367. B.J. van Wyk. *Kronecker Product, Successive Projection, and Related Graph Matching Algorithms*. PhD thesis, University of the Witwatersrand, Johannesburg, 2003.

368. B.J. van Wyk and M.A. Van Wyk. The spherical approximation graph match-

ing algorithm. In *Proceedings of the International Workshop on Multidisciplinary Design Optimisation*, pages 280–288, 2000.

369. B.J. van Wyk, M.A. Van Wyk, and F. Virolleau. The CGGM algorithm and its DSP implementation. In *Proceedings of the 3rd European DSP Conference on Education and Research*, pages CD–ROM, 2000.

370. M.A. van Wyk and J. Clark. An algorithm for approximate least-squares attributed graph matching. *Problems in Applied Mathematics and Computational Intelligence*, pages 67–72, 2000.

371. M.A. van Wyk, T.S. Durrani, and B.J. van Wyk. A RKHS interpolator-based graph matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):988–995, 2003.

372. V. Venkateswar and R. Chellappa. Hierarchical stereo and motion correspondence using feature groupings. *International Journal of Computer Vision*, 15:245–269, 1995.

373. A.Y. Chervonenkis V.N. Vapnik. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16:264–280, 1971.

374. C. von der Malsburg. Pattern recognition by labeled graph matching. *Neural networks*, 1:141–148, 1988.

375. S.V. Vrbsky and J.W.S. Liu. Producing monotonically improving approximate answers to database queries. In *Proceedings of the IEEE Workshop on Imprecise and Approximate Computation*, pages 72–76, 1992.

376. R.J. Wallace and E.C. Freuder. Anytime algorithms for constraint satisfaction and sat problems. *SIGART Bulletin*, 7(2), 1996.

377. W.D. Wallis, P. Shoubridge, M. Kraetz, and D. Ray. Graph distances using graph union. *Pattern Recognition Letters*, 22(6/7):701–704, 2001.

378. R.L. Wang, Z. Tang, and Q.P. Cao. An efficient approximation algorithm for finding a maximum clique using hopfield network learning. *Neural Computation*, 15(7):1605–1619, 2003.

379. X. Wang, A. Jagota, F. Botelho, and M. Garzon. Absence of cycles in symmetric neural networks. In D.S. Touretzkya, M. Mozer, and M. Hasselmo, editors, *Proceedings of the 9th Neural Information Processing Systems (NIPS) 1996*. MIT Press, 1996.

380. C. Watkins. Dynamic alignment kernels. Technical report, Department of Computer Science, Royal Holloway, University of London, 1999.

381. J.W. Weibull. *Evolutionary Game Theory*. MIT Press, Cambridge, MA, 1995.

382. M.P. Wellman and C.L. Liu. State-space abstraction for anytime evaluation of probabilistic networks. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI'94)*, pages 567–574, 1994.

383. H. Wersing. *Spatial Feature Binding and Learning in Competitive Neural Layer Architectures*. PhD thesis, Faculty of Technology, University Bielefeld, 2000.

384. M.L. Williams, R.C. Wilson, and E.R. Hancock. Deterministic search for relational graph matching. *Pattern Recognition*, 32(7):1255–1271, 1999.

385. G.V. Wilson and G.S. Pawley. On the stability of the travelling salesman problem algorithm of Hopfield and Tank. *Biological Cybernetics*, 58:63–70, 1988.

386. R.C. Wilson and E.R. Hancock. Structural matching by discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):634–648, 1997.

387. F. Wysotzki. Artifical intelligence and artifical neural nets. In L. Budach, editor, *Neural Informatics. Proceedings of an International Workshop*, 12/1989, pages 43–51. Akademie der Wissenschaften der DDR, Berlin, GDR, 1989.

388. F. Wysotzki. Artifical intelligence and artifical neural nets. In *Proceedings of the 1st Workshop on Artificial Intelligence*, Shanghai, September 1990. TU Berlin and Jiao Tong University Shanghai.

389. F. Wysotzki. Structural matching with artificial neural networks. In *Wissenschaftliche Zeitschrift*, volume 37, pages 113–119. Technische Hochschule Ilmenau, GDR, 1991.

390. J. Xu. GMA: A generic match algorithm for structural homomorphism, isomorphism, and maximal common substructure match and its applications. *Journal of Chemical Information and Computer Science*, 36(1):25–34, 1996.

391. L. Xu and I. King. A PCA approach for fast retrieval of structural patterns in attributed graphs. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 31(5):812–817, 2001.

392. L. Xu and E. Oja. Improved simulated annealing, boltzmann machine, and attributed graph matching. In L.B. Almeida and C.J. Wellekens, editors, *Neural Networks*, pages 151–161. Springer, 1990.

393. Y. Yamada, E. Tomita, and H. Takahashi. A randomized algorithm for finding a near-maximum clique and its experimental evaluations. In *Transactions of IEICE Japan*, volume J76-D-I(2), pages 46–53, 1993.

394. J.-C. Yen, F.-J. Chang, and S. Chang. A new winner-take-all architecture in artifical neural networks. *IEEE Transactions on Neural Networks*, 5(5):838–843, 1994.

395. J.-C. Yen and S. Chang. Improved winner-take-all neural network. *Electronic Letters*, 28(7):662–664, 1992.

396. B. Zelinka. On a certain distance between the isomorphism classes of graphs. *Časopis pro pěstováni matematiky*, 100:371–373, 1975.

397. W.P. Ziemer. *Weakly differentiable functions. Sobolev spaces and functions of bounded variation*. Springer, 1989.

398. S. Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms*. PhD thesis, University of California, Berkely, 1993.

399. S. Zilberstein. Resource-bounded sensing and planning in autonomous systems. *Autonomous Robots*, 3:31–48, 1996.

400. S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.