Community Detection in Hypergraphs

vorgelegt von
Nicolas Neubauer, M.Sc.
aus Berlin

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. Dr. Dr. h.c. Sahin Albayrak
Berichter:	Prof. Dr. Klaus Obermayer
Berichter:	Prof. Dr. Arjen P. de Vries

Tag der wissenschaftliche Aussprache: 26. Juni 2012

Berlin 2012
D 83

Acknowledgements

First of all, I would like to thank my advisor Klaus Obermayer for creating in his Neural Information Processing Group at Technische Universität Berlin the atmosphere of intellectual curiosity to which this thesis owes its existence, and for being consistently supportive all along the way.

Many colleagues in this group have influenced me during the writing of this thesis. In particular I would like to thank: Wendelin Böhmer for many great discussions, and in particular for telling me that my initial graph renderings were incomprehensible, leading to the creation of the custom visualization package described later on. Bartek Ochab for great work together, and for insisting on a proper naming for what's now called hyper-incident connected components. Hendri Murfi for the good time in our office and shared adventures in information retrieval. Johannes Jain for graph- and general science advice. Johannes Mohr and Sambu Seo for their hospitality. Andreas König for scraping valuable data and Matei Leventer for contributing the SVM-based text classifier. Klaus Wimmer and Michael Sibila for the occasional bit of rock'n'roll.

At TU Berlin, I am indebted to the Integrated Graduate Program on Human-Centric Communication (IGP H-C3) for funding my research and the writing of this thesis (the research leading to these results has also received funding from the European Community's Seventh Framework Programme [FP7/2007-2011] under grant agreement nr 21644 (PetaMedia) and via DFG grant no. OB 102/10-2 (Learning Agents for Text Classification)), and within H-C3, in particular to my second advisor Thomas Sikora, who got me in touch with the EU Network of Excellence PeTaMedia, and through that with a whole new group of colleagues sharing a passion for social data. I would also like Danny Pannicke for good chats about writing a dissertation in general, Robert Wetzker for playing a major role in moving my attention to social bookmarking as well as providing the priceless Delicious dataset, and Jérôme Kunegis for being my main address for graph-related discussions.

Furthermore, I would like to thank everyone at VZ.net who made our cooperation such an enjoyable experience, first of all Falk-Florian Henrich for initiating the contact, but also the rest of "Team Pie": Ines Abdallah, Ingo Schramm, Sean Buttinger and Luis Osa. You all spent considerable amounts of time helping me, but also encouraged me with your curiosity and good discussions – besides simply being fun to work with.

Outside of work, I would like to thanks Frank Schumann for regularly engaging his rigorous scientific mind on a topic so unrelated to his own work, and Jan Peters for his gentle but determined motivational impulses, as well as Julia Hesselmann for taking the first look at an early version of this document as well as for general awesomeness. Finally, I'd like to thank my girlfriend Angelika Honerkamp and my mother Roswitha Schäfer-Neubauer for supporting and tolerating me during the creation of this document.

Abstract

Many datasets can be interpreted as graphs, i.e. as elements (nodes) and binary relations between them (edges). Under the label of complex network analysis, a vast array of graph-based methods allows the exploration of datasets purely based on such structural properties. Community detection, as a subfield of network analysis, aims to identify well-connected subparts of graphs. While the grouping of related elements is useful in itself, these groups can furthermore be collapsed into single nodes, creating a new graph of reduced complexity which may better reveal the original graph's macrostructure. Therefore, advances in community detection improve the understanding of complex networks in general.

However, not every dataset can be modelled properly with binary relations – higher-order relations give rise to so-called hypergraphs. This thesis explores the generalization of community detection approaches to hypergraphs. In the focus of attention are social bookmarking datasets, created by users of online bookmarking services who assign freely chosen keywords, so-called "tags", to documents. This "tagging" creates, for each tag assignment, a ternary connection between the user, the document, and the tag, inducing particular structures called 3-partite, 3-uniform hypergraphs (henceforth called 3,3- or more generally k,k-hypergraphs). The question pursued here is how to decompose these structures in a formally adequate manner, and how this improves the understanding of these rich datasets.

First, a generalization of connected components to k, k-hypergraphs is proposed. The standard definition of connected components here rather uninformatively assigns almost all elements to a single giant component. The generalized so-called hyperincident connected components, however, show a characteristic size distribution on the social bookmarking datasets that is disrupted by, e.g., spamming activity – demonstrating a link between behavioural patterns and structural features that is further explored in the following.

Next, the general topic of community detection in k, k-hypergraphs is introduced. Three challenges are posited that are not met by the naive application of standard techniques, and three families of synthetic hypergraphs are introduced containing increasingly complex community setups that a successful detection approach must be able to identify.

The main methodical contribution of this thesis consists of the following development of a multi-partite (i.e. suitable for k, k-hypergraphs) community detection algorithm. It is based on modularity optimization, a well-established algorithm to detect communities in non-partite, i.e. "normal" graphs. Starting from the simplest approach possible, the method is successively refined to meet the previously defined as well as empirically encountered challenges, culminating in the definition of the "balanced multi-partite modularity".

Finally, an interactive tool for exploring the obtained community assignments is introduced. Using this tool, the benefits of balanced multi-partite modularity can be shown: Intricate patters can be observed that are missed by the simpler approaches. These findings are confirmed by a more quantitative examination: Unsupervised quality measures considering, e.g., compression document the advantages of this approach on a larger number of samples.

To conclude, the contributions of this thesis are twofold. It provides practical tools for the analysis of social bookmarking data, complemented with theoretical contributions, the generalization of connected components and modularity from graphs to k, k-hypergraphs.

Zusammenfassung

Viele Datensätze können als Graphen aufgefasst werden, d.h. als Elemente (Knoten) und binäre Verbindungen zwischen ihnen (Kanten). Unter dem Begriff der "Complex Network Analysis" sammeln sich eine ganze Reihe von Verfahren, die die Untersuchung von Datensätzen allein aufgrund solcher struktureller Eigenschaften erlauben. "Community Detection" als Untergebiet beschäftigt sich mit der Identifikation besonders stark vernetzter Teilgraphen. Über den Nutzen hinaus, den eine Gruppierung verwandter Element direkt mit sich bringt, können derartige Gruppen zu einzelnen Knoten zusammengefasst werden, was einen neuen Graphen von reduzierter Komplexität hervorbringt, der die Makrostruktur des ursprünglichen Graphen unter Umständen besser hervortreten lässt. Fortschritte im Bereich der "Community Detection" verbessern daher auch das Verständnis komplexer Netzwerke im allgemeinen.

Nicht jeder Datensatz lässt sich jedoch angemessen mit binären Relationen darstellen – Relationen höherer Ordnung führen zu sog. Hypergraphen. Gegenstand dieser Arbeit ist die Verallgemeinerung von Ansätzen zur "Community Detection" auf derartige Hypergraphen. Im Zentrum der Aufmerksamkeit stehen dabei "Social Bookmarking"-Datensätze, wie sie von Benutzern von "Bookmarking"-Diensten erzeugt werden. Dabei ordnen Benutzer Dokumenten frei gewählte Stichworte, sog. "Tags" zu. Dieses "Tagging" erzeugt, für jede Tag-Zuordnung, eine ternäre Verbindung zwischen Benutzer, Dokument und Tag, was zu Strukturen führt, die 3-partite, 3-uniforme (im folgenden 3,3-, oder allgemeiner k, k-) Hypergraphen genannt werden. Die Frage, der diese Arbeit nachgeht, ist wie diese Strukturen formal angemessen in "Communities" unterteilt werden können, und wie dies das Verständnis dieser Datensätze erleichtert, die potenziell sehr reich an latenten Informationen sind.

Zunächst wird eine Verallgemeinerung der verbundenen Komponenten für k, k-Hypergraphen eingeführt. Die normale Definition verbundener Komponenten weist auf den untersuchten Datensätzen, recht uninformativ, alle Elemente einer einzelnen Riesenkomponente zu. Die verallgemeinerten, so genannten hyper-inzidenten verbundenen Komponenten hingegen zeigen auf den "Social Bookmarking"-Datensätzen eine charakteristische Größenverteilung, die jedoch bspw. von Spam-Verhalten zerstört wird – was eine Verbindung zwischen Verhaltensmustern und strukturellen Eigenschaften zeigt, der im folgenden weiter nachgegangen wird.

Als nächstes wird das allgemeine Thema der "Community Detection" auf k, k-Hypergraphen eingeführt. Drei Herausforderungen werden definiert, die mit der naiven Anwendung bestehender Verfahren nicht gemeistert werden können. Außerdem werden drei Familien synthetischer Hypergraphen mit "Community"-Strukturen von steigender Komplexität eingeführt, die prototypisch für Situationen stehen, die ein erfolgreicher Detektionsansatz rekonstruieren können sollte.

Der zentrale methodische Beitrag dieser Arbeit besteht aus der im folgenden dargestellten Entwicklung eines multipartiten (d.h. für k, k-Hypergraphen geeigneten) Verfahrens zur Erkennung von "Communities". Es basiert auf der Optimierung von Modularität, einem etablierten Verfahrung zur Erkennung von "Communities" auf nicht-partiten, d.h. "normalen" Graphen. Ausgehend vom einfachst möglichen Ansatz wird das Verfahren itera-

tiv verfeinert, um den zuvor definierten sowie neuen, in der Praxis aufgetretenen Herausforderungen zu begegnen. Am Ende steht die Definition der “ausgeglichenen multi-partiten Modularität”.

Schließlich wird ein interaktives Werkzeug zur Untersuchung der so gewonnenen “Community”-Zuordnungen vorgestellt. Mithilfe dieses Werkzeugs können die Vorteile der zuvor eingeführten Modularität demonstriert werden: So können komplexe Zusammenhänge beobachtet werden, die den einfacheren Verfahren entgehen. Diese Ergebnisse werden von einer stärker quantitativ angelegten Untersuchung bestätigt: Unüberwachte Qualitätsmaße, die bspw. den Kompressionsgrad berücksichtigen, können über eine größere Menge von Beispielen die Vorteile der ausgeglichenen multi-partiten Modularität gegenüber den anderen Verfahren belegen.

Zusammenfassend lassen sich die Ergebnisse dieser Arbeit in zwei Bereiche einteilen: Auf der praktischen Seite werden Werkzeuge zur Erforschung von “Social Bookmarking”-Daten bereitgestellt. Demgegenüber stehen theoretische Beiträge, die für Graphen etablierte Konzepte – verbundene Komponenten und “Community Detection” – auf k, k-Hypergraphen übertragen.

Contents

1. Introduction

1.1. Motivation

Machines and human beings have formed an intricate symbiosis. We rely on a global network of machines for an ever-growing amount of tasks, with vast capacities for data processing and transmission at our disposal. In turn, algorithms increasingly rely on us to attach meaning to these data. Large amounts of often tiny traces of our meaning-creating behaviours are harvested, fed back into computational systems and subjected to their raw processing power, resulting in electronic services that seem to better understand us.

This pattern is at the core of some of the most successful services created in the recent past. Google's PageRank (Page et al., 1998) algorithm attributes autority scores to individual web pages based on incoming links from other sites. Amazon's recommendation system has become one of the site's most prominent features, computing item similarities based on purchasing patterns. In both cases, (human-generated) contents or objects are archived into (machine-generated) indices, which become more accessible through the collection of (human-generated) relevance statements like hyperlinks or purchases and (machine-generated) extrapolations.

A less antropomorphic way to put this is to simply observe that interfaces increasingly adapt their navigational structures through the collective actions of their users. This approach has in fact been named "social navigation" almost 20 years ago:

> In social navigation, movement from one item to another is provoked as an artifact of the activity of another or a group of others. (Dourish and Chalmers, 1994)

This thesis focusses on extracting structures meaningful for social navigation from a particular type of user behaviour, social bookmarking. Social bookmarking sites allow users to "tag" (freely annotate) documents found on the web, creating rich annotations of web content and leading to millions of document/user/tag triples. These triples can be interpreted as edges of generalized graphs called 3-uniform 3-partite hypergraphs – graphs whose edges connect three elements from three distinct sets –, structures to which many concepts from graph theory and network analysis have been generalized only recently, or not yet at all.

Methodologically, this thesis focusses on decomposing these hypergraphs into connected components and communities, reducing their complexity and facilitating their exploration. These efforts should however have implications beyond the concrete application domain of social bookmarking data. On the theoretical side, missing generalizations are provided, while on the applied side, various other datasets, both in information retrieval and in other areas, can be interpreted as 3-uniform 3-partite hypergraphs.

More background to these areas will be provided in the next section, before the chapter concludes with an overview of examined datasets, a table of symbols and an outline of the rest of this document.

1.2. Background and Related Work

This thesis is located at the intersection of the three topics of social bookmarking, community detection and hypergraph theory. In the following, the relevant background in the individual areas will be introduced, along with work in pairwise intersections of these areas where appropriate. Finally, the work most closely related to this thesis, situated in the intersection of all three topics, will be reviewed.

1.2.1. Social Bookmarking

Saving "bookmarks" is a basic functionality of web browsers: Users can store their favorite URLs for easy later access. When the website Delicious started its service in 2003, it had the simple purpose of letting users take their bookmarks from their local drives to the web. In what may have initially seemed like a minor design decision, Delicious no longer supported the hierarchical folder structure traditionally used for managing bookmarks, instead offering users to attach to each bookmark a number of freely chosen keywords, so-called "tags".

The combination of central storage and informal annotation would soon become a well-known pattern for services and interfaces called "social bookmarking" or "social tagging".

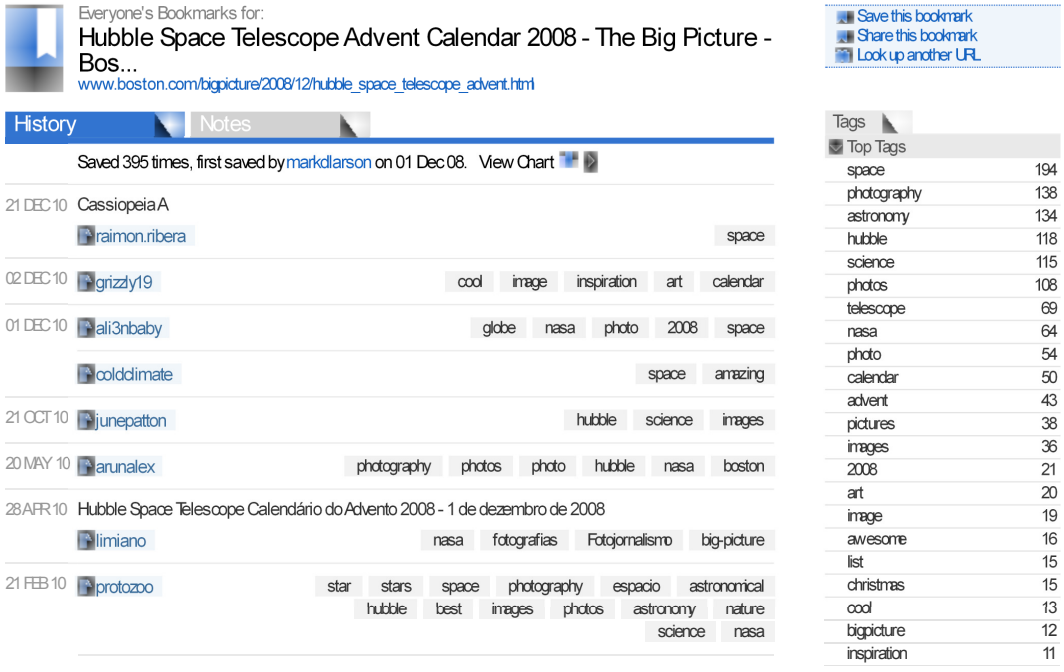


Figure 1.1.: Delicious interface showing the data for a particular URL, showing eight (of 395) users' detailed entries on the left (tags on gray background) and the tag distribution on the right

One of its advantages is that the flat annotations enabled new navigation interfaces, ridden of the heavy physical connotations of the folder metaphor. Each tag can be selected and act as as "top level folder". Besides showing the associated documents, all other tags co-assigned to some documents with this tag can be displayed to refine the search, creating ad-hoc hierarchical structures as required by the user.

Moreover, the collection of many user's bookmarks in a central place allowed for social discovery: Users could see which other users have tagged shared documents or which other documents have been tagged with a tag of their interest. Following these possibilities to a more global view, the aggregated datasets, thought of as commented traces of online behavior, have become the subject of a substantial body of research. The discussion below will trace these developments, starting from an individual point of view and then broadening its scope to more collective phenomena.

1.2.1.1. A Typical Interface

Let us first take a look at the interface of Delicious, more precisely the detail screen for a particular URL (delicious.com, 2010). This not only ensures the principle of social bookmarking services is properly demonstrated, but will also allow some initial observations about the aggregated data that will be drawn upon later on.

By featuring Delicious that prominently, by the way, I do not intend to make a hard claim about Delicious being the first social bookmarking site ever. However, Smith (2007) claims the use of the term "tag" in the specific sense used here goes back to a column name in the first Delicious database. In any case, it is one of the oldest and most successful social bookmarking systems and is therefore used as a running example in this section.

Figure 1.1 shows one of the main screens of Delicious: The overview of all tags given to a particular URL. On the left side, we see a list of users who have bookmarked the URL, including user names (in blue), their notes (above, where applicable) and the tags they have chosen to attach (in gray). On the right, there is a list of the tags most frequently used. Several things are notable about this list:

- First of all, the subject of the actual website (a collection of images obtained via the Hubble space telescope) can be inferred just by looking at the top tags: "space", "photography", and "astronomy".

- While the most popular tag "space" is also found in the page title, tags like "astronomy", "science" or "art" embed the document in a context that seems trivial but might be hard to construct for a machine. Other tags, like "cool", "awesome" or "inspiration" add subjective judgements (Höltschi et al. (2008) claims that subjectivity is a central contribution of social tags, although that aspect will not be further investigated here), while the tag "2008" hints at personal organization schemes of the users that do not relate to the content of the page itself.

- The top tag has been assigned by almost half of the users, whereas the tag at position 10 was only used by slightly above 10% of the users. This is not a coincidence. Golder and Huberman (2006), in one of the earliest and most influential papers on the

topic, demonstrate that this type of long-tailed distribution (i.e. few highly popular tags and quick decay in popularity) is characteristic for social bookmarking systems. They also show that over time (or more precisely, as more and more users add tags), the tag distribution stabilizes: Fewer tags are added and the relative frequencies of the tags become fixed.

In short, a lot of the features that make social bookmarking services interesting to study can already be found on a single screen.

1.2.1.2. Types of Social Bookmarking Systems

The properties of the datasets created through social bookmarking systems depend on certain features of these systems. Marlow et al. (2006) provide different dimensions along which bookmarking systems can differ. Let us take the photo management service Flickr as an example for a tagging-enabled service which strongly differs from Delicious. The dimensions most important for the tagging dynamics are

source Which items are tagged – users' own, system-wide, or global resources? While Flickr provides a tagging of user-provided photographs, Delicious lets users tag arbitray (global) resources.

tagging rights Who can tag resources? Where users own resources, it may make sense to restrict tagging rights to the owners (or whoever they extend those rights to, as it is actually done in Flickr), whereas in other systems, e.g. with global resources as Delicious, anyone should be able to tag any resource.

tagging support How are users supported in tagging documents? Many social bookmarking systems come with a recommendation engine for tags. A simpler support method is just showing other users' tags. These methods may play a crucial role in the observed convergence effects.

Vander Wal (2005) coined the term "Folksonomy" (as a mixture of folks and taxonomy) for the datasets created by social bookmarking systems, and summarizes the main poles in their properties by distinguishing between "broad" (e.g. Delicious) or "narrow" (e.g. Flickr) folksonomies, depending on whether many and few people contribute to the tagging of individual resources. Understanding and exploiting the collective dynamics of social bookmarking systems is at the center of this thesis, so the focus will be on broad folksonomies.

1.2.1.3. Tags as Keywords

Meaningful keywords can significantly speed up retrieval tasks – however, such keywords can be obtained through various sources. Why should we be interested in social tagging as yet another source when we can obtain keywords from the authors of documents, from keyword extraction algorithms or from professional indexing services?

Kipp (2007) has compared keywords provided both by authors and by PubMed with those generated by users in CiteULike, a social bookmarking service for scientific articles which will be discussed in further detail later on. She reports significant differences between user- and expert-generated descriptors. A striking example of added value are tags describing methodological specifics like "family-studies" absent in the more official descriptor system, directly reflecting the focus of the users' interests. In a similar spirit, Al-Khalifa and Davis (2006) compare automatically generated keywords to tags and a manually created index, finding higher similarity between tags and the index than between computer-generated keywords and the index. Heymann and Garcia-Molina (2009) compare tags and Library of Congress Subject Headings, i.e. a taxonomy generated by experts over many years. They find similar vocabularies, but different usage patterns – the overlap between works labelled with similar tags is rather small, in ways that "are ultimately bad for retrieval using expert assigned controlled vocabularies".

While the comparsion of different keyword collections helps illustrate the differences between the mechanisms, actual differences in usefulness for retrieval tasks can only be estimated using such methodology. In contrast, Melenhorst et al. (2008) report an impressive study in which "194 participants tagged a total of 115 videos, while another 140 participants searched the video collection for answers to eight questions" – using either tags or professionally or automatically generated metadata. They conclude that "social tags yield effective retrieval processes, whereas automatically generated metadata do not".

1.2.1.4. Social Dynamics

A large body of research suggests that the reasons for the effectiveness of social bookmarking systems cannot be found by regarding the individual tagger, but instead lie in the dynamics of the social interactions enabled by these systems. More generally, there is a trend of enabling consumers to be also (at least small-scale) content creators, summarized under the heading of "Produsage", a mix between production and usage (Bruns, 2008). Here, however, we want to focus on a more specific discussion of the mechanisms inside social bookmarking systems.

Golder and Huberman (2006) have provided one of the first quantitative studies of social bookmarking systems. As already mentioned above, they famously showed a "stabilization" effect: As the number of users who tag a document grows, the relative frequency of the assigned tags becomes stable. Characteristic frequency/rank distributions in the connectivity of tags were found to be created by tagging behaviour by Cattuto et al. (2007b), who could also show that significant deviations from these patterns were caused by spam entries. Cattuto et al. (2007a) argue that "... users of collaborative tagging systems share universal behaviours that ... appear to follow simple activity patterns". Halpin et al. (2007) and Fu (2008) aim to make explicit these patterns, providing generative models to reproduce the long-tailed distributions found in the data and the tagging processes of individual subjects, respectively. Schifanella et al. (2010), on the other hand, exploit the fact that Flickr and Last.fm allow, in addition to tagging, for explicit social connections. They can show local alignment effects in the vocabulary of connected users, to the extent that similar vocabularies can be used for the prediction of friendships. This makes a strong case for the social

feedback mechanisms at work in these systems. Wetzker et al. (2010) further explore the relation between individual and collective vocabulary and propose a mechanism for "translating" between the two.

Several articles focus on the connecting the collective activity in bookmarking systems to other fields like computational linguistics. Hotho et al. (2006a) have provided early visions for possible applications. In addition to an algorithm called FolkRank for tag recommendation (to be discussed below in more detail), they provide a simple mechanism to mine concept hierarchies from tag co-occurrences. Cattuto et al. (2008) have later combined these approaches through "semantic grounding", i.e. looking up tags in WordNet (Fellbaum, 1998), a machine-readable lexical database annotated with relations like is-synonym-of, is-hyponym-of etc. Glushko et al. (2008) regard tagging datasets as valuable empirical data for research on cultural categorization.

It is impossible to treat all these works in the depth they would deserve. We will encounter some of them again later on – for now however, the goal of this short introduction is rather to provide pointers to the most central papers, as well as to give a first impression of the inherent complexity of the datasets in question.

1.2.1.5. Harnessing the Collective Aspects

One of the simplest and most famous applications using tagging data are the once omnipresent "tag clouds": The most popular tags for, e.g., a set of documents are displayed, with the size of each tag representing its relative frequency. Compared to simply using the most popular tags as a regular keyword list, these visualizations made a first step towards showing the collective nature of the underlying data, by emphasizing the differences in popularity between different tags. While many variations of tag clouds have been conceived, Hassan-Montero and Herrero-Solana (2006) must be noted for proposing an early clustering of tags to make the relative position of tags inside the clouds more meaningful, along with Stefaner (2007), who proposes various ideas for tag-related interfaces, one of which consists of aligning a single user's tags by similarity.

Tag and tag-based recommendation, i.e. recommending sets of tags for a given document or recommending documents for given tags, is another field of applications using tagging data to improve working with tags, see e.g. Jäschke et al. (2007); Symeonidis et al. (2008); Jäschke et al. (2009). Works from this field will be highlighted again later on for their methodological aspects. However, let us here briefly review other tag-based applications that expand their focus to areas outside the social bookmarking domain.

One line of work tries to apply the knowledge contained in social bookmarking datasets to the task of web search. While Bao et al. (2007) report that a variant of PageRank enhanced with this sort of information shows greatly improved results and Yanbe et al. (2007) propose an interface for the integration of web search and tags, Heymann et al. (2008) perform a more quantitative analysis and come to the conclusion that the sheer number of web sites is too large, compared to the number of sites that get annotated. Naaman (2009) has coined the term "Spatio-Tempo-Social" for a more exploratory use of tags and other social data sources: Instead of improving a particular task, he proposes to to employ this information for "Learning from and about Humans with Social Media". The "World Explorer" described

by Ahern et al. (2007) is a great example for this line of work: photos from Flickr which contain both tags and the geographical coordinates of where they were taken are used to created a map overlay describing the most important terms for the displayed region.

The motivation for the work described in this thesis is much in the spirit of these last two sections: Understanding the structures inside tagging datasets, while keeping an eye open for possible applications, navigational or otherwise. My particular methodological focus for this goal lies on community detection, which will be introduced now.

1.2.2. Community Detection

Graphs are discrete structures consisting of nodes V and edges E between those nodes:

Definition 1.1 (Graph) *A* graph $G = (V, E)$ *is a set* V *of* $|V| = N$ nodes *or* vertices *and a set* $E \subset V \times V$ *of* $|E| = M$ *edges. The* adjacency matrix A *of* H *is an* $|V| \times |V|$ *matrix such that* $A_{i,j} = |e \in E : v_i \in e \wedge v_j \in e|$*, i.e. non-zero entries denote adjacent nodes. The* incidence matrix B *of* H *is a* $|V| \times |E|$ *matrix such that* $B_{i,j} = 1$ *if* $e_j \in v_i$*, and 0 otherwise.*

Building on the mathematical branch of graph theory, complex network analysis has emerged in recent years as an interdisciplinary research area, studying the properties of complex systems that can be interpreted as graphs. By interpreting social bookmarking datasets as graphs, we can apply the toolbox of complex network analysis on them. Community detection is the one tool I will focus on in particular.

1.2.2.1. Methods for Community Detection

Community detection, the identification of closely connected groups of nodes in complex networks, has been a vibrant field of research field for the last years (Girvan and Newman, 2002). Identifying such groups has various applications (Newman, 2006): The identification of related elements based on connectivity can be useful in itself, as for communities in social networks or clusters of related documents. Additionally however, related elements can be collapsed into single nodes, creating a new graph of reduced complexity. This may help reveal the original graph's macrostructure and functional or semantic modules. Therefore, recent advances in detecting communities have increased the understanding of complex networks in general.

As pointed out by, e.g., Newman (2006), community detection (also called graph clustering, which is however more ambiguous since it could also relate to the task of clustering different graphs) is related to, but not equivalent to graph partitioning. Graph partitioning enforces a division of the graph into a given number of partitions while minimizing a criterion like the number of inter-partition edges (the "cut size"), typically balanced by a term rewarding similarly sized partitions. This setting is geared towards solving concrete problems with constraints formulated as networks. Community detection is more exploratory in spirit and aims to extract the optimal number and size of well-connected partitions from the data as well. A typical graph partitioning problem might be to seat 12 people on two tables

of six while avoiding friends (adjacent nodes) to be torn apart. The community detection equivalent would come up with its own proposals concerning number and sizes of tables.

What exactly does it mean for a group of nodes to be well-connected? Numerous definitions exist. Each one implies a quality measure for particular groupings which in turn leads to corresponding optimization approaches. "Good" connectivity can be defined, e.g., related to the cut size for approaches inspired by partitioning (Leskovec et al., 2008), as robustness against removal of "bridging" edges (Girvan and Newman, 2002), or by information-theoretic means concerning the efficient coding of trajectories through the graph (Rosvall and Bergstrom, 2007) – please refer to (Schaeffer, 2007; Fortunato, 2010) for in-depth overviews over community detection algorithms, or to (Danon et al., 2005; Lancichinetti and Fortunato, 2009) for quantitative comparisons.

1.2.2.2. Modularity

Throughout this thesis, however, I will focus on and extend a measure called modularity (Newman, 2006). For a given assignment of a graph's vertices into communities, it compares the number of intra-community edges (i.e. edges between elements from the same community) to the *expected* number of such edges, were edges evenly distributed over nodes. Since the property we are looking for in communities is well-connectedness between its members, the former quantity being larger than the latter should correspond to suitable communities.

Definition 1.2 (Modularity) *For a graph $G = (V, E)$, let M be the number of edges $|E|$ and A the adjacency matrix. Let σ be a function assigning vertices V to communities. Then the* modularity *Q of a community assignment σ relative to a null model P is*

$$Q = \frac{1}{2M} \sum_{(i,j) \in V \times V} [A_{ij} - P_{ij}] \delta(\sigma(i), \sigma(j))$$

where $\delta(x, y) = 1$ iff $x = y$ and 0 otherwise (Newman, 2006).

Q rewards community assignments where the actual adjacency (A) is higher than the expected (P) adjacency for all pairs of nodes in a community. The canonical choice for P is the mean adjacency in a "configuration model" graph, where edges are randomly distributed such that a given degree distribution is kept. This amounts to

$$P_{ij} = \frac{k_i k_j}{2M},$$

where k_i is the degree of node i.

Optimizing modularity is a well-established method for community detection, and various efficient methods for this optimization exist, for example a spectral method on the adjacency matrix by Newman (2006), or a bottom-up clustering method proposed by Clauset et al. (2004) and optimized by Wakita and Tsurumi (2007).

At the same time, several criticisms have been voiced concerning modularity. The so-called "resolution limit", e.g. pointed out by Fortunato and Barthélemy (2007) prohibits

the identification of communities below a certain size in large, sparse networks where the expected connectivity becomes so small that a single edge creates above-expectation connectivity, resulting in groups of nodes being falsely classified as communities. This problem is mitigated for my purposes by the fact that the applications introduced later on consider the whole clustering hierarchy obtained by bottom-up modularity optimization, not only a single community assignment. Even if two small communities are merged in the solution for optimal modularity, i.e. a specific point in the clustering tree, further exploration of that tree will still expose the underlying structure.

Another criticism lies in the fact that modularity "typically admits an exponential number of distinct high-scoring solutions and typically lacks a clear global maximum" (Good et al., 2010). While this implies that results returned by modularity optimization should be treated with caution in some contexts, I believe that its widespread use, its intuitive interpretation and the existence of previous work on bipartite generalizations (see below) make modularity an attractive point to start investigating the challenges that will be sketched in the following.

1.2.2.3. Community Detection on Bipartite Graphs

Bipartite graphs are graphs in which edges only connect nodes from two distinct sets:

Definition 1.3 (Bipartite Graph) *A* Bipartite Graph *is a graph* $G = (V, E)$ *whose nodes* V *can be divided into two* partitions V_1, $V_2 : V = V_1 \cup V_2, V_1 \cap V_2 = \emptyset$ *such that* $E \subset V_1 \times V_2$.

Many networks found in real-world settings are bipartite graphs (sometimes also called two-mode networks): Terms that appear in documents, users who buy products, patients exhibiting medical symptoms all imply graphs in which elements from one set are invariably only connected to elements from the other one. While it is possible to treat bipartite graphs simply as graphs, additional insights can often be gained by taking into account the bipartite property where it is known. For community detection in particular, it is typically desirable to obtain distinct sets of communities for the two partitions. It is therefore not surprising to find a vast body of literature on community detection in bipartite graphs, also known as "Co-Clustering" when the graph-theoretical aspect is less prominent than the point of creating two distinct sets of clusterings.

Carrasco et al. (2003) for example have proposed a combined top-down (via graph partitioning) und bottom-up (via a flow-based method) clustering algorithm for advertiser/ keyword graphs. Dhillon et al. (2003); Chakrabarti et al. (2004) aggregate elements into communities in order to minimize the information required to encode bipartite adjacency matrices. Kumar et al. (2008) propose the "k neighborhood graph", connecting only those elements from one domain which share at least k elements in the other. Evans and Lambiotte (2009) proposes a community detection algorithm that supports overlapping communities (i.e. elements can belong to more than one community) by considering the so-called "line graph", i.e. the graph of incident edges, a concept that will be discussed in more depth in Section 2.2.1.2. Liu et al. (2009) emphasize in particular the need, in the context of incremental optimization, to reevaluate the assignments for one partition after updating the assignments in the other.

It will become clear in Section 1.2.3 that in order to deal with bookmarking data, a further modification to the underlying datastructure is required that is somewhat similar to the modification from "normal" to bipartite graphs. Therefore, it is particularly interesting to find, from the vast array of bipartite clustering methods, those which have a systematic relationship to their non-partite baselines. Modularity is a promising candidate in this respect as well: Barber (2007); Guimerà et al. (2007); Murata (2009) all propose generalizations of modularity for bipartite graphs. On the one hand, this shows that modularity isn't transfered easily or unambiguously to the bipartite case; on the other hand, I share these author's (supposed) intuition that it is worth looking for derived measures matching the original's conciseness and elegance. While different bipartite modularity measures are discussed in 4.2.1, we will now get closer to the reason we need a specialized community detection method at all.

1.2.2.4. Social Bookmarking and Community Detection

Social bookmarking datasets are complex, noisy and huge – a 2007 crawl of Delicious by Wetzker et al. (2008) contains about half a billion of tag assignments. As argued above, community detection methods let us view the "bigger picture" of a graph by grouping similar nodes. Therefore, their application on those datasets appears tempting for distilling the latent information contained.

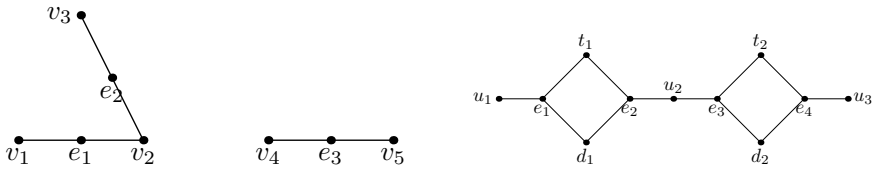
Work explicitly dealing with communities in social bookmarking data often focusses on one type of elements. Ramage et al. (2009) propose a tag-based document clustering, Schifanella et al. (2010) cluster users by their tagging behaviour. Mostly however, tags are clustered, e.g. by Begelman et al. (2006); Halpin et al. (2007); Shepitsen et al. (2008); Wartena and Brussee (2008). Li et al. (2008) cluster tags for social interest discovery, then compare the associated users and urls. Lambiotte and Ausloos (2006) and Zlatić et al. (2009) define generic similarity measures applicable for any partition, however restrict their analysis to the effects on single partitions.

In my view, a full understanding of the involved community structures requires a simultaneous grouping of users, documents and tags. While some work employs bipartite community detection, e.g. Tang and Liu (2010), this does not capture the full content of the involved information either. As it turns out, properly representing the ternary relationship inherent in the data requires an extension of the formal framework considered so far. This extension, the hypergraph, will be introduced in the following.

1.2.3. Hypergraphs

This section introduces hypergraphs, the data structures required to properly represent social bookmarking data in the terminology of networks. Let us start from the obvious question: Why can't we simply use graphs to represent social bookmarking data?

Every time a user u tags a document d with a tag t, this can be interpreted as an edge between u, d, and t. However, conventional graphs only allow for binary edges. One possibility to deal with this issue would be to store three edges (d, u), (u, t), and (d, t), which however comes at the price of information loss. To see this, consider four tagging events



(a) Graph G_s

(b) Hypergraph H_s, plotting hyperedges as nodes e

Figure 1.2.: A sample graph and a sample hypergraph. Note that edges appear as nodes in the hypergraph visualization

(d_1, u_1, t_1), (d_2, u_1, t_2), (d_2, u_2, t_1), and (d_1, u_2, t_2). By only storing pairwise information it would be no longer retrievable which user tagged d_1 with t_1 since both users have used both tags, both tags were used for d_1, and the ternary relationship, by definition, was not stored.

The question of how crucial exactly this information loss is turns out to be a recurring issue throughout the future chapters. However, let us here discuss the formal structure required to store the full information.

1.2.3.1. Definitions

Definition 1.4 (Hypergraph) *A* hypergraph $H = (V, E)$ *is a set* V *of* $|V| = N$ nodes *and a family* E *of* $|E| = M$ *subsets of* V *called* edges. *If* $\exists e \in E : v_i \in e \wedge v_j \in e$, *nodes* v_i *and* v_j *are* adjacent. *If* $\exists v : v \in e_1 \wedge v \in e_2$, *edges* e_1 *and* e_2 *are* incident.

Hypergraphs, first named so by Berge (1970), according to Chvátal (2003), are generalizations of graphs in which edges can connect arbitrary amounts of nodes. Estrada and Rodriguez-Velazquez (2005) give an overview of complex real-world networks that are best understood as hypergraphs, e.g. social networks in which relevant interactions take place between more than two actors, like buyer, seller and broker, or food webs in which all species competing for the same prey are connected through an edge. Zhou et al. (2006) generalize graph-based, semi-supervised machine learning techniques to hypergraphs. However, the class of hypergraphs is a very large one and these works discuss hypergraphs with an arbitrary number of edges. Here, we are concerned with a very specific subset of hypergraphs, since each edge connects exactly three elements from three distinct sets. These structures are called 3-uniform, 3-partite hypergraphs:

Definition 1.5 (k-Uniform, k-Partite Hypergraphs) *Let the* size *of an edge be the number of nodes it contains. The maximum size of an edge in* E *is called the* range *of* H. *If the size of all edges in* $E = r$, H *is a* r-uniform *hypergraph. A hypergraph is* k-partite *if* V *can be partitioned in* k *sets (from here on called* domains *instead of "partitions" to avoid confusion in the context of graph partitioning)* $V_1, \ldots, V_k$ *such that nodes from the same set are never adjacent. If* $k = r$, H *is a* k-partite, k-uniform, *or simply* k, k-hypergraph, *and its edges are* k*-tuples containing one element out of each set* V_i. *For brevity, let* $K = \{1, \ldots, k\}$ *and, where* k*=3, let* D, U *and* T *denote* H*'s 3 domains (documents, users and tags). The*

Table 1.1.: Incidence Matrices of the sample graphs

(a) $B(G_s)$

	$\mathbf{e_1}$	$\mathbf{e_2}$	$\mathbf{e_3}$
v_1	1		
v_2	1	1	
v_3		1	
v_4			1
v_5			1

(b) $B(H_s)$

	$\mathbf{e_1}$	$\mathbf{e_2}$	$\mathbf{e_3}$	$\mathbf{e_4}$
d_1	1	1		
d_2			1	1
u_1	1			
u_2		1	1	
u_3				1
t_1	1	1		
t_2			1	1

adjacency tensor A *of a* k*,* k*-hypergraph is a* $|V_1| \times \ldots \times |V_k|$*-dimensional tensor such that* $A_{v_1,\ldots,v_k} = |e \in E : e = (v_1, \ldots, v_k)|$.

"Normal" graphs are included in the set of hypergraphs as uniform hypergraphs with $r = 2$. A bipartite graph can be described as a 2,2-hypergraph. k, k-hypergraphs with $k > 2$ can thus best be conceived as generalizations of bipartite graphs. Imagine, e.g., a tag/document matrix representing connections between the two domains, inducing a bipartite graph. Introducing a third, say, user dimension turns this matrix into a tensor which induces a 3,3-hypergraph. This represents data in which documents and tags are no longer just either connected or not, but instead associated with each other by one or several users, as in social bookmarking data. More generally, such structures can of course represent any other type of data in which elements of three different kinds are in a ternary relation to each other.

Figure 1.2 proposes a visualization technique for hypergraphs that will be used throughout this thesis. Since edges with more than two vertices are notoriously hard to visualize, hypergraphs are turned into a bipartite graph by introducing "edge nodes" $e_1, \ldots, e_4$ and transforming a ternary edge like (d_1, u_1, t_1) into three binary edges $(d_1, e_1), (u_1, e_1), (t_1, e_1)$, allowing traditional visualization. To put it differently, what's plotted is the graph obtained by interpreting $B(H_s)$ as an adjacency matrix of a bipartite graph, with the two domains made up of the original nodes and the original edges, respectively. The sample graph G_s (for comparison) is defined by the incidence matrix in Table 1.1 a), and the sample 3-partite 3-uniform hypergraph H_s given by Table 1.1 b).

1.2.3.2. Social Bookmarking and Hypergraphs

It seems, in fact, that only in the context of social bookmarking k, k-hypergraphs have become the focus of major research interest. Hotho et al. (2006b), for example, propose a generalized version of PageRank, "FolkRank", computing tags or pages authorities using a model working on the native 3-dimensional structure. Cattuto et al. (2007b) have proposed generalizations of concepts like cliquishness or transitivity to 3,3-hypergraphs, drawing on

social bookmarking data for examples. Random generative models for 3,3-hypergraphs capturing characteristics of real social bookmarking datasets have been proposed by Ghoshal et al. (2009).

1.2.3.3. Community Detection in Hypergraphs

Closely related to the task at hand is work on hypergraph partitioning (Selvakkumaran and Karypis, 2006), which however does not address the specifics of k, k-uniform hypergraphs and is concerned with partitioning instead of community detection.

As k-dimensional tensors can be interpreted as adjacency tensors of k, k-hypergraphs, much work based on tensor decompositions (Kolda and Sun, 2008) is strongly related. These techniques have been applied to other r-uniform hypergraphs, e.g., on RDF-triples describing ontologies on the Semantic Web (Franz et al., 2009) or on datasets including time (Acar et al., 2005). Sun et al. (2005) propose a three-dimensional generalization of Singular Value Decomposition to improve web search using a threedimensional structure of users, queries, and clicked web pages. Many approaches to tag recommendation draw on tensor factorization methods as well, e.g. (Symeonidis et al., 2008; Rendle et al., 2009; Krestel et al., 2009; Wetzker et al., 2009).

Factorization methods try to find approximations of the original matrices or tensors by recreating these structures in a lower-dimensional space, mapping the original rows and columns to linear combinations of the new ones. These new elements might be interpreted as communities; however factorization methods are not optimized towards finding particularly sparse representations, i.e. clear community assignments. Cichocki et al. (2009) propose a threedimensional generalization of Non-Negative Matrix Factorization (NMF). NMF (Lee and Seung, 1999) is geared towards finding sparse mappings from original to reduced elements, which might make it a better choice. Another constraint however is that the dimensionality of the reduced structure, which would correspond to the number of communities, is typically not optimized during the process (as it is in community detection), but given as a hyper-parameter that is at best optimized in an external loop.

To my knowledge, the only article on community detection in partite hypergraphs that employs factorization methods is by Lin et al. (2009). All other works on community detection in three-dimensional structures work with a more graph-oriented techniques (i.e. expecting sparsely filled tensors) and are, interestingly, typically defined with respect to social bookmarking data directly. It is at this specific intersection of social bookmarking, community detection and hypergraphs that the work in this thesis is situated. The work in this particular area will be reported in the following section.

1.2.4. Community Detection in Hypergraphs for Social Bookmarking

An early approach in the spirit of multi-partite community detection was provided by Jäschke et al. (2008), whose "shared conceptualizations in folksonomies" correspond to simultaneous document, user, and tag communities. However, these assignments are not made for all nodes but only for particularly frequently co-occurring instances.

Closer to the work described here is the method by Lu et al. (2009), who propose a prototype-based tripartite clustering approach that creates individual clusters in all three partition based on edges between cluster members. It however solves a slightly different task since the number of clusters to be found in each dimension need to be provided as parameters. Nevertheless, this method's performance will be used for benchmarking purposes in Section 4.2.

Most closely connected to this thesis is the tripartite modularity described by Murata (2010a), which is based on my earlier work in that field, whereas that earlier work was in turn inspired by the bipartite modularity formulated by Murata (2009). A quantitative comparison between these methods will be provided in Section .

Concludingly, there is little work on community detection suited for application on bookmarking data, and most of the work has been created in parallel with the work presented here.

1.2.5. Summary

This section is intended to both motivate community detection in hypergraphs and show the need for additional work in this area. The basic argument can be outlined as follows:

- Social bookmarking data is presumably full of latent knowledge and intricate structure.
- Community detection is a promising approach to unveil the bigger picture in complex networks.
- 3-partite, 3-uniform hypergraphs are the appropriate choice for representing social bookmarking data.
- Community detection in 3-partite, 3-uniform hypergraphs requires special methods.

More generally, the task described here is the simultaneous discovery of communities along different domains (multi-partite community detection) in sparse regimes, i.e. for adjacency tensors which suggest a graph-like interpretation of the data. While social bookmarking data was the original motivation as well as the focus of a large portion of previous work dealing with these particular data structures, this is in fact a more general theoretical setting. It is in this sense of generality that I try to refrain from using content-based features from the individual domains like URL patterns, user names, or tag semantics, and instead try to formulate content-agnostic methods that focus on structural features (an exception is the work presented in Chapter 2, which is the earliest work described here and shows the evolution from content-based to purely structural methods). Even the fixed values of $k = 3$, in my opinion, should only be a setting for a particular problem domain. To stress this point, more general statements about k, k-hypergraphs (instead of just 3,3) are made wherever this can be achieved without exaggerated notational efforts.

1.3. Datasets

Most of the datasets examined in this thesis occur across different chapters. Let me therefore list these datasets at this early occasion. This has the additional advantage of providing a more concrete view on the data discussed above.

1.3.1. Delicious

This corpus contains a month (December 2007) worth of entries from `delicious.com`, taken from the corpus described in (Wetzker et al., 2008). No explicit spam information is given in this dataset, but the authors report to have manually identified a certain amount of spamming activity.

1.3.2. Bibsonomy

Many analyses were performed on the `bibsonomy.org` dataset (Knowledge & Data Engineering Group, University of Kassel, 2008) as provided to the participants of the PKDD/ECML 2008 Tag Spam Discovery Challenge. Bibsonomy is a social bookmarking platform for both bibliographic references and URLs offered by the Hotho et al. (2008). The dataset contains manually assigned labels classifying users as spammers or legitimate users.

1.3.3. CiteULike

`citeulike.org` is a social bookmarking system for bibliographic references only. The dataset contains all entries up to February 22nd, 2008, obtained from `citeulike.org/faq/data.adp`. Not too much spamming activity is to be expected here because the only annotated entries are academic articles.

1.3.4. Visualize.us

Visualize.us is a social bookmarking site for images. The crucial difference to photo sharing services like Flickr is that the bookmarked documents are not photos uploaded personally, but rather images found somewhere on the web. In systems which include a sense of ownership, tagging rights are often limited to owners or their friends. With Visualize.us, every user can tag every document, leading to a potentially higher number of tags and tagging users per document. As mentioned in the introduction, social bookmarking datasets with this property are also called "broad folksonomies" – Flickr on the other hand is the prototypical example for "narrow folksonomies" used in Vander Wal (2005)'s original definition of these terms. The data was obtained through a crawler written by Andreas König.

1.3.5. MovieLens

MovieLens (`movielens.org`) is a collaborative movie recommendation service provided by the GroupLens research group of the University of Minnesota. The data collected through this service is available in several datasets under `grouplens.org/node/73`.

Table 1.2.: Examined networks and the number of vertices (by type) and edges.

name	$\lvert\mathbf{D}\rvert$	$\lvert\mathbf{U}\rvert$	$\lvert\mathbf{T}\rvert$	$\lvert\mathbf{E}\rvert$	$\frac{\lvert\mathbf{D}\rvert}{\lvert\mathbf{E}\rvert}$	$\frac{\lvert\mathbf{U}\rvert}{\lvert\mathbf{E}\rvert}$	$\frac{\lvert\mathbf{T}\rvert}{\lvert\mathbf{E}\rvert}$
Delicious	3,536,030	283,414	727,400	19,599,270	18.0%	1.4%	3.7%
Bibsonomy	1,574,963	38,920	396,474	16,818,699	9.0%	0.2%	2.4%
without spam	294,196	2,638	72,776	947,306	31.1%	0.2%	7.7%
CiteULike	734,442	22,814	153,688	2,419,430	30.4%	0.9%	6.3%
Visualize.us	518,701	17,153	82,152	2,298,816	22.6%	0.7%	3.6%
MovieLens	10,681	4,009	15,240	95,580	11.1%	4.2%	15.9%
AOL	1,607,028	520,956	1,071,618	52,094,425	3.1%	1.0%	2.1%

The data used here is part of the "Movielens 10M" daatset, which contains, in addition to 10 million ratings of 10,000 movies by 72,000 users, around 100,000 tag assignments.

1.3.6. AOL

In order to contrast social bookmarking data with other data of the same format, the AOL search log dataset described by Pass et al. (2006) was examined. Each combination of searcher, search term and clicked document is interpreted as an edge.

1.3.7. Summary

Table 1.2 shows the basic statistics of the introduced datasets. While the datasets differ in absolute size as well as in the relative sizes of the different domains, some commonalities can be found. For example, it holds that $|D| > |T| > |U|$ in all datasets. The AOL search log dataset differs from the bookmarking datasets by having the lowest number of documents and tags in relation to the number of edges, implying that there might be less variety in search terms used and documents selected during web search than there is in applied tags and tagged documents during social bookmarking. More generally, this is a first, general hint that the process underlying the creation of these datasets becomes visible through basic statistical properties.

1.4. List of Symbols

Table 1.3.: Table of Symbols

symbol	short explanation	definition
$G = (V, E)$	a graph with nodes V and edges E	Def. 1.1, p 7
$H = (V, E)$	a hypergraph with nodes V and edges E	Def. 1.4, p. 11
A	adjacency tensor of a hypergraph	Def. 1.4, p. 11
k	the number of domains in a k, k-hypergraph	Def. 1.5, p. 11
(d, u, t)	an edge from a 3,3-hypergraph	Def. 1.5, p. 11
P	expected adjacency between node or communities	Def. 1.2, p. 8
M	the number of edges in a (hyper)graph	Def. 1.2, p. 8
Q	modularity	Def. 1.2, p. 8
δ	Kronecker's delta function, 1 for identity	Def. 1.2, p. 8
σ	a community assignment function, mapping nodes on communities	Def. 1.2, p. 8
$\boldsymbol{\sigma}$	a k-partite community assignment (k different σ functions)	Def. 3.1, p. 50
$C(\boldsymbol{\sigma})$	the community space induced by $\boldsymbol{\sigma}$	Def. 3.1, p. 50
$V_{d,i}(\boldsymbol{\sigma})$	nodes of domain d in community i given by $\boldsymbol{\sigma}$	Def. 3.1, p. 50
$\boldsymbol{\sigma}^\tau$	the τth state in a k-partite hierarchical clustering	Def. 3.2, p. 51
φ	a single join: domain, source, and target community to be joint	Def. 3.2, p. 51
Σ	a k-partite hierarchical clustering, a list of joins	Def. 3.2, p. 51
$\boldsymbol{\sigma} \circ \varphi$	the application of a join on a previous community assignment	Def. 3.2, p. 51
$\mathbf{c}_{\boldsymbol{\sigma}}$	a community edge induced by $\boldsymbol{\sigma}$	Def. 3.3, p. 51
$\mathrm{vol}(\mathbf{c}_{\boldsymbol{\sigma}})$	the volume of $\mathbf{c}_{\boldsymbol{\sigma}}$	Def. 3.4, p. 52
$\rho(\mathbf{c}_{\boldsymbol{\sigma}})$	the density of $\mathbf{c}_{\boldsymbol{\sigma}}$	Def. 3.4, p. 52
$r(\mathbf{c}_{\boldsymbol{\sigma}}, i)$	the relative importance $\mathbf{c}_{\boldsymbol{\sigma}}$ for community i	Def. 3.4, p. 52
$C_{\boldsymbol{\sigma},\alpha}$	all community edges of relative importance α	Def. 3.4, p. 52
$e_l m$	normalized number of edges between two communities l and m	Def. 4.1, p. 69
a_l	normalized number of edges in community l	Def. 4.1, p. 69
Q_{M}	Murata's modularity	Def. 4.1, p. 69
$E_{\mathrm{proj}}(v_i, v_j)$	the set of edges connecting nodes v_i and v_j	Def. 4.2, p. 70
$E_{i,j}$	the bipartite projection of a hypergraph w.r.t. its ith and jth domain	Def. 4.2, p. 70
Q_{CB}	coupled bipartite modularity	Def. 4.3, p. 70
$GQ(\boldsymbol{\sigma}, \varphi)$	the gain in modularity obtained by applying φ to σ	Def. 4.4, p. 71
$f_d(\boldsymbol{\sigma})$	correspondence function: the importance of $\boldsymbol{\sigma}$ for its dth community	Def. 4.5, p. 79
Q_{MP}	multi-partite modularity	Def. 4.5, p. 79
$Q_{\mathrm{MP}\vert\ldots}$	multi-partite modularity with a particular correspondence function	Def. 4.5, p. 79
Q_β	mixed multi-partite modularity	Def. 4.6, p. 93
$Q_{\mathrm{MP},\alpha}$	dampened multi-partite modularity	Def. 4.8, p. 102
$Q_{\mathrm{MPbal}\vert\mathrm{lin}}$	balanced multi-partite modularity	Def. 4.9, p. 106

1.5. Outline

This rest of this document is structured as follows:

Chapter 2: Hyperincident Connected Components introduces a generalization of connected components to k, k-hypergraphs. It demonstrates that the hyperincident connected components of social bookmarking datasets follow a typical distribution, which is disrupted by, e.g., spamming activity.

Chapter 3: Multi-Partite Community Detection introduces the general topic of community detection in k, k-hypergraphs and provides three families of synthetic benchmarking graphs.

Chapter 4: Multi-Partite Modularity provides a set of modularity measures suitable for application on k, k-hypergraphs. Starting from the most simple approach possible, the method is successively refined to meet the various challenges posed in Chapter 3.

Chapter 5: Community Detection on Real Data introduces an interactive tool for exploring obtained community assignments and illustrates the advantages of different community detection approaches using real-life datasets.

Chapter 6: Conclusions closes the main part of this document.

Appendix A.1 contain details about the software packages provided for the main findings of the core chapters 2 to 5.

Appendix A.2 contains additional figures.

2. Hyperincident Connected Components

My work on social bookmarking data started by participating in the spam detection task taking place as part of the 2008 ECML/PKDD Discovery Challenge (Hotho et al., 2008). The dataset described in Section 1.3.2 was provided to researchers, with users manually classified into spammers and legitimate users. The task was to identify the characteristics of spamming behaviour and train a classifier to predict new users' status in a previously unknown test dataset.

The most relevant result of this work, in my view, is the subsequent definition of hyperincident connected components, a way of decomposing hypergraphs purely based on structural features that singles out spammers rather successfully, in particular given that it works completely unsupervised. The finding that structural features can provide such strong signals about the contents of social bookmarking data will be crucial for the further course this thesis, leading to the work on community detection. Furthermore, connected components can be interpreted as an extreme form of communities, where two nodes live in the same community as long as a path exists between them. Therefore, even though this chapter does not deal with community detection in the strictest sense, it plays – apart from the stand-alone value of the contained results – an important role to highlight the motivations behind the work presented later on.

Section 2.1: Spam Detection describes several measures for predicting a users' probability of being a spammer and discusses their performance both when used individually and when combined into a single predictor.

Section 2.2: Hyperincident Connected Components introduces the notion of hyperincident connected components and its applicability to spam detection.

The work focussing on spam detection has been published as (Neubauer and Obermayer, 2008), the parts focussing on hyperincident connected components as (Neubauer et al., 2009; Neubauer and Obermayer, 2009a).

A database-backed version of the algorithm described in this chapter for decomposing hypergraphs into their hyperincident connected components is provided as a downloadable software package, *hcc* – see Appendix A.1.1 for details.

2.1. Spam Detection

2.1.1. Background

Distinguishing artificial bookmarking/tagging behaviour (such as spamming) from genuine human information management activities can help sharpen our understanding of the underlying cognitive and social processes – it has, however, also become a practical task. Social bookmarking sites now receive significant attention, creating incentives for spammers to penetrate these systems. When targeting users, spammers tag fake sites with popular tags, trying to trick users into visiting the posted site when they browse the entries for a given tag. Search engines can be targeted by tagging the promoted website with a random tag: Social bookmarking sites show a list of top entries for each tag, and for a page to be in the top (and probably only) position in such a list for any tag might lead search engines to boost that page's ranking. Even though it unclear whether search engines actually use that information, the possibility alone seems to have created sufficient incentive for web spammers to populate social bookmarking systems: The motivation for the spam detection challenge stems from operating a real social bookmarking system, which turns out to involve dealing with huge amounts of spam.

2.1.2. Related Work

Heymann et al. (2007); Koutrika et al. (2007) have simulated the impact of several spamming practices on the overall properties of a social bookmarking dataset in dependence of a number of key parameters. In (Cattuto et al., 2007b), spam is mentioned briefly as it causes a deviation from an otherwise smooth strength distribution of a tag network. A number of indicators for suspicious activity was proposed by Wetzker et al. (2008) based on an analysis of a vast corpus of tagging data obtained from Delicious. Most related to the current task is however the work of Krause et al. (2008), in which the organizers of the workshop describe experiments on an earlier version of the dataset used here. The following contributions follow a similar approach by creating user features to train a classifier on. However, I will explore other features, varying the exploitation of co-occurrence patterns already used in (Krause et al., 2008), but also introducing new features based on network analysis, and perform a text classification on the URL components of the bookmarks.

2.1.3. The Challenge Dataset

The training dataset provides 14,074,956 triples $E = (d, u, t) \in D \times U \times T$, where D is the set of 1,425,108 documents, U is the set of 31,715 users, and T is the set of 310,234 tags. Each of these triples represents a single association a user of Bibsonomy has created between a tag and a document. Documents can either be bibliographic references or WWW bookmarks and come with associated metadata like URL, title etc. Users are simply presented as IDs and labeled as spammers or non-spammers. Tags are strings.

The most striking fact is that 29,248 of 31,715 users are spammers, i.e., only 7.78% are legitimate users. Most tags and documents are used by spammers or non-spammers exclusively and can thus be regarded as spam (84% of documents, 15.7% of tags), or non-spam

(15.7% of documents, 11% of tags) as well. Overlap between use by spammers and non-spammers is very rare in documents (0.3%), but more frequent among tags (7%). This indicates two different incentives for spammers: They may post spam bookmarks under frequently used tags, such that other users browsing the repository are led to their pages. Posting bookmarks under other, sometimes randomly created tags, probably serves the purpose of creating links from reputated sites (the bookmarking site) to the spammed page, trying to trick search engines into improving the spammed page's rank.

The merged training and test dataset consists of 16,818,699 triples, 1,574,963 documents, 38,920 users and 396,474 tags. Of the new 7,205 users in the test set (the prediction targets), only 171, i.e. 2.37% are regular users. Furthermore, it is notable that around 90% of all documents are only bookmarked by one person.

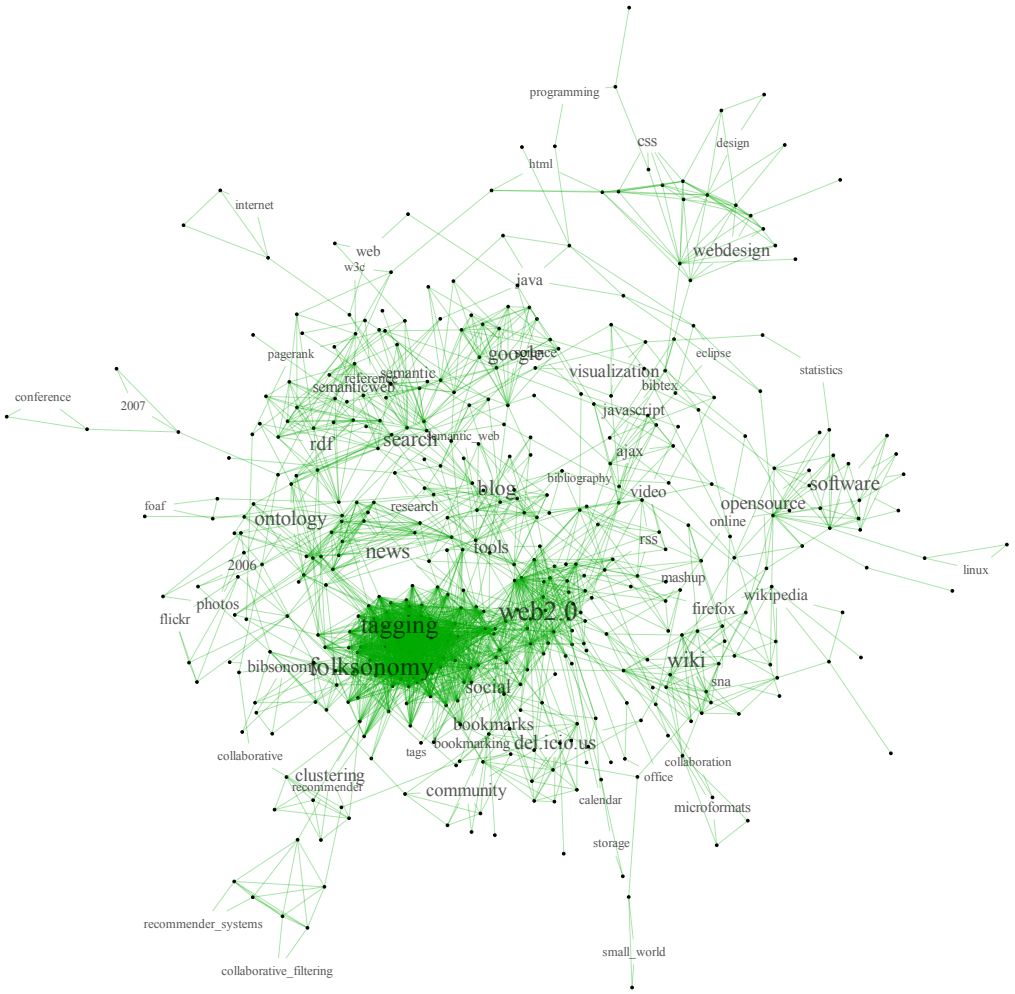
2.1.4. Tag/Document Clouds

On this occasion, let me briefly introduce a visualization method which extracts a large-scale view of the contents of a given social bookmarking dataset. It is inspired by Stefaner (2007), who positions tags by their mutual relations, extending the normal tag cloud by using spatial position to indicate semantic similarity. However, his visualization focusses on a single user's set of tags, whereas the method described here provides a more global view.

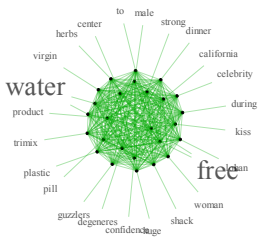
The visualization focusses on the n most frequently bookmarked documents. For each of these documents, a "tag vector" is constructed in a similar fashion to the term vectors frequently used in Information Retrieval: Each tag is assigned to a unique index in a vector whose size equals the number of unique tags used by all documents. Each document can then be represented as a vector whose ith element contains the number of times users have associated it with the corresponding tag t_i. The cosine similarity assigns a value between 0 (if the vectors are orthogonal, i.e. the documents do not share any tags) and 1 (if the two tag vectors are identical) and is given, for two vectors A and B, by $\frac{A \cdot B}{||A|| ||B||}$, i.e. the normalized inner product.

Having thus obtained pairwise similarities between all documents, the m most similar pairs are connected. Additionally each document is connected to its most frequently assigned tag, and tag labels are logarithmically scaled by the number of associated documents. The resulting graph is rendered using the GraphViz (Gansner and North, 2000) visualization toolkit. All tag/document clouds depicted here were constructed using n=1000 and m=3000, which seems to provide a good tradeoff between complexity and readability.

Figure 2.1 visualizes the top entries of the Bibsonomy social bookmarking dataset, first without spam, then with spam included (visualizations of the other datasets can be found in Appendix A.2.1). We can see very complex and subtle patterns in the clean data, which are overshadowed by the spam entries in the second plot. These two figures not only provide an example of spam in social bookmarking systems, but also motivate the basic assumption underlying the further work: Spammers appear to not only post different websites with different tags, but they behave differently in such a fundamental way that it structurally changes the resulting networks.



(a) spam-free



(b) with spam

Figure 2.1.: Tag/Document Cloud of the Bibsonomy dataset

2.1.5. Spam Measures

Users are classified using three different types of features, based on co-occurences, network properties and URL terms. Co-occurrence features are based on the assumption that users associated with similar documents and tags as spammers are likely to be spammers themselves. Network-based features work on a collective scale, assuming common behavioural patterns which can be identified in the graph structures created by tagging activities. Finally, a text classification on the URLs' components identifies frequent terms in spam URLs. With these features, an SVM is trained for classification. Here, the features are introduced in more detail, whereas the next section will describe the training and prediction process in more detail.

2.1.5.1. Co-Occurrence Features

Distributing "Spamminess" Let a tag's or a document's *spamminess* be the frequency by which users that use that tag / tag that document are classified as spammers. I formalize these notions for documents; they are equivalently definied for tags. Let $U_+(d)$, $U_-(d)$ and $U_?(d)$ be the set of spam, non-spam and unknown users who tagged a document. Then we can define a document's spamminess $s(d)$ as well as a confidence $c(d)$ for that measure, based on the fraction of labelled vs unlabelled users, as

$$s(d) = \frac{|U_+(d)|}{|U_+(d)| + |U_-(d)|}, \quad c(d) = \frac{|U_+(d)| + |U_-(d)|}{|U_+(d)| + |U_-(d)| + |U_?(d)|}.$$

If there are no known users for a given element, the confidence is set to 0, and spamminess to the average of all documents (around 0.8).

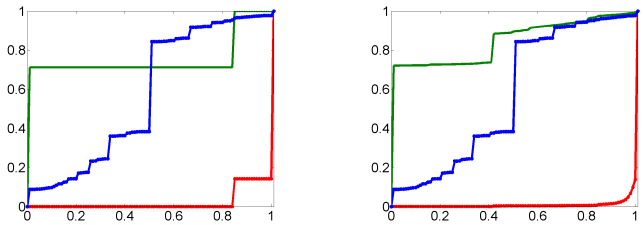
See Figure 2.2(a) for the distribution of the resulting values. This figure was generated using the training dataset while holding out each fifth user's label to create a setting similar to the test situation. Non-spam documents have either a value of 0 or, if unknown (due to the held-out values), the assigned average value, whereas spam documents have a value of either this average if unknown or otherwise 1.

From the documents' spaminess ratings, an unknown user's spamminess can be computed as

$$s(u) = \frac{\sum_{d \in D(u)} s(d)c(d)}{\sum_{d \in D(u)} c(d)}, \text{where } D(u) = \{d \in D : \exists t \in T : (d, u, t) \in E\},$$

i.e. the average spamminess of all documents the user has tagged, weighted by the confidence.

Iterated Spamminess Distribution We see in Figure 2.2(a) that a significant part of both spam and non-spam documents receive the default, average value (implying confidence 0) and can thus not be used to classify users. This is due to the high fraction of documents tagged by only one user – if those single users are unlabelled, the corresponding documents cannot be used for spam prediction. However, even if we cannot tell a document's spamminess by the users that have tagged it, we might learn something from the tags it was tagged



(a) simple distribution (b) iterated distribution

Figure 2.2.: Accumulative document spamminess after spamminess distribution for spam(red), non-spam(green) and mixed(blue) documents

with. In the same manner, we can estimate a tag's spamminess by the documents it was used for. In order to use this information, a second feature is computed for each element, its iterated spamminess s_i and the according confidence c_i. We initialize s_i and c_i with the original values s and c, and then compute

$$c_i(d) = \frac{\sum_{t \in T(d)} c_i(t)}{|T(d)|}, \text{where } T(d) = \{t \in T : \exists u \in U : (d, u, t) \in E\}$$

and

$$s_i(d) = \frac{\sum_{t \in T(d)} : s_i(t) c_i(t)}{c_i(d)}.$$

In short, documents whose spamminess cannot be predicted obtain the weighted average spamminess of their associated tags, and their average confidence. This process is repeated iteratively for all tags and documents with confidence 0 until no such elements remain or no further information can be spread. Figure 2.2(b) shows the smoothing effect of this transformation: Almost all spam documents now receive a weighting of close to 1, whereas the prediction for the previously unpredictable non-spam documents could be reduced from 0.8 to 0.4, on average.

Cosine Similarity As introduced in (Krause et al., 2008), a simple way to exploit co-occurence information is to create a similarity function between users based on their used tags, and predict a user's spamminess as the sum of the known users' spamminess weighted by that similarity. I integrated that approach by creating tag (and document) vectors for each user such that each component corresponds to a given tag, and the value of that component would be 1 if the user used that tag. Then, the cosine between the two tag vectors serves as a similarity function.

2.1.5.2. Network Features

Next, network features are turned into user features. To examine only relevant portions of the overall graph structure, I define *induced graphs* as the bipartite graphs gained by fixing an element from one of the three sets, and connecting those elements from the other two

sets that are connected via the fixed element. For example, we might fix a document and then examine all users and tags associated with it, with edges connecting users with the tags they used for the document. More formally, induced graphs are defined by their edges E_D, E_U and E_T obtained by fixing a particular document, user, or tag as

$$E_D(d') = \{(u,t) \in U \times T \ : \ (d',u,t) \in E\}$$

$$E_U(u') = \{(d,t) \in D \times T \ : \ (d,u',t) \in E\}$$

$$E_T(t') = \{(d,u) \in D \times U \ : \ (d,u,t') \in E\}$$

Then, the following measure are computed for all of these graphs:

Connected components A connected component of a graph G is a set of nodes such that any node can be reached from any other node in that component by travelling along the edges in G (this will be introduced more formally in the next section, where this definition is generalized – see Definition 2.1). If a graph is disjoint, the number of connected components describes the number of disconnected subgraphs – the graph in Figure 1.2(a) is an example of a graph with two connected components. The number of components found is directly used as a measure.

Characteristic Path Length The characteristic path length is the average shortest path distance between two arbitrary nodes in the graph. This number was computed for all graphs in which no element set exceeded 1000 elements.

Degree Ratios Finally, the ratio between the average degrees of each element set was taken into account, normalized by the size of the other set. For a user-induced graph, this would mean

$$d_{d,t}(u) = \frac{\text{avg degree}(D(u))}{|T(u)|} \cdot \left(\frac{\text{avg degree}(T(u))}{|D(u)|} \right)^{-1},$$

where avg degree(S) is the average degree of all elements in S. It is only after these normalizations and the division that relevant differences between spam and non-spam users appear.

2.1.5.3. URL Classification

A central aspect in deciding whether a given bookmark is spam or not should be its content. While the bookmarked URLs cannot simply all be downloaded, their content can be estimated by analyzing the terms used in the URLs themselves. Each URL is split by dashes, dots, colons, and frequent elements like “http”, “www” or “html” are removed, creating a feature vector containing a tfidf representation of its terms. Then, the feature vectors of all URLs with spamminess 0 are used as negative samples, and an equal number of URLs with spamminess 1 as positive samples. After training a linear SVM (Fan et al., 2008), unknown URLs are classified with a value ($\text{urlspam}(d)$) between -1 and 1, and all URLs with a

value between 0 or 1 are considered spam, skipping those URLs which do not contain any known URL part. This yields a new user feature

$$\mathrm{url}(u) = \frac{\sum_{d \in D_{\mathrm{url}}(u)} \mathrm{urlspam}(d)}{|D_{\mathrm{url}}(u)|},$$
$$D_{\mathrm{url}}(u) = \{d \in D : (\exists t : (d, u, t) \in E) \wedge \mathrm{urlspam}(d) \text{ is defined}\}.$$

2.1.5.4. Other Features

Connection Strength For each tag/document pair, the number of users that used it together were counted. It turns out that high values tend to imply spamminess. Therefore, two user features are included measuring the averages of each document's a) average and b) maximum connection strength .

Counting Finally, the average number of user per document (again averaged to the single user), the average ratio of tags and users per document, and simply the number of entries per user are integrated as features.

2.1.6. Detection Approach

2.1.6.1. Creating a validation dataset

Trying to simulate the setting of predicting the unknown test dataset, a validation dataset was created from the training dataset. Users, ordered by date of registration, are split into five subsets, each containing the nth fifth of spam and the nth fifth of non-spam users. Training on the first four fifths and evaluating on the last one, the situation of predicting future users knowing roughly four times as many from the past is simulated, as it is the case in the actual test scenario.

2.1.6.2. Training a classifier

With the a given features, a Support Vector Machine is trained using SVMLight (Joachims, 1999). The parameters are obtained by optimizing for the previously created validation dataset while training on the first four fifths of the data. Across various situations, a polynomial kernel of degree 6 with a balancing factor of 0.077 (the fraction of non-spammers in the training dataset) to be best. With these parameters, a classifier is trained on the whole training dataset (i.e. all fifths), to be applied on the test dataset.

2.1.7. Results

Algorithms are evaluated using the AUC measure, the area under the ROC (Receiver Operating Curve). This measure accepts a ranked list and expresses the probability that a randomly chosen positive sample will be ranked higher than a randomly chosen negative one (Fawcett, 2006), thereby assigning 1 to a perfect separation and 0 to a ranking in which all positive samples are predicted to be more negative than any negative one. Let us now first review the performance of the individual features and then evaluate the compound classifier.

2.1.7.1. Single Features

A detailed list of the features generated from each group is presented in Table 2.1. It documents the AUC values for each feature when used alone as predictor, both on the probe and the test dataset.

Co-Occurrence Features The average spamminess of used elements, particularly of tags, is a strong predictor of a user's spamminess. Iterating spamminess distribution helps to increase the expressiveness of document spamminess. In fact, the product of tag spamminess and iterated document spamminess performs better than the final predictor on the test set.

URL prediction Apart from co-occurrence features, the URL predictor is the strongest single predictor of spamminess.

Network Features Many of the features describing the statistical properties of induced graphs seem to indicate a tendency towards spamminess or non-spamminess, but no single feature is useful as a stand-alone predictor. The degree ratio between documents and tags in user-induced graphs, and the number of connected components in tag-induced graphs seem to be the most relevant single predictors.

Other features The ratio of tags per user, for documents, turns out to be a useful measure. Also, the simple number of entries per user provides a tendency towards spamminess.

2.1.7.2. Compound Results

The SVM-based approach described above achieved an AUC of 0.913 on the test set for the submission run, using basically all the features introduced earlier (see column "S" in Table 2.1). In hindsight, it turns out that leaving out network features entirely (see column "O") yielded the best result of 0.961. The performance of network-based features alone (column "N" in the overview) lies at an AUC of 0.854. Finally, the performance of using, without an additional classifier, the product of the tag spamminess and the iterated document spamminess amounts to 0.929.

2.1.8. Discussion

The most striking result of the presented experiments is that, at the end of the day, the constructed classifier performs worse than a simple product of two of the features used, tag and iterated document spamminess. What happened? As it turns out, the combined spamminess features perform a lot worse (almost 0.1 AUC) on the validation set than on the test set. The trained classifier weighs the features accordingly and is thus not able to benefit from the improved spamminess features on the test set. The quality of the network measures remains stable from probe to test set, and so does the submitted classifier. Leaving out network properties during training (classifier "optimal") forces the classifier to weigh co-occurrence features more strongly, and thus the increased quality of those features can be put to use. To conclude, the unexpected behaviour of the classifier has more to do with

Table 2.1.: Features by group, and AUC value if used as single predictor

Feature name	AUC(val.)	AUC(test)	Used by S	N	O
Co-Occurence Features					
Avg. Spamminess					
Documents	0.676	0.689	s		o
Tags	0.839	0.926	s		o
Avg. Spamminess (iterated)					
Documents	0.813	0.900	s		o
Tags	0.842	0.918	s		o
User Cosine Similarity					
Documents	0.554	0.533	s		o
Tags	0.823	0.887	s		o
Avg. Document * Avg. Tag Spamminess					
Normal(tags) * Iterated(documents)	0.843	0.929			
Features of Induced Graphs					
Degree ratios					
Avg (deg(user)/deg(document)) per tag	0.518	0.565	s	n	
Avg (deg(user)/deg(tag)) per document	0.669	0.660	s	n	
deg(docs)/deg(tag)	0.683	0.674	s	n	
Characteristic Path Length					
User	0.607	0.579	s	n	
Avg. per tag	0.375	0.358	s	n	
Avg. per document	0.659	0.658	s	n	
Connected components					
#connected components/#entries by user	0.328	0.350	s	n	
Avg. #connected components per tag	0.681	0.704	s	n	
Other Features					
Connection strength					
Avg (Avg. connection strength per document)	0.560	0.576	s	n	
Avg (Max. connection strength per document)	0.549	0.559	s	n	
URL classification					
Avg. URL prediction per document	0.814	0.787			o
Counting features					
Avg. #users per document	0.503	0.515	s	n	
Avg. #tags/#users per document	0.674	0.660	s	n	
# entries	0.642	0.627	s	n	o

the discrepancy between the final test set and the final fifth I chose as a validation set, than with the features themselves.

The best submission (Gkanogiannis and Kalamboukis, 2008) reached an AUC score of 0.98, focussing on optimizing text classification on URL components as was part of the described classifier. Still, the network features have to be regarded as promising, as they produce insight into the structural properties of spamming behaviours, and in contrast to co-occurrence or URL features, no labeled users are needed for their construction. This could prove valuable when examining new datasets for which no labels have been created yet. Finally, a content-based approach like the text classification may not be available or feasible for situations where other objects than URLs are involved. Therefore, I have further examined structural properties after the completion of the actual challenge, leading to results described in the next section.

2.2. Hyperincident Connected Components

Decomposing a graph into its connected components, as briefly discussed already, is a fundamental diagnostic procedure: The number of components and their relative size, in particular that of the largest one (known as the potential "giant component") can yield valuable insights into the basic structure of a graph and the processes underlying its formation. Decomposing tagging networks into their connected components however turns out to be uninformative: As will be shown later on, they tend to consist of a single connected component containing more than 99,9% of all nodes. In this section, I therefore propose a generalization of the notion of connected components to hypergraphs – hyperincident connected components – that raises the requirements for being connected to a level that creates more meaningful component distributions for the networks we are dealing with here.

This section starts by defining connected components and their hyperincident generalization. Only with these definitions in place, the related literature can be meaningfully discussed. Following that, an efficient construction algorithm is proposed, and then the results of applying both measures on the various tagging datasets are discussed. Building on these main results, two additional topics are discussed: The relation of the found distributions to components in bipartite projections of the original graphs, and the application of the found measures for supervised spam detection.

2.2.1. Definitions

2.2.1.1. Connected Components

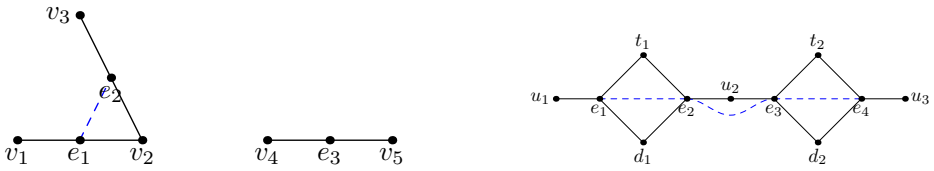
The connected components of a graph define its disjunct subgraphs, i.e. a partition of its nodes such that a path exists between any pair of nodes within a component, but no path exists between any two nodes from two different components.

Definition 2.1 (Connected Components) *Let $H = (V, E)$ be an arbitrary hypergraph. A* path *exists between two nodes v_1 and $v_2 \in V$ iff either v_1 and v_2 are adajacent, or $\exists v_x$ such that v_1 and v_x are adjacent, and a path exists between v_x and v_2. Let $CC(H)$, the* connected components *of H, be a unique partition of V such that there exists a path between any two nodes in each partition, but no path between any two nodes from two different partitions.*

This definition makes a point of using an arbitrary *hyper*graph even though connected components are usually defined on graphs: The regular definition does not refer the size of the involved edges and can therefore be directly applied to the more general concept of hypergraphs. Revisiting the sample graphs G_s and H_s from Section 1.2.3, G_s is an example of a graph composed of two connected components, H_s is an example of a hypergraph consisting of a single connected component (see Figure 2.3).

2.2.1.2. Edge-Based Formulation of Connected Components

As we have just seen, the definition of connected components is agnostic of the size of edges – either two vertices are adjacent via at least one edge, or they are not. However,



(a) G_s with two connected components (b) H_s with one component

Figure 2.3.: Sample graphs. Blue, slashed connections indicate incident edges.

this definition takes a node-centric view on connected components. We can instead regard connected components from the perspective of the edges involved.

Definition 2.2 (Edge-Path) *An* edge-path *exists between two edges e_1 and e_2 iff either e_1 and e_2 are incident, or $\exists e_x$ such that e_1 and e_x are incident, and an edge-path exists between e_x and e_2.*

Figure 2.3 shows the incident edges in G_s and H_s, respectively. The notion of an edge-path in H is equivalent to the notion of a regular, vertex-based path in the line graph $L(H)$, which is defined as the graph connecting incident edges in H.

Definition 2.3 (Connected Edge-Components) *Let two edges e_1, e_2 belong to the same* connected edge-component *if there exists an edge-path between e_1 and e_2. Let $eCC(H)$, the* connected edge-components *of H, be a unique partition of E such that there exists an edge-path between any two edges in each partition, but none between any two edges from different partitions.*

Again, an alternative definition of $eCC(H)$ is $CC(L(H))$, where $L(H)$ is the line graph of H. Later on, it will however be notationally more convenient to keep considering H instead of $L(H)$, so $L(H)$ is constructed only implicitly.

$eCC(H)$ is a set of connected components of edges. This can be used to retrieve a partition of vertices analogous to $CC(H)$:

Definition 2.4 (Conn. Components (edge-based)) *$CC'(H)$ describes a unique partition of V such that any two vertices v_1 and v_2 belong to the same partition if $\exists e_1, e_2 \in E : v_1 \in e_1 \wedge v_2 \in e_2$ and e_1 and e_2 belong to the same connected edge-component. Any isolated vertex $v : \nexists e \in E : v \in e$ forms an additional, distinct component.*

The edge-based definition of connected components induces the same partition on the set of all nodes as the traditional, node-based one:

Lemma 2.1 *$CC(H)$ and $CC'(H)$ describe an identical partition of V, i.e. two vertices v_1, v_2 belong to the same component c in $CC(H)$ iff they belong to the same component c' in $CC'(H)$.*

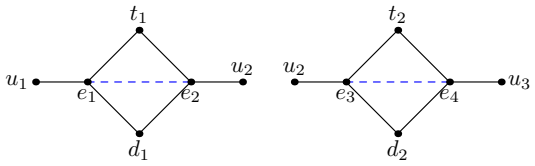


Figure 2.4.: The two 2-incident connected components in H_s. Blue, slashed connection indicate 2-incidence between edges e_1 and e_2 / e_3 and e_4, respectively. e_2 and e_3 are incident via u_2, but not 2-incident – sharing neither a tag nor a document – turning a single connected component into two 2-incident connected components.

Proof 2.1 *If v_1 and v_2 are connected according to $CC(H)$, a path must exist between them. This means that there must be edges e_1 and e_2 such that $v_1 \in e_1$ and $v_2 \in e_2$ and an edge-path exists between e_1 and e_2, which is the definition of belonging to the same component according to $CC'(H)$.*

2.2.1.3. Hyperincident Connected Components

With connected components defined in terms of edges, they can be generalized to "hyperincident" connected components.

Definition 2.5 (m-Incidence) *Let two edges e_1, e_2 be* m-incident *if $|e_1 \cap e_2| \geq m$.*

This notion evolves around the fact that in a hypergraph, incident edges may share either one vertex (as in a normal graph) or more than one (since they may have more than two vertices). This creates a graded form of connectivity between edges, where a hypergraph of range r can have edges that are up to $m = r - 1$-incident (for two edges to be exactly r-incident would mean that all vertices are shared, which would imply the two edges are in fact identical). We can now identify paths between edges that are m-incident:

Definition 2.6 (m-Incident Edge-Paths) *An* m-incident edge-path *exists between two edges e_1 and e_2 if either e_1 and e_2 are m-incident, or $\exists e_x$ such that e_1 and e_x are m-incident and an m-incident path exists between e_x and e_2.*

In parallel to the above construction of edge-based connected components, these paths can be used to construct components of edges:

Definition 2.7 (m-Incident Edge-CCs) *Let the* m-incident connected edge-components *of H,*

$$eHCC(H, m) = \{ec_1, \ldots, ec_n\}$$

be a unique partition of E into n disjoint sets ec_i such that there exists an m-incident connected edge-path between any two edges in each partition, but none between any two edges from different partitions.

From these components of edges, again components of vertices can be obtained:

Definition 2.8 (m-Incident CCs) *Let the* m-incident connected components *of* H*,*

$$HCC(H, m) = \{c_1, \ldots, c_o\} ,$$

be a family of subsets of V *such that for each* m*-incident connected edge-component* $ec_i \in eHCC(H, m)$*, there exists an* m*-incident connected component* c_i *such that for any* v_j*:* $v_j \in c_i \leftrightarrow \exists e \in ec_i \wedge v_j \in e$

For a more intuitive understanding, consider Figure 2.4 showing the two 2-incident connected components of H_s: e_1 and e_2 are 2-incident via t_1 and d_1, as are e_3 and e_4 via t_2 and d_2. e_2 and e_3 are only 1-incident via u_2 which does not suffice to connect them into a single 2-incident connected component.

Note that this leads to u_2 being part of both components; although $eHCC(H, m)$ is a partition of E, $HCC(H, m)$ no longer needs to be a true partition of V and thus a vertex can be part of more than one component.

The nomenclature in these definitions differs from my earlier publications in a less prolific use of the prefix "hyper"; instead of speaking about m-hyperincident e.g. edges, it appears more reasonable to speak about m-incident edges and use hyperincidence for all cases dealing with $m > 1$. This furthermore helps to highlight the fact that m-incidence and all derived concepts are true generalizations, i.e. they fall back to the usual, graph-based definitions for $m = 1$.

2.2.2. Related Work

Connected components are well-researched phenomena: In particular, the emergence of giant components in random graphs was stochastically analyzed over 50 years ago (Erdos and Renyi, 1960). It could be shown that as soon as the relation of edges to vertices reaches one, giant components are asymptotically certain to emerge as the number of vertices approaches infinity. These findings were generalized to hypergraphs by Schmidt-Pruzan and Shamir (1984) without however changing the definition of connectedness. In a similar fashion, Bradde and Bianconi (2009) examine component distributions of 3,3-hypergraphs with an unchanged notion of connectedness. McGlohon et al. (2008) stresses the importance of the comparatively underresearched "next-largest components" and provides an examination of their temporal evolution in real graphs. The notion of *identifiability* has been proposed as a generalization of connectedness (Goldschmidt, 2005) that constrains the maximum number of intermediate edges from an initial set of vertices.

m-incident connected components are not to be confused with m-vertex-connected components, that describe subgraphs which remain connected if one were to remove up to $m - 1$ vertices, or to m-edge-connected components that describe subgraphs which remain connected if up to $m - 1$ edges were removed. Intuitively, this can be seen because m-hyperincident connectivity is a feature enforced between any two edges in the component, whereas m-vertex- or edge-connectity is a global feature of the graph (consider the example of a cycle as a 2-vertex-connected graph). A relation, however, can be constructed

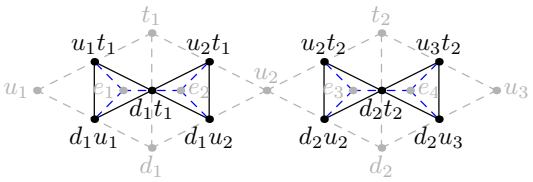


Figure 2.5.: $C_2(H_s)$. Blue, slashed connections connect edges e and their 2-combinations $C_2(e)$. e_1 and e_2 being 2-incident in H through t_1 and d_1 is reflected by d_1t_1 being a part of both $C_2(e_1)$ and $C_2(e_2)$

as follows: For every m-incident connected hypergraph H, its line graph $L(H)$, i.e., the graph in which edges sharing vertices in H are connected vertices, is m-edge-connected if we allow $L(H)$ to be a multigraph and create an edge in $L(H)$ between two vertices in $L(H)$ for every vertex their corresponding edges share in L. Considering Figure 2.4, this means we can remove any d, u, or t without the components being disconnected because by definition, any two e are connected by two such vertices.

Various generalizations of connectivity have been introduced under the label of hyper-connectivity (hence the more specific term of hyperincident connectivity), e.g. for complete lattices (Serra, 1998) or to describe the global connectivity property of graphs (Boesch, 1986) that for every minimum vertex cut D of a graph G, $G - D$ has exactly two connected components, one of which is is an isolated vertex. To conclude, however, to my best knowledge, no generalization of connectedness in the sense defined here has been proposed so far.

2.2.3. Construction

The m-incident connected components of a hypergraph $H = (V, E)$ can be efficiently computed by computing the connected components of a structure I call the m-combination graph of H, and decomposing that structure into regular connected components.

Definition 2.9 (m-Combination Graph) *Let $H = (V, E)$ be a hypergraph of range $> m$. For an edge $e = \{v_1, \ldots, v_k\}$, $C_m(e)$, the* m-combinations *of e are all $\binom{|e|}{m}$ combinations of m of its contained vertices. Let $C_m(H)$ be the* m-combination graph *of H defined by the set of edges E', where E' contains an edge between all members of $C_m(e)$, for all edges $e \in E$.*

The black edges and nodes in Figure 2.5 show the 2-combination graph of H_s: The fact that e_1 connects d_1, u_1, and t_1, e.g., leads to the new nodes u_1t_1, d_1u_1, and d_1t_1 being connected.

Lemma 2.2 *For a given hypergraph $H = (V, E)$, two edges e_1, e_2 belong to the same m-incident connected component in $eHCC(H, m)$ iff their m-combinations $C_m(e_1)$, $C_m(e_2)$ belong to the same connected component in $CC(C_m(H))$.*

Algorithm 1 Generation of 2-incident components from a 3,3-hypergraph.

```
function HCC(E)                                          ▷ E = [(d1, u1,t1), ...]
    E_pairs ← []                                         ▷ Will hold edges of C2(E)
    for (d, u, t) ∈ E do
        for pair ∈ [((d,u), (d,t)), ((d,u), (u,t)), ((d,t), (u,t))] do   ▷ Edges between binary
            if pair ∉ E_pairs then                                       ▷ edges
                E_pairs += pair
            end if
        end for
    end for
    CC ← ConnectedComponents(E_pairs)                    ▷ returns {node: component, ... }
    H ← {}
    for (d, u, t) ∈ E do
        H[(d,u,t)] ← CC[(d, u)]                          ▷ = CC[(d, t)] = CC[(u, t)]
    end for
    return H                                             ▷ H = {(d1, u1, t1): component_id, ... }
end function
```

Proof 2.2 *e_1 and e_2 are connected iff either*

- *e_1 and e_2 directly share an m-combination c: there exists $c = \{v_1, \ldots, v_m\} : c \subset e_1 \wedge c \subset e_2$, i.e. $c \in C_m(e_1) \wedge c \in C_m(e_2)$, or*
- *e_1 shares such a combination with e_x, and e_x and e_2 are connected.*

Either way, e_1 and e_2 are connected in $C_m(H)$ iff they are connected through a chain of m-incident edge pairs, i.e. iff they belong to an m-incident connected edge-component.

Lemma 2.3 *Finding the 2-incident connected components of a 3,3-uniform hypergraph $H = (V, E)$*

a) *can be reduced to finding the connected components of a graph $C_2(H) = (V', E')$, and*

b) *$|V'| \leq 3|E|$ and $|E'| \leq 3|E|$.*

Proof 2.3 *a) follows from the previous lemma. For proving b), consider that H is composed of edges $e = (d, u, t)$. For each e, $C_2(e) = \{(d, u), (d, t), (u, t)\}$. So by adding e, at most these three vertices are added to V' – some of these combinations may already have been added by other edges. Connecting those vertices adds the set of edges $\{((d, u), (d, t)), ((d, u), (u, t)), ((d, t), (u, t))\}$ in $C_2(H)$. Again, vertices can be shared, so 3 is an upper bound as well.*

Algorithm 1 sketches the resulting algorithm, which first creates the new graph $C_2(H)$ and then calls a library function to compute its connected components. The computation of connected components can be performed very effiently, and as the algorithm simply creates a graph with at most three times as many nodes and edges, it follows – and this was

Table 2.2.: Number of connected components and relative size of giant component. Number of 2-incident connected components and relative sizes of giant and next-largest component (GHCC/NLHCC). See Table 1.2 on p. 16 for basic properties of datasets.

name	**#CC**	**|GCC|**	**#2-HCC**	**|GHCC|**	**|NLHCC|**	$\frac{\|\mathbf{NLHCC}\|}{\|\mathbf{GHCC}\|}$
Delicious	42	> 0.999	43,232	0.79	0.04	0.05
Bibsonomy	8	> 0.999	111,121	0.21	0.08	**0.38**
without spam	7	> 0.999	9,727	0.91	0.01	0.011
CiteULike	11	> 0.999	57,738	0.64	0.01	0.016
Visualize.us	7	> 0.999	1,684	0.98	<0.001	<0.001
MovieLens	3	> 0.999	265	0.96	0.01	0.013
AOL	247	> 0.999	26,874	0.99	1.0E-5	1.0E-5

confirmed in practice – that the creation of 2-incident components can be performed very quickly. In contrast, computing the 2-vertex- or 2-edge-connected components of a graph (see Section 2.2.2) is computationally much more intensive. Adding new information is even simpler: for each incoming edge e, we simply have to test if the three tuples in $C_2(e)$ have so far lived in different hyperincident components, and, if so, merge them to a single, new one.

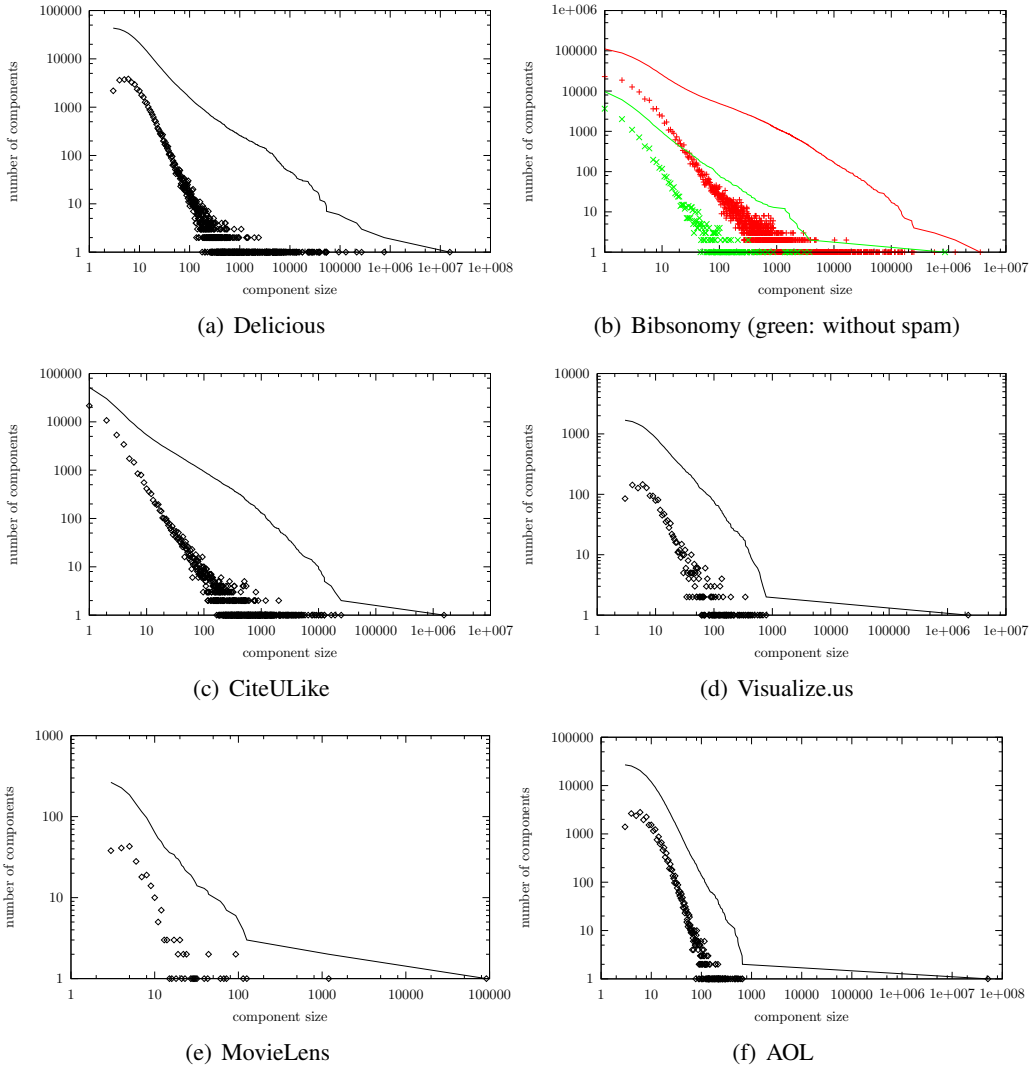
2.2.4. Results

2.2.4.1. Distribution of Component Sizes

All aforementioned networks are decomposed into their regular (1-incident) and 2-incident connected components. Table 2.2 shows the number of resulting regular components and the relative size of the giant component. We can see that in all cases, the networks are basically entirely connected, i.e. the giant component contains more than 99.9% of all nodes. In contrast, a much higher number of components and a less crisp distribution of sizes (except in the AOL dataset) can be observed for 2-incident connected components (for the sake of brevity, any further mentioning of "components" refers to the 2-incident connected components).

Figure 2.6 plots the distribution of the sizes of each dataset's components: In all cases, we see a characteristic power law-like distribution. Only for the smallest, most frequent components plotted on the left of each figure, we see a decay in all datasets except Bibsonomy and CiteULike.

To further explore the relationship between the giant (gcc) and the next largest components (nlc), refer to Figure 2.7, where for each dataset, the size (in terms of number of edges contained) of the top 10 components is displayed on a log scale. In the legitimate tagging datasets, we see a sharp contrast between the sizes of the giant and the next largest component. The difference to the third-largest component and the following ones is not par-



(a) Delicious

(b) Bibsonomy (green: without spam)

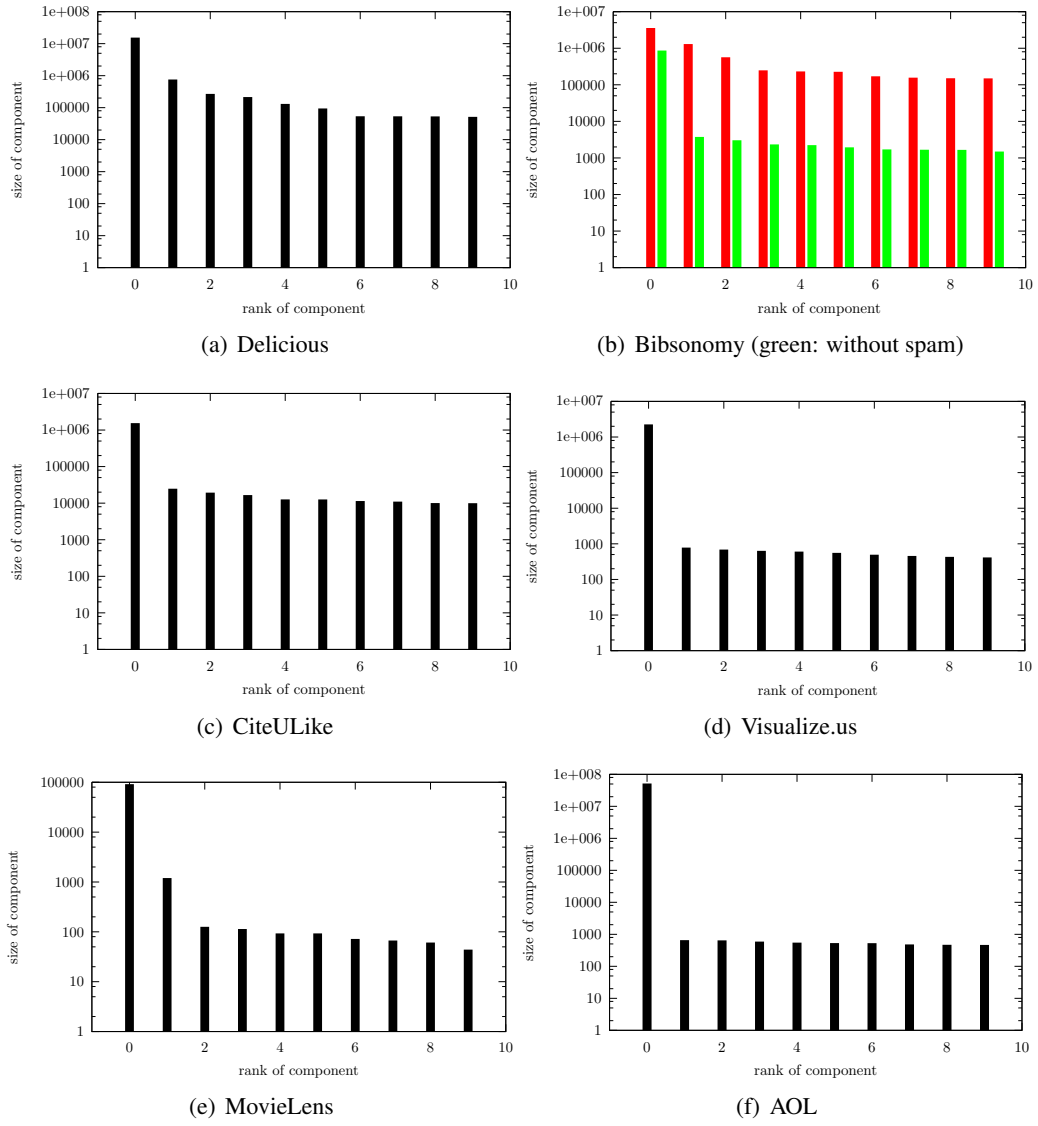
(c) CiteULike

(d) Visualize.us

(e) MovieLens

(f) AOL

Figure 2.6.: Distribution of component sizes for different datasets. Points indicate number of components (y) per component size (x), lines indicate number of components (x) of size >= y.

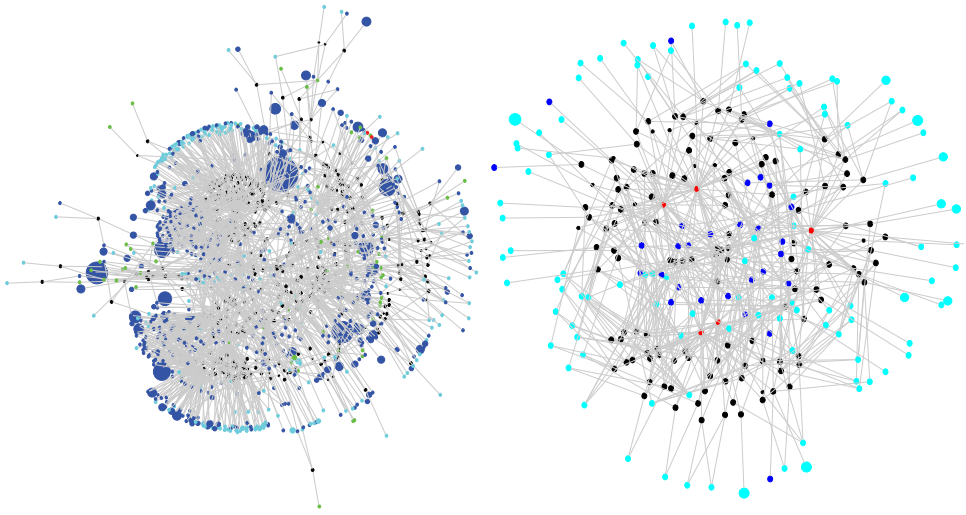


(a) Delicious (b) Bibsonomy (green: without spam)

(c) CiteULike (d) Visualize.us

(e) MovieLens (f) AOL

Figure 2.7.: Size of 10 largest components for different graphs – the size of the next-largest components decays sharply for the regular tagging datasets. For the search log dataset f), the size next-largest components decays even sharper, while it decays very smoothely when spam is included as in (b)/red.



(a) part of giant component (b) spam component

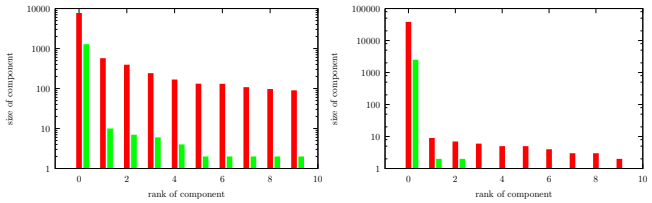
Figure 2.8.: Different hyperincident components in the Bibsonomy dataset. We start with all edges belonging to a single user, then incrementally add 2-incident edges until we either reach a limit of 2000 nodes (left) or no more 2-incident edges exist (right). Green dots: legitimate users, red dots: spammers, blue dots: documents, cyan dots: tags, black dots: edges. Enlarged dots indicate groups of equivalently connected items.

ticularly large. In the spam-contaminated Bibsonomy dataset, the next-largest components are much larger relative to other datasets. Examining the Delicious dataset, we also see relatively sharp difference between giant and second-largest component, but it is slightly weaker expressed than in the other tagging datasets. This may be a result of the moderate level of spam entries reported by Wetzker et al. (2008). In the AOL dataset, in contrast, the dominance of the giant component is much stronger than in any of the tagging datasets. This can be taken as a hint that the decomposition is successfully capturing the fundamentally different underlying formation dynamics.

2.2.4.2. Spam Content of Components

It turns out that the big next-largest components of the spammed Bibsonomy dataset are completely made up by spam users. While the giant component contains a rather large fraction of legitimate users (20% vs 7% in the overall dataset), not a single one can be found in the 128 next-largest components.

Figure 2.8 visualizes the situation: Starting with all edges containing a single user, 2-incident edges are recursively added. In Figure 2.8(a), which starts with a legitimate user, we have to stop expanding at some point (here, after 2000 nodes), as the user is (as is likely) part of the giant component. Figure 2.8(b) shows a typical large non-giant component



(a) User/Document-Graph (b) User/Tag-Graph

Figure 2.9.: Size of 10 largest components of the induced graphs for the Bibsonomy dataset.

composed of almost identical spam users – expansion does not have to be terminated here, as the edges displayed are all that can be reached via a 2-incident path from one of the original user's edges. Almost no such structures of limited size exist for legitimate users; in fact, the quantitative analysis of components' contents was initially motivated by not finding a single example of a component of manageable size not exclusively composed of spam users.

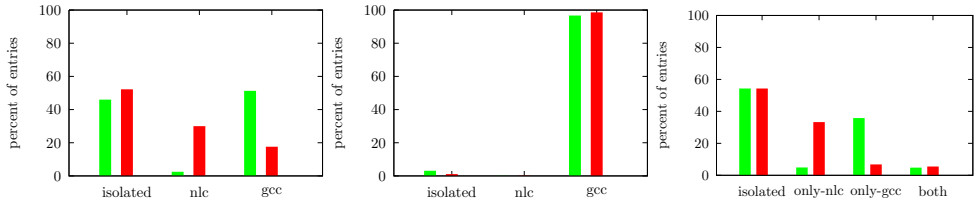
To understand the process that might be responsible for this effect, it is important to stress again the fact that 2-incident connected components induce a partition of *edges*, whereas vertices as documents, users, or tags can be part of several components. Due to this, a spammer u may enter legitimate documents with legitimate tags for disguise, probably connecting to corresponding edges in the giant component – but this does not result in all of their additional entries automatically being connected to the giant component as well. Consider a spammer who tries to appear legitimate by tagging cnn.com with 'news'. The corresponding edge would probably be connected to the giant component and (correctly) appear legitimate. However, by enforcing 2-incident connectivity, this would not lead to another entry containing a spam tag and a spam document necessarily being connected to that component as well, because two nodes of that new edge would have to be connected before, not only the one representing the user. Like this, certain cloaking measures of spammers are automatically countered.

2.2.4.3. Bipartite Connected Components

In this section, a question will be discussed that was originally brought up by my colleague Robert Wetzker: How much of the specific component sizes found in the hypergraph could be due to bipartite connectivity patterns already? In order to examine this questions, let us first set up the following definitions.

The *user/document-graph UD(H)* is defined by the edges $\{(d, u) : \exists(d, u, t) \in H\}$, i.e., tags are ignored and only shared documents imply connectedness. The *user/tag-graph UT(H)* analogously is defined by the edges $\{(u, t) : \exists(d, u, t) \in H\}$. The obvious third possibility, the *document/tag* graph is left out here because it would not contain the labelled users, making a comparison between spam and non-spam versions of the graph difficult.

Figure 2.9 shows the size of the top 10 largest components (as the number of con-



(a) User/Document-Graph (b) User/Tag-Graph (c) Hypergraph

Figure 2.10.: Distribution of spam and non-spam users into the isolated, next-largest and giant components

tained users) of $UD(H)$ and $UT(H)$, for the Bibsonomy dataset. It turns out that the user/document graph in fact shares the signature discrepancy between the size distributions of the spam and the spam-free dataset. The user/tag graph, on the other hand, does not express this effect that strongly. The spam-laden dataset does in fact show next-largest components slightly larger in relative size, but most users are contained in the giant component (please note the change in scale between the two figures).

Let us regard this situation on a more global scale. For this purpose, components are classified as the giant component (gcc), next largest components (nlc) describing any other component containing more than one user, and the remaining isolated components containing only edges from one user. Figure 2.10 shows the distribution for the three different types of graph, again split by spam and spam-free datasets. Note that in the case of the full hypergraph, users can be parts of several components, leading to the additional possibility of a user being both in the giant and a non-giant component. The following results can be read from this figure:

- The user/tag graph is almost fully connected.
- In the user/document graph, on the other hand, the characteristic component size distributions of the hypergraph can partially already be found.
- The ratio of spam users in the giant component, however, is lower in the hypergraph than in the document/user graph.

To conclude, it appears that although much of the component size distribution in the hypergraph is caused by connectivity patterns in the user/document graph, the hypergraph does contain additional structural information leading to an even sharper contrast between spam and non-spam. In the next and final part, the effects of these findings for spam detection will be discussed.

2.2.4.4. Application for Spam Detection

Based on the results of Figure 2.10, a simple classification scheme can be devised that directly uses the component membership information: Users receive a spam rating of -1 if contained in the giant component, 0 if isolated, and 1 if they are members in a large but

	min # docs/user			
	0	**1**	**10**	**100**
User/Document	0.73	0.78	0.81	0.84
User/Tag	0.49	0.49	0.50	0.50
Hypergraph	0.73	0.78	0.84	0.88
Combined	0.76	0.81	0.87	0.91

Table 2.3.: AUC values of the different connectivity-based classifiers

non-giant component. These scores can be computed for $UD(H)$, $UT(H)$ and H. The predictions based on $UD(H)$ and H's components are combined into a single predictor by adding the scores obtained from membership in both graphs' components. Assuming that many of the isolated components are caused by users which have tagged only very few documents, the classifiers are also evaluated on the subsets of users having tagged more than 1, 10, or 100 documents.

Table 2.3 shows the performance values when applying these heuristics. As was to be expected from the distribution of users, $UT(H)$ does not provide any usable information. The other approaches however do well, particularly on users with more documents. The hypergraph's connectivity is slightly more informative than $UD(H)$'s. However, the performance of the combined predictor exceeds that of both individual ones. This suggests that neither graph's connectivity is redundantly encoded by the other's, i.e., there must be users which are in a next-largest component in one graph but not in the other.

2.2.5. Conclusion

I have introduced a generalization of connected components and shown their applicability to 3-partite 3-uniform hypergraphs. Tagging activity, in the datasets examined, creates a specific distribution of connected components, with the second-largest component being one to two orders of magnitude smaller than the giant one. Different creation processes create different pattern. In particular, spamming activity appears to create a large number of next-largest components of much higher relative size than their non-spam counterparts. These components, in the data examined, are made up completely by spammers: In the examined, spam-annotated dataset examined, not a single legitimate user was found in the second- to 129th largest components.

These findings were used to create classifiers for detecting spam users in tagging datasets. Obviously, the unsupervised classifier does not reach the performance of supervised classifiers with domain-specific features. However, it can be used without labeled information and might provide first indications of users that need to be further examined, particularly when applied to users which have already had the time to form some revealing connectivity patterns.

I believe these findings are good news – not only for researchers in spam detection, but for anyone interested in learning from tagging data. In bookmarking their favourite resources,

users create traces of complex cognitive processes. The results indicate that the complexity these processes is mirrored by the structures emerging from the use of social bookmarking systems, too subtle to be easily reproduced by malevolent spammers. This reinforces the implicit assumption that underlies work on tag analysis in general: That tagging data is a rich source of non-trivial latent information.

3. Multi-Partite Community Detection

This chapter defines the task of multi-partite community detection. It highlights the special requirements of this task and provides both terminology and quantitative criteria for discussing these issues later on.

Section 3.1: Motivation generally describes the task of multi-partite community detection. A toy example illustrates the basic process.

Section 3.2: Definitions puts the intuitions of the previous section into mathematical terms.

Section 3.3: Synthetic Datasets defines three families of synthetic datasets which represent different challenges such that different community detection algorithms can be directly evaluated with respect to those challenges.

The evaluation methodology (synthetic datasets and test routines) described in this chapter is provided as a downloadable software package, *mpcd* (Multi-Partite Community Benchmarks) – see Appendix A.1.2 for details.

3.1. Motivation

In Section 1.2.2.1, community detection for "normal" (non-partite) graphs has been introduced: The goal is to group nodes which are structurally close in the graph for some definition of "close", both for the sake of identifying those groups of similar nodes – which may point to functional, semantic or social relations in the underlying data – and in order get a clearer view of a graph's macro-structure. These goals basically remain the same in the multi-partite case.

Section 1.2.2.3 has introduced bipartite community detection, finding distinct community assignments for the two partitions of a bipartite graph. Multi-partite community detection aims to generalize bipartite community detection to k, k-hypergraphs with arbitrary k.

Before proceeding to define multi-partite community detection and related vocabulary more formally, let us consider an example showing different possible community assignments in a small graph.

3.1.1. An example

The hypergraph H_s is given by the edges listed in Table 3.1(a) and visualized in Figure 3.1(a). As we can see, it is defined by 10 edges connecting 3 documents, 2 users and 2 tags. The goal is to find one partition for each of those three sets.

3.1.1.1. Description

Figures 3.1(a) - 3.1(e) show not one, but five possible community assignments. They build on each other, starting with the trivial solution of assigning each node to a single community. From there on, each community assignment is like the previous one, except one community has been merged with another one. Even though community detection primarily aims to provide a single partition of a graph's elements into communities, such a recursive process is typical for the optimization of community assignments. Girvan and Newman (2002) and Newman (2006), e.g., describe top-down approaches of recursively splitting communities, starting with all elements in the same community, whereas e.g. Clauset et al. (2004) describe a bottom-up algorithm that joins two communities at each step, starting with each element

Table 3.1.: Sample data as visualized in Figure 3.1

(a) edges representing H_s

d	u	t	d	u	t
1	1	1	1	2	1
1	1	2	1	2	2
2	1	1	2	2	1
2	1	2	2	2	2
3	1	1	3	2	1

(b) sample output of community detection

domain	el.	el.	modularity	step
2	1	2	0.351111	1
1	2	1	0.402223	2
3	2	1	0.455556	3
1	3	1	0.500000	4

in its own community. In both cases, even though a single, optimal state is returned as a result, such recursive processes imply an entire clustering hierarchy. Later on, the analysis will at times be extended to this hierarchy. Figure 3.1(f) highlights the recursive nature of the merging process – three dendrograms, sharing a temporal scale (y), visualize the progressive clustering in each domain.

Table 3.1(b) describes the individual merging operations that lead to community assignments (b)–(e). The first line, "2 1 2 ...", e.g., indicates that element 1 and 2 of domain 2 (i.e., users) are the first to be merged. Correspondingly, after four joins, one element per domain remains in Figure 3.1(e). This table also represents an example of the output community detection algorithms should produce. The fourth column indicates the quality of the community assignment after each merge (here, it shows the value of the coupled bipartite modularity that will be introduced in Section 4.2, which however is not important for our example).

3.1.1.2. Interpretation

Several things can be observed in Figure 3.1. First of all, the notion of tripartite community structure becomes evident: An overall community assignment consists of a set of three individual community assignments, one per domain. Also, it intuitively introduces *community edges*, groups of edges incident to elements from the same communities, that will be defined more formally in the next section and are visualized here as black or gray circles.

Furthermore, we see how choosing the optimal solution out of a a hierarchy of community assignments is not obvious even for this simple example. (c) is a sweet spot because all redundant elements, but none with different connectivity have been grouped. In real data, however, using identical connectivity as a selection criterion might result in a high number of small communities, as most elements are connected slightly differently. So a solution like (d) might be more appropriate: Although the difference between Tag 1 and 2 is hidden, the gray community edge to Document 3 indicates that this element is less connected to the user and the tag community than the other documents. In terms of our sample quality measure, (see Table 3.1(b)), the last (trivial) solution should in fact be considered optimal (this special case – the trivial solution becoming optimal – is examined in more depth in Section 4.2.4.3).

Table 3.2.: 3-partite hierarchical clustering induced by the joins in Table 3.1(b)

$\mathbf{k}$	**1**			**2**		**3**	
	$\mathbf{d_1}$	$\mathbf{d_2}$	$\mathbf{d_3}$	$\mathbf{u_1}$	$\mathbf{u_2}$	$\mathbf{t_1}$	$\mathbf{t_2}$
σ_k^0	1	2	3	1	2	1	2
σ_k^1	1	2	3	1	1	1	2
σ_k^2	1	1	3	1	1	1	2
σ_k^3	1	1	3	1	1	1	1
σ_k^4	1	1	1	1	1	1	1

Finally, it should be noted that the path from (a) to (e) – i.e. which nodes to join at each step – is only one of many possible ones. The actual challenge lies in constructing these paths. An alternative path from (a) to (e) might e.g. first join Documents 1+3, and join Document 2 later on. Even without any quantitative definition, this seems like a bad idea, given that Documents 1 and 2 are identically connected, whereas 1 and 3 are not.

3.1.2. Challenges of Multi-Partite Community Detection

Why can't we simply use existing community detection techniques to perform analyses like the one just shown? Refering back to the definition of modularity (Definition 1.2 on page 8), we can find the following implicit assumptions are made:

- The goal is to create a single set of community assignments.
- Nodes are either in the same community, or not.
- Edges are binary.

The previous assumptions (the last one for $k > 2$) become invalid when dealing with (k, k) hypergraphs. This leads to the three challenges for multi-partite modularity detection:

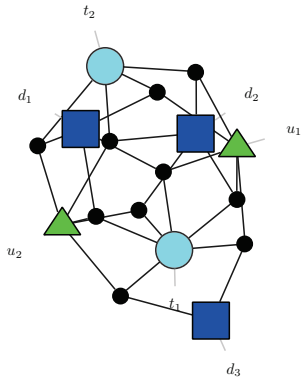
1. **Distinct community structures** Different domains may have different community structures: Communities in one domain may not exactly correspond to another community in another domain; there may also be different numbers of communities. Therefore, k different sets of communities should be supported.

2. **Correspondence instead of equality** If we assume distinct community structures per domain, the notion of connected nodes (which, by definition of partiteness, come from different domains) incident to the "same" community becomes obsolete. If the number of communities per domain are equal, speaking of elements from different domains being in the "same" domain may be tempting if correspondence is strong. But if, for example, there are two communities A and B in one domain and only one community C in another, it becomes clear that neither A or B are the same community as C. Instead, a generalized notion of elements from different domains belonging to *corresponding* communities is required.

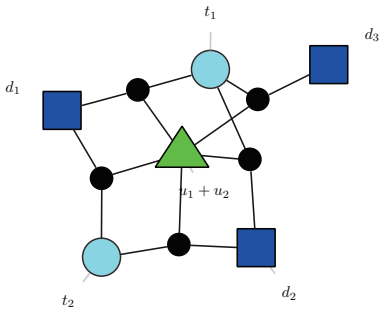
   We also need to take care not to punish the non-adjacency of (necessarily non-adjacent) elements in the same community within a single domain, as done by standard modularity.

3. **Hyper-Incidence** While the previous two points have been pointed out before (e.g. by Murata (2009)) for bipartite graphs, dealing with hypergraphs additionally requires us to generalize a binary notion of correspondence to k-ary one, measuring the joint correspondence between all involved communities.

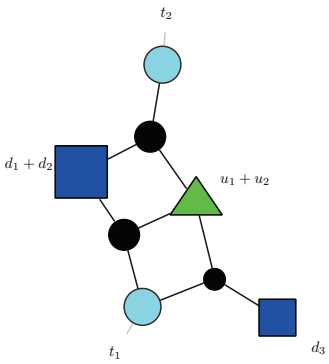
The three families of synthetic hypergraphs that will be introduced later on in this chapter highlight these challenges. However, let us now define some required terminology.



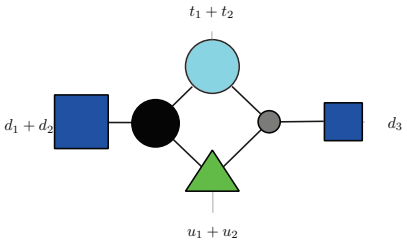
(a) Original hypergraph. Black dots represent hyperedges. Each coloured shape represents a single object.



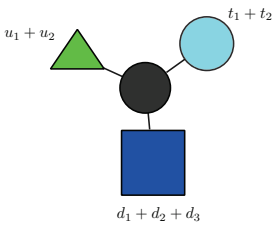
(b) After the first join (user 1+2), each black dot now represents a community edge of two edges (*,*,user 1) and (*,*,user 2).



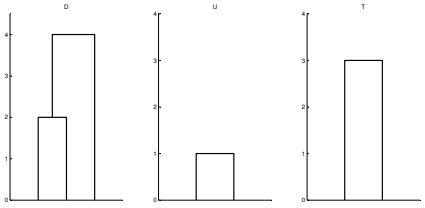
(c) After the second join (document 1+2), the basic structure of the hypergraph becomes apparent: documents 1 and 2 connected to tags 1 and 2, with document 3 only connected to tag 1.



(d) The third join (tag 1+2) introduces *lossy* compression: The right community edge implies edges between tag 2 and document 3 that don't exist and is hence coloured gray instead of black.



(e) After the fourth join (documents 1+2 and 3), all elements from the same domain have been assigned to a single community.



(f) The combined dendrograms of this process, for documents, users, and tags. Figure (a) represents the bottom, Figure (e) the top of the trees.

Figure 3.1.: A sample hypergraph H_s, given by the edges in Table 3.1(a), clustered by the steps given by Table 3.1(b) / a 3-partite hierarchical clustering $\boldsymbol{\sigma}$ given by Table 3.2

3.2. Definitions

3.2.1. Community Assignments

The result of community detection is a function assigning nodes to communities. Let us fix this formally for the k-partite case.

Definition 3.1 (k-partite community assignment) *Let a* k-partite community assignment *for a k,k-hypergraph H be defined as a k-tuple $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$ of* community assignment functions $\sigma_d : V_d \to C_d$*, projecting the vertices in V_d onto* communities $C_d \subset \mathbb{N}$.

Let the codomain of $\boldsymbol{\sigma}$, $\mathbf{C}_{\boldsymbol{\sigma}} = C_1 \times \ldots \times C_k$ be the community space *induced by $\boldsymbol{\sigma}$. Applying each element of $\boldsymbol{\sigma}$ to each element v_i of an edge $e = (v_1, \ldots, v_k)$ lets us interpret $\boldsymbol{\sigma}$ as a vector-valued function projecting edges into the community space:*

$$\boldsymbol{\sigma} : E \to \mathbf{C}, \boldsymbol{\sigma}(e) = (\sigma_1(v_1), \ldots, \sigma_k(v_k)).$$

Let $V_{d,i}(\boldsymbol{\sigma})$ be the partitions of V_d (the nodes in domain d) induced by $\boldsymbol{\sigma}$, i.e. the content of the communities:

$$V_d = \bigcup_{i \in C_d(\boldsymbol{\sigma})} V_{d,i}(\boldsymbol{\sigma}), \;\; v \in V_{d,i}(\boldsymbol{\sigma}) \Leftrightarrow \sigma_d(v) = i.$$

Let the weight a *of $V_{d,i}(\boldsymbol{\sigma})$ be the number of edges incident to the elements of the partition,*

$$a_{d,i}(\boldsymbol{\sigma}) = |\{e \in E : \sigma_d(e_d) = i\}|.$$

Literature on non-partite modularity assumes a single community assignment function σ, and also work on bipartite modularity assumes a single σ containing assignments for elements of both domains. For many discussions, I have found it handy to address the different domains explicitly, using e.g. the above notation of a vector-valued $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$ and distinct sets of nodes. I will typically provide definitions in both notations and use whichever notation is more concise in the further discussion. When switching between multi-partite (left) and non-partite (right) notations, it is useful to keep in mind that for $\mathbf{c} = (i, j, k)_{\boldsymbol{\sigma}}$,

$$|\mathbf{c}_{\boldsymbol{\sigma}}| = e_{i,j,k}M \text{ and } a_{d,i}(\boldsymbol{\sigma}) = a_i M.$$

In practice, community assignments come in groups. Any community detection approach that works by binary merges implicitly creates an ordered list of community assignments. This list is of some interest in itself, as we may

- choose to pick another intermediate result as the final assignment than the algorithm chose which originally created the assignments (see Section 4.2.3).
- need to talk about different hypothetical sets of clusterings (see Section 4.4),
- compute global properties to understand the clustering process (see Section 4.5), or
- use it to allow an interactive exploration of the clustering results (see Section 5.2).

Let us therefore define this structure more precisely.

Definition 3.2 (k-partite hierarchical clustering) *A* k-partite hierarchical clustering *is an ordered list*

$$\Sigma = \boldsymbol{\sigma}^0, \ldots, \boldsymbol{\sigma}^{N-k}$$

of $N - k + 1$ k*-partite community assignments .*

$\boldsymbol{\sigma}^0 = (\sigma_1^0, \ldots, \sigma_k^0)$ *assigns each element to an individual community, i.e.,*

$$\forall i \in K,\ \forall (v, v') \in V_i \times V_i : v \neq v' \rightarrow \sigma_i^0(v) \neq \sigma_i^0(v').$$

$\boldsymbol{\sigma}^{N-k}$ *assigns, per domain, each element to a single community, i.e.*

$$\forall i \in K,\ \forall (v, v') \in V_i \times V_i : \sigma_i^{N-k}(v) = \sigma_i^{N-k}(v').$$

Each intermediate community assignment differs from its predecessor in that two communities have been merged into one. The difference between two consecutive community assignments $\boldsymbol{\sigma}^\tau$ *and* $\boldsymbol{\sigma}^{\tau+1}$ *is described as a* join $\varphi = (d, s, t)$ *that contains the source community* s *and target community* t *from domain* d *which have been merged. Let us define the application of a join* φ *to a community assignment* $\boldsymbol{\sigma}$ *as* $\boldsymbol{\sigma} \circ \varphi$ *such that*

$$\varphi = (d, s, t) \Rightarrow \boldsymbol{\sigma}^\tau \circ \varphi = \boldsymbol{\sigma}^{\tau+1},\ \sigma_{d'}^{\tau+1}(v) = \begin{cases} t & d = d' \wedge \sigma_d^\tau(v) = s \\ \sigma_{d'}^\tau(v) & \textit{else.} \end{cases}$$

Figures 3.1 (a) through (e) depict the five different stages of community assignments given by a particular hierarchical clustering. Table 3.2 enumerates the exact community assignments at each step and domain.

3.2.2. Community Edges

By merging elements into communities, a new graph is implicitly constructed in which the new nodes correspond to communities of original nodes, and the edges correspond to groups of edges running between elements from the same community. Let us call those implied, aggregate edges *community edges*.

Definition 3.3 (community edges) *Let a* community edge

$$(c_1, \ldots, c_k)_{\boldsymbol{\sigma}} = \mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}} = C_1 \times \ldots C_k$$

be a set of edges equivalent under $\boldsymbol{\sigma}$*:*

$$\mathbf{c}_{\boldsymbol{\sigma}} = \{e \in E : \boldsymbol{\sigma}(e) = \mathbf{c}\}.$$

Individual communities c_i *indexed by* $\mathbf{c}_{\boldsymbol{\sigma}} = (c_1, \ldots, c_k)$ *are called* incident *to* $\mathbf{c}_{\boldsymbol{\sigma}}$.

In simpler terms: a list of communities $\mathbf{c} = (c_d, c_u, c_t)$ looks a lot like an edge (d, u, t), and community edges make this relation explicit by relating $\mathbf{c}$ to all the edges between elements from the communities it contains. As an example, the community assignment $\boldsymbol{\sigma^2}$ as given by Table 3.2 clusters the nodes of H_s into 2, 1, and 2 communities respectively. This gives rise to four community edges:

$$
\begin{aligned}
(1,1,1)_{\boldsymbol{\sigma^2}} &= \{(1,1,1),(1,2,1),(2,1,1),(2,2,1)\} \\
(1,1,2)_{\boldsymbol{\sigma^2}} &= \{(1,1,2),(1,2,2),(2,1,2),(2,2,2)\} \\
(3,1,1)_{\boldsymbol{\sigma^2}} &= \{(3,1,1),(3,2,1)\} \\
(3,1,2)_{\boldsymbol{\sigma^2}} &= \emptyset
\end{aligned}
$$

Arenas et al. (2007) examine the effects of merging nodes to modularity optimization, and, in the process, also discuss the properties of aggregate edges. Since their work is focussed on non-partite graphs, however, they have to deal with other phenomena, in particular emerging self-loops which do not play a role in the multi-partite case because the nodes that are merged are, by definition, not incident to each other. Here, instead, it will be necessary later on to discuss certain other properties of community edges:

Definition 3.4 (properties of community edges) *The* weight *of a community edge $\mathbf{c}_{\boldsymbol{\sigma}}$ refers to the number of edges inside it, is notated as $|\mathbf{c}_{\boldsymbol{\sigma}}|$, and is implicitly defined already, since community edges are defined as the set of edges they unite.*

The volume *of a community edge $\mathbf{c}_{\boldsymbol{\sigma}}$ is given by*

$$\mathrm{vol}(\mathbf{c}_{\boldsymbol{\sigma}}) = \prod_{i \in K} |v \in V_i : \sigma_i(v) = c_i|,$$

i.e. the maximum number of edges that could theoretically belong to $\mathbf{c}_{\boldsymbol{\sigma}}$ if there was an edge between all involved elements.

The density *of a community edge $\mathbf{c}_{\boldsymbol{\sigma}}$ is given by*

$$\rho(\mathbf{c}_{\boldsymbol{\sigma}}) = \frac{|\mathbf{c}_{\boldsymbol{\sigma}}|}{\mathrm{vol}(\mathbf{c}_{\boldsymbol{\sigma}})},$$

i.e., the ratio of the actual to the maximum number of contained edges.

The relative importance $r(\mathbf{c}_{\boldsymbol{\sigma}}, i)$ *of an community edge $\mathbf{c}_{\boldsymbol{\sigma}}$ for its ith incident community c_i is given by*

$$r(\mathbf{c}_{\boldsymbol{\sigma}}, i) = \frac{|\mathbf{c}_{\boldsymbol{\sigma}}|}{|\{e = (v_1, \ldots, v_k) \in E : \sigma_i(v_i) = c_i\}|} = \frac{|\mathbf{c}_{\boldsymbol{\sigma}}|}{a_{d,c_i}(\boldsymbol{\sigma})},$$

describing the ratio of the number of edges in $\mathbf{c}_{\boldsymbol{\sigma}}$ to the total number of edges incident to elements in c_i.

The community edges of importance α *are a subset of all possible community edges as given by the community space $\mathbf{C}_{\boldsymbol{\sigma}}$:*

$$\mathbf{C}_{\boldsymbol{\sigma},\alpha} = \{\mathbf{c}_{\boldsymbol{\sigma}} \in C_{\boldsymbol{\sigma}} : \exists i : r(\mathbf{c}_{\boldsymbol{\sigma}}, i) \geq \alpha\},$$

i.e. those community edges which have at least a relative importance of α for at least one of their incident communities.

The black circles in Figure 3.1(a) through 3.1(d) are community edges of density 1. Their size differences in (c) reflect differences in volume (4 vs 2). The gray circle in (d) represents community edges with densities< 1. The single circle in (e) also refers to a density< 1 – it is nevertheless plotted in black, because its density value is the highest (since only one) in the plot and normalization is applied.

To give a final intuition about community edges: Only the black dots in Figure 3.1(a) represent single edges – the other figures use black dots to represent several edges: community edges. For an initial community assignment $\boldsymbol{\sigma}^0$ which assigns each node to a single community, the set of community edges furthermore is equivalent to the set of edges. Using the notation introduced here, the new graphs implied by creating larger communities can be described naturally as entities which live in the same space as the original graph on one side of the spectrum and a graph consisting of just one hyperedge on the other end of it.

3.2.3. Evaluation Measures

Throughout this thesis, a recurring topic is the evaluation of obtained community assignments. This section introduces normalized mutual information (NMI), an established supervised quality measure that rates the similarity between two community assignments and will be used in particular to compare predicted with correct assignments. In cases where no ground truth is available, a quality measure may be required nevertheless to compare different clusterings. While modularity is in fact such a measure, it is beneficial to have a "neutral" measure since different versions of modularity are being compared in the first place. For this purpose, a compression measure based on minimum description length is introduced as well.

3.2.3.1. Normalized Mutual Information

Definition 3.5 (NMI) *The* Normalized Mutual Information *between two community assignments σ_A and σ_B is defined as*

$$\mathrm{NMI}(\sigma_A, \sigma_B) = \frac{2\mathrm{MI}(\sigma_A, \sigma_B)}{\mathrm{H}(\sigma_A) + \mathrm{H}(\sigma_B)},$$

where MI *is mutual information and* H *entropy.*

Danon et al. (2005) give the following implementation-oriented definition of MI *and* H*: Let $a = |C_{\sigma_A}|$ and $b = |C_{\sigma_B}|$ be the number of communities found by the two clusterings. Let $\mathbf{N} \in \mathbb{N}^{a \times b}$ be a confusion matrix in which rows correspond to communities in σ_A, columns correspond to communities in σ_B, and elements N_{ij} count the elements which have been assigned to i by σ_A and to j by σ_B. Let furthermore $||\mathbf{N}||$ be the sum of all entries in $\mathbf{N}$ and $Ni.$ and $N.j$ denote the sum of entries of the ith row and jth column of $\mathbf{N}$, respectively. Then the mutual information can be computed as*

$$\mathrm{MI}(\sigma_A, \sigma_B) = -\sum_{i=1}^{a}\sum_{j=1}^{b} N_{ij} \log\left(\frac{N_{ij}||\mathbf{N}||}{N_{i.}N.j}\right)$$

and entropy is given by

$$\mathrm{H}(\sigma_A) = \sum_{i=1}^{a} N_{i.} \log\left(\frac{N_{i.}}{||\mathbf{N}||}\right), \qquad \mathrm{H}(\sigma_B) = \sum_{j=1}^{b} N_{j.} \log\left(\frac{N_{j.}}{||\mathbf{N}||}\right).$$

If both assignments predict a single community, the denominator vanishes. In this case, we define NMI *to be 1, because the correct assignment has been predicted.*

To evaluate a k-partite community assignment $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$, the mean NMI *of the domain-specific assignments is computed:*

$$\mathrm{NMI}(\boldsymbol{\sigma}) = \frac{1}{k} \sum_{i=1}^{k} \mathrm{NMI}(\sigma_i).$$

This measures the information one community assignment contains about the other, normalized to values between 1 in the case of perfect agreement and 0 for complete independence, e.g. if one of the assignments assigns all elements to one community.

A simpler alternative might be the approach used by e.g. Newman (2004). Assuming that σ_A is the correct assignment and σ_B the predicted one, each predicted cluster C_B is assigned to a correct cluster in C_A such that $C_A = \operatorname{argmax} c(\sigma_A(v) = c \wedge \sigma_B(v) = C_B)$. Then each element assigned to C_B which is not in σ_A is considered wrong, and the quality is defined as the fraction of correctly clustered elements. As argued e.g. by Danon et al. (2005), this measure may be problematic in certain cases. Consider, e.g., a simple case where elements numbered from 1 to 6 are assigned to clusters to $[1, 1, 2, 2, 3, 3]$ by the correct assignment σ_A. Now consider two possible assignments: σ_B which assigns the elements to $[1, 1, 1, 1, 3, 3]$ and σ_C, assigning to $[1, 1, 1, 3, 3, 3]$. In both cases, the two elements previously assigned to cluster 2 are mis-aligned. The simple correctness measure would therefore assign a score of $\frac{2}{3}$ to both. However, the two elements are still in the same cluster in σ_B but torn apart by σ_C. NMI accounts for this different by assigning a score of 0.73 to σ_B and 0.51 to σ_C. It is this sensitivity to border cases which has made NMI a frequently adapted quality measure in the literature (see e.g. recent papers on benchmarking by Lancichinetti et al. (2008)), and it will be used throughout this thesis.

3.2.3.2. Minimum Description Length

For cases in which no ground truth is available, a measure is introduced here that describes the amount of information required to transmit a k-dimensional adjacency tensor under a given community assignment. It follows the basic approach and argumentation put forth by Chakrabarti et al. (2004) for a two-dimensional case. When sorting nodes by community membership, well-connected communities should translate to densely populated regions in the sorted adjacency tensor, which results in better compression, linking community quality to information-theoretic terms. Liu and Murata (2010) have recently proposed to directly optimize a similar measure as a method for community detection. Here however, it is only used for comparing different results.

The general approach consists of dividing the tensor into sub-tensors and transmitting them individually with an optimized encoding based on their sparseness or density. Since

the structural information about the decomposition needs to be transmitted as well, the cost for transmitting the adjacency tensor of a hypergraph $G = (V, E)$ with domains K under a community assignment $\boldsymbol{\sigma}$ can be formulated as

$$\mathbf{I}(G, \boldsymbol{\sigma}) = \mathbf{I}(G, \boldsymbol{\sigma})_{\text{coding}} + \mathbf{I}(G, \boldsymbol{\sigma})_{\text{structure}} .$$

$\mathbf{I}(G, \boldsymbol{\sigma})_{\text{coding}}$ represents the actual coding costs, i.e. the amount of information required to transmit whether an individual k-tupel of elements are adjacent or not. Without compression, each tuple requires a single bit. Assuming that elements are grouped such that the adjacency tensor is partitioned in sparse and dense regions, this can be achieved more efficiently, transmitting the information for each community edge separately:

$$\mathbf{I}(G, \boldsymbol{\sigma})_{\text{coding}} = \sum_{\mathbf{c}_{\boldsymbol{\sigma}} \in C} \text{vol}(\mathbf{c}_{\boldsymbol{\sigma}}) H(\rho(\mathbf{c}_{\boldsymbol{\sigma}})).$$

$\text{vol}(\mathbf{c}_{\boldsymbol{\sigma}})$ is the size of the sub-tensor being encoded, i.e. the number of bits to transmit. $\rho(\mathbf{c}_{\boldsymbol{\sigma}})$ describes the ratio between ones and zeroes in the current sub-tensor, influencing the encoding costs per bit: The entropy $H(p)$ (Shannon, 1948) describes the information required on average to transmit a single bit, based on the probability of it being 1, and is given for the binary case as

$$H(p) = -p \log_2 p - (1 - p) \log_2 (1 - p).$$

This function has its maximum at $p = \frac{1}{2}$, such that any imbalance towards more ones or more zeroes enables compression schemes that allow for more efficient transmission. This is how particularly dense and sparse sub-tensors contribute to lower encoding costs. Please note that no statement is made about the actual encoding used to transmit these bits – all we need from information theory here is the average cost of this implied coding.

Still, structural information needs to be included in order to inform the potential receiver about the meaning of the data to be transmitted. In order to see the necessity of this, imagine reducing the sub-tensor size to single bits – this would reduce the coding costs to 0. However, having overall transmission costs of 0 for a non-empty tensor would not make sense; the answer is that all information would go into this structural part. Indeed, once the position and size of each non-empty sub-tensor of size 1 is transmitted, no further encoding of the actual contents would be required, at the cost of exploding structural costs however. In order to express the trade-off between coding and structural costs, we need to spell out the encoding of this structural part, which is made up of four different components:

$$\mathbf{I}(G, \boldsymbol{\sigma})_{\text{structure}} = \mathbf{I}(G, \boldsymbol{\sigma})_{\text{count}} + \mathbf{I}(G, \boldsymbol{\sigma})_{\text{assignment}} + \mathbf{I}(G, \boldsymbol{\sigma})_{\text{density}} + \mathbf{I}(H, \boldsymbol{\sigma})_{\text{hascontent}}.$$

$$\mathbf{I}(G, \boldsymbol{\sigma})_{\text{count}} = \sum_{i \in K} 32 + \log_2 |V_i|$$

computes the cost of transmitting the number of nodes and communities for all domains i. The number of nodes $|V_i|$ per domain cannot be known in advance and is therefore difficult to compress – Chakrabarti et al. (2004) employ a principle of universal code length for

integers of unknown range, but here it seems easier to assume that the number of nodes per domain will always fit into a 32-bit integer and simply transmit this number uncompressed. Once this number is known, however, the following number of communities – maximally as large as $|V_i|$ – can be encoded using only as many bits as required.

$$\mathbf{I}(G, \boldsymbol{\sigma})_{\text{assignment}} = \sum_{i \in K} |V_i| \log_2 |C_i|$$

computes the cost of transmitting the community assignment for each node. Note that an efficient coding of the community id by $\log_2 |C_i|$ is possible because the number of communities has been transmitted before.

$$\mathbf{I}(G, \boldsymbol{\sigma})_{\text{density}} = \sum_{\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}}, |\mathbf{c}_{\boldsymbol{\sigma}}| > 0} \log_2 \operatorname{vol}(\mathbf{c}_{\boldsymbol{\sigma}})$$

computes the cost of transmitting the number of ones per community edge, informing the implicit compression scheme.

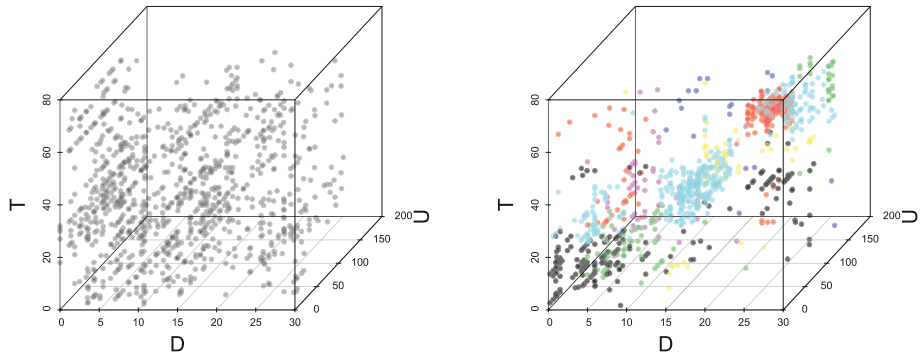
$$\mathbf{I}(H, \boldsymbol{\sigma})_{\text{hascontent}} = \min \left(\prod_{i \in K} |C_i|, \quad |\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}} : |\mathbf{c}_{\boldsymbol{\sigma}}| > 0| \sum_{i \in K} \log_2 |C_i| \right) + 1$$

computes the cost for transmitting for each potential community edge out of $\mathbf{C}_{\boldsymbol{\sigma}}$ whether it contains edges, since the previous encodings only transmit for actual, i.e. non-empty community edges. This can either be a single bit per community edge – all community edges are listed in order, and the first bit encodes whether content follows – or a list of positions for all non-empty community edges. Although this costs additional information required to encode the indices of the incident communities, it may be advantageous if many communities exist, so the encoding can change between the two schemes to choose the more efficient one. The single bit at the end is used to transmit which encoding is chosen.

Using this coding scheme, all necessary information to unambiguously reconstruct the compressed adjacency tensor is provided. The final costs can be summarized by

$$\begin{aligned} \mathbf{I}(G, \boldsymbol{\sigma}) = & \sum_{\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}}, |\mathbf{c}_{\boldsymbol{\sigma}}| > 0} (\operatorname{vol}(\mathbf{c}_{\boldsymbol{\sigma}}) H(\rho(\mathbf{c}_{\boldsymbol{\sigma}})) + \log_2 \operatorname{vol}(\mathbf{c}_{\boldsymbol{\sigma}})) \\ + & \sum_{i \in K} (32 + \log_2 |C_i| + |V_i| \log_2 |C_i|) \\ + & \min \left(\prod_{i \in K} |C_i|, \quad |\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}} : |\mathbf{c}_{\boldsymbol{\sigma}}| > 0| \sum_{i \in K} \log_2 |C_i| \right) + 1. \end{aligned}$$

Example Figure 3.2 illustrates the relationship between community assignments and minimum description length, anticipating some technology and concepts from later chapters. We examine a constrained tag expansion (see Section 5.1.2; basically a number of edges around a given tag, obtained in a particular fashion), here around the tag "Charity" from the Delicious dataset (see Section 5.3.2 for further discussion of the results).



(a) Unsorted adjacency tensor

(b) Sorted adjacency tensor

Figure 3.2.: The same adjacency tensor, once unsorted and once with axis elements grouped by community membership. Blocks of same colour group edges that connect elements from the same communities (community edges). The sorted tensor can be seen to be decomposed into subcubes that are either sparsely or densely populated with edges, allowing better compression.

Figure 3.2(a) shows the unsorted adjacency tensor. In the absence of community assignments, a better way of encoding this tensor than the one proposed above is to simply transmit the coordinates of all edges. As the subgraph contains 30 documents, 154 users and 73 tags, each edge's coordinates are encoded by 5, 8, and 7 bits respectively. Having 1001 edges to transmit and assuming a 8-bit header per domain encoding the number of elements yields a size of 20044 bits to transmit the edges in the graph / the position of non-zero elements in the tensor.

Figure 3.2(b) shows the same tensor, only that axes are sorted by community membership as assigned by the balanced modularity optimization introduced in Section 4.5.5. Instead of edges being almost uniformly distributed in the three-dimensional space, distinct regions of dense and sparse population can be identified. Edges belonging to the same community edge (i.e. connecting elements from the same communities) are shown as blocks of identical colors (remote blocks of same colour do not have a special relation and are caused by lack of sufficient colors). By applying the proposed compression scheme, only 7664 bits would be required to transmit the tensor. This corresponds to a bit more than 38% of the original size.

3.3. Synthetic Datasets

A common way (Danon et al., 2005; Meila, 2007) of judging the performance of community detection algorithms is the construction of synthetic graphs in which a community structure is defined over nodes, and edges connect nodes from the same communities with a probability $1 - \mu$, where μ is a noise or mixing factor. This structure is sometimes described as a (relaxed, for $\mu > 0$) caveman graph (Watts, 1999; Schaeffer, 2007). The ability of algorithms to reconstruct the original clustering is then used as a performance indicator. Here, generalizations suitable for k, k-hypergraphs are proposed. Evaluation will be performed by summing the previously introduced NMI measure over all domains.

In non-partite graphs, edges connect either nodes from the same or from different communities. The most direct generalization to k, k-hypergraphs would imply that for each community in a domain, there exists exactly one other community in each other domain such that edges only connect elements from those communities. More generally however, one might only require that all elements from one community are consistent with respect to which communities in other domains they are connected to, allowing for more complicated relations than the 1:1:1 relation described first. In the following, three families of synthetic hypergraphs are introduced, each more complicated than the previous one, starting with the simplest generalization.

3.3.1. SIMPLE

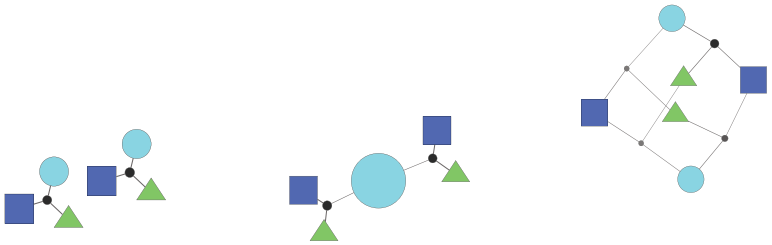
For the first, SIMPLE scenario, we define n communities with a fixed number of nodes for each domain. Each community in one domain is defined to *correspond* to exactly one community in the two other domains.

Edges are generated by picking a random node and then picking another node from all other domains, each with probability $\sqrt[k-1]{(1-\mu)}$ from the corresponding community. Like that, the overall probability of all elements of an edge coming from corresponding communities remains $\sqrt[k-1]{(1-\mu)}^{k-1} = 1 - \mu$.

Figure 3.3(a) illustrates this idea for two communities per domain: Edges, in the absence of noise, only exist between elements from either the first set of document, user, and tag communities, or the second one. Figure 3.4(a) shows a randomly generated graph from this family with 5 elements per community. Here, color indicates community membership. Elements from unambiguously corresponding communities, i.e. all documents, users and tags from one set of mutually corresponding communities are colored identically (although not technically from the same community since coming from different domains). Figure 3.5(a) shows the adjacency tensor of the same graph. Across all axes, elements are ordered by community membership. Each range on an axis has specific ranges on the other axes to which it has connections.

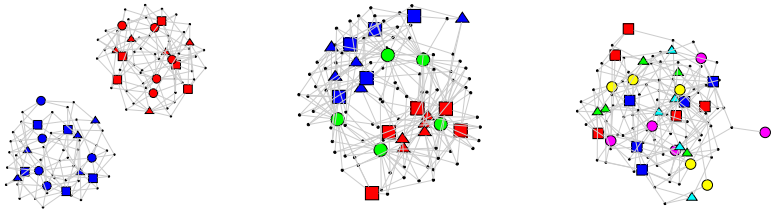
3.3.2. OVERLAPPING

A more interesting case is the presence of OVERLAPPING communities. Here, the simplest such model is examined: Two document and two user communities, but only one tag com-



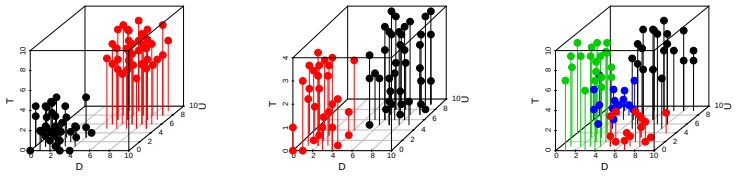
(a) SIMPLE (b) OVERLAPPING (c) CONTRADICTIVE

Figure 3.3.: Community structure of synthetic datasets. Edges are placed between communities connected by edges (black dots) with probability $1 - \mu$. In all figures, squares represent (communities of) documents, triangles are users, and circles tags.



(a) SIMPLE (b) OVERLAPPING (c) CONTRADICTIVE

Figure 3.4.: Graphs randomly generated from the corresponding families ($\mu = 0$ for all graphs). Elements from identical or unambiguously corresponding communities share the same color.



(a) SIMPLE (b) OVERLAPPING (c) CONTRADICTIVE

Figure 3.5.: Adjacency tensors of the graphs from Figure 3.4. Elements are ordered by community membership. Two edges have the same color exactly if they belong to the same community edge, i.e., the elements they connect are all members of the same communities. Visualization created with the R package ScatterPlot3D by Ligges and Mächler (2003).

munity exist. Imagine a community of tags around the topic of "memory" which may be used by e.g. computer scientists and psychologists to annotate distinct concepts, and different communities of documents. This shows why different numbers of communities for different domains might be necessary (challenge 1) and why it does not make sense to talk about objects from different domains being from the "same" community (challenge 2); there is no single e.g. document community to which the "memory"-related tags could be said to belong more than to the other. A community detection algorithm should be able to identify the two different sets of document and user communities without mistakenly separating the single tag community into two.

Each edge is created by picking a random document, picking a user from the corresponding community with probability $1 - \mu$ and always adding a tag from the single tag community.

Again, Figure 3.3(b) illustrates the basic community structure. Figure 3.4(b) shows a randomly generated graph. Documents and users form tightly connected groups – again, elements from corresponding sets of communities share the same color, but the tag community is located between the two. This situation is represented in Figure 3.5(b) by the fact that edges are clearly separated along two axes, but overlap on the third (y) axis.

"Overlapping" in this case means that a community in one domain may correspond to several communities in another domain. There is a body of work on multiple community assignments per node, e.g., (Lehmann et al., 2008) and (Lancichinetti et al., 2009), in which case communities are also said to overlap. This is, however, a different sense of the word and a topic that is not further investigated here.

3.3.3. CONTRADICTIVE

The final scenario assumes that each community is connected to each other community, but in a CONTRADICTIVE fashion. Imagine two groups of people A and B, two communities of tags (positive and negative), and two sets X and Y of, say, movies. This example shows a situation where people from group A tag all movies X positively, the other movies negatively, and the other group of people acts exactly in the oppositive way. This highlights the extension of correspondence noted in challenge 3: The question "Do A and X correspond?" needs to be replaced by the question "Do A, X and + correspond?"

Edges are created by a generalized process which could be also used to create the previous examples as well as more complicated scenarios. A set of L of legal community edges is defined – combinations of communities between whose elements edges are desired. For each edge, it is randomly decided whether it is supposed to be a "legal" edge (with probability $1 - \mu$) or a noisy one (otherwise). Then, nodes are randomly assembled to edges until the edge is, depending on its purpose, either a member of the legal community edges or not. To further illustrate the idea: the SIMPLE graphs would be constructed by defining $L = \{(0,0,0),(1,1,1)\}$, the OVERLAPPING graphs through $L = \{(0,0,0),(1,1,0)\}$. CONTRADICTIVE graphs, finally, are defined by the legal community edges $L = \{(1,1,1),(1,0,0),(0,1,0),(0,0,1)\}$.

Figure 3.3(c) again shows the general community structure. Each user community is connected to each document and each tag community, but when users from different commu-

nities tag documents from the same community, they use tags from different communities. We see six different colours used in the sample graph in Figure 3.4(c) as there are no sets of communities which could be said to correspond to each other across domains. The adjacency tensor in Figure 3.5(c) finally offers a different view on the same data. If we divide the axis in halves and label them 0 and 1, edges cluster in areas corresponding to the legal community edges.

3.3.4. Conclusion

Various extensions to the provided hypergraph models could be conceived – for example, communities could be of different sizes (Lancichinetti et al. (2008) examine this for non-partite cases), and graphs with a higher number of communities and more complicated relations could be made up. However, the goal of these models is rather to provide a *minimal* set of examples that highlight the challenges provided in Section 3.1.2. As will become more obvious later on, they require at least a one-, two- or three-dimensional representation of the data, respectively, and thus separate different classes of algorithms by their representational power. In the next chapter, modularity will be incrementally generalized to handle those challenges.

4. Multi-Partite Modularity

This chapter generalizes modularity to k, k-hypergraphs. It discusses a number of modularity measures which, building on top of each other, finally fulfill all the requirements for community detection on k, k-hypergraphs as laid out in the previous chapter:

Section 4.1: Non-Partite Modularity explains how the regular modularity measure for non-partite graphs can be applied to k, k-hypergraphs and why this approach is, in many cases, deficient.

Section 4.2: Coupled Bipartite Modularity introduces an approach based on the simultaneous optimization of the bipartite modularity of several bipartite graphs derived from the original hypergraph.

Section 4.3: Multi-Partite Modularity generalizes both bipartite modularity and the involved notion of correspondence to provide a family of truly multi-partite modularity measures.

Section 4.4: Hybrid Modularity Optimization deals with the conflict between coupled bipartite and multi-partite modularity measures which perform well in different situations. A joint optimization approach is proposed which combines the capabilities of both methods.

Section 4.5: Balanced Multi-Partite Modularity finally discusses an effect that often occurs with real graphs and leads to the formation of single giant communities. This effect is eliminated by balancing the correspondence function, leading to the final version of multi-partite modularity as proposed in this thesis.

Section 4.6: Discussion concludes this chapter.

The coupled bipartite modularity optimization is provided as a downloadable software package, `mpcd` (Multi-Partite Community Detection) – see Appendix A.1.3 for details.

Parts of this chapter have been published in Neubauer and Obermayer (2009b), Neubauer and Obermayer (2010) and Neubauer and Obermayer (2011).

4.1. Non-Partite Modularity

When a hypergraph is reduced to a simple graph, regular modularity optimization approaches can be applied. However, information is lost in this process. This section introduces a baseline algorithm based on such a reduction to test if the information loss is relevant for the task of community detection.

4.1.1. Definition and Optimization

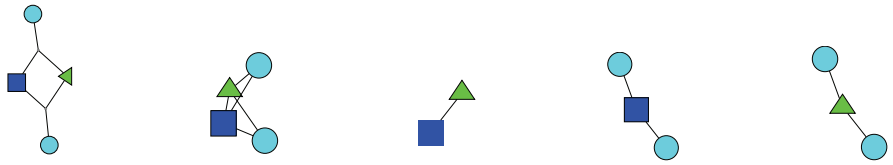
The hypergraph is "projected" (Figure 4.1(b)): for each edge (d, u, t), three new edges (d, u), (d, t), and (u, t) are introduced. If another edge, e.g., (d, u, t') exists, the weight of edge (d, u) is accordingly increased. Clauset et al. (2004)'s fast modularity optimization is applied on the resulting graph, and the community assignments obtained are used for the original nodes.

For example, d_1, u_1 and t_1 might be mapped to nodes 1,2 and 3 in the graph handed to the non-partite community detection algorithm. Assuming that the algorithms assigns these nodes all to a single community 1, d_1, u_1 and t_1 would be assigned to (now distinct) document, user and tag communities of id 1.

It should be noted that by this approach, none of the challenges defined in Section 3.1.2 are attacked – the original data is merely transformed such that original modularity becomes syntactically applicable. We will use this baseline algorithm to test whether these challenges actually become relevant in practice.

Interpreting the non-partite clustering A small complication in translating from the non-partite to the 3-partite graph occurs when we require the entire hierarchical clustering instead of just the final set of community assignments. Algorithm 2 shows how a tripartite hierarchical clustering can be obtained from the non-partite clustering in the face of merges between nodes which refer to elements from different domains.

Consider two documents d_1,d_2 and two users u_1,u_2 (let us restrict ourselve to two domains for this example). These are mapped onto nodes 1,2,3, and 4, respectively, before passing on the graph to the non-partite modularity optimization. Suppose the resulting clus-



(a) original (b) projected (c) H_{DU} (d) H_{DT} (e) H_{UT}

Figure 4.1.: Different ways of reducing a $3, 3$-hypergraph composed of elements from the domains D, U, and T (squares, triangles and circles). Projection (b) replaces hyperedges by binary edges between the incident nodes. A decomposition into bipartite graphs yields three new graphs (c)-(e).

tering tree is $(3 \to 1, 2 \to 1, 4 \to 1)$. The problem here is that node 3 (u_1) is merged into node 1 (d_1), i.e. two elements from different domains are merged, while we are interested only in merges between elements from the same domain. By the time 1 is merged with 4 (u_2), we need to remember that this community, although a document community, "carries" a user community as well.

In this example, the algorithm would be called with parameters $C = ((3,1),(2,1),(4,1))$ for the obtained clustering, and a $Map = ((1 : d_1, 2 : d_2), (3 : u_1, 4 : u_2))$ containing the mapping between non-partite and original node ids.

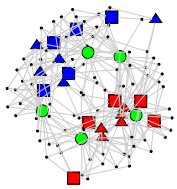
- When the first join $3 \to 1$ is processed, $Carry[2][1]$ is set to 3, indicating community 1 now carries element 3 from dimension 2.
- When the second join $2 \to 1$ is processed, two elements from the same dimension are merged, resulting in the output (d_2, d_1).
- When the third join $4 \to 1$ is processed, this is treated as $3 \to Carry[2][1] = 4$, resulting in the output (u_2, u_1).

The result is the desired set of clustering trees of depth 2: $((d_2, d_1),(u_2, u_1))$.

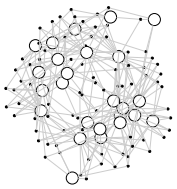
4.1.2. Explanation of Evaluation Charts

Figure 4.3 shows the performance of the non-partite approach on the three synthetic datasets, along with the coupled bipartite approach to be introduced in the next section for better comparison. Since this type of chart will occur several times throughout this chapter, the following section will first describe what exactly is shown in these figures before the actual results are discussed.

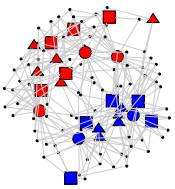
Each row shows the performance on one family of synthetic datasets. In all experiments, graphs were created with 10 nodes per community for the SIMPLE and OVERLAPPING families and with 5 nodes per community for the CONTRADICTIVE family. Per row, the performance on three different edge densities (difficulties) is shown. The individual charts



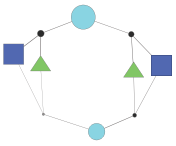
(a) OVERLAPPING sample graph with correct community assignments



(b) OVERLAPPING sample graph without domain information



(c) Community assignments by non-partite method



(d) Wrongly reconstructed community structure

Figure 4.2.: The non-partite method wrongly splits up the single tag community in an OVERLAPPING graph.

Algorithm 2 Converting a non-partite into a tripartite hierarchical clustering

function RECOVERTRIPARTITE($C,\ Map$)
▷ $C = [(v_{(\text{source},1)}, v_{(\text{target},1)}), (v_{(\text{source},2)}, v_{(\text{target},2)}), \ldots]$
▷ $Map = [\{v_i : d_1, v_j : d_2, \ldots\}, \{v_k : u_1, \ldots\}, \{v_l : t_1, \ldots\}]$

$Carry \leftarrow [\{\}, \{\}, \{\}]$ ▷ $Carry[d][v_1] = v_2 \Leftrightarrow v_2$ was merged into v_1, but
▷ v_2 refers to a different domain (d) than v_1
$C3 \leftarrow []$ ▷ will hold result

for $(s, t) \in C$ **do** ▷ s(ource) is merged into t(arget)
$dim_{\text{source}} \leftarrow d \in \{1, 2, 3\} : s \in Map[dim]$
$dim_{\text{target}} \leftarrow d \in \{1, 2, 3\} : t \in Map[dim]$
for $d \in \{1, 2, 3\}$ **do** ▷ over all domains
$s' \leftarrow -1$
$t' \leftarrow -1$
if $dim_{\text{source}} = d$ **then** ▷ Does s refer to an element from d?
$s' \leftarrow s$
else if $s \in Carry[d]$ **then** ▷ or carry an element from d?
$s' \leftarrow Carry[d][s]$
end if
if $dim_{\text{target}} = d$ **then**
$t' \leftarrow t$
else if $t \in Carry[d]$ **then**
$t' \leftarrow Carry[d][t]$
end if

if $s' \neq -1 \wedge t' \neq -1$ **then** ▷ s and t refer to elements from d
$s_{\text{real}} \leftarrow Map[d][s']$ ▷ obtain IDs from hypergraph
$t_{\text{real}} \leftarrow Map[d][t']$
$C3 += (d, s_{\text{real}}, t_{\text{real}})$ ▷ store that in domain d, $s_{\text{real}} \rightarrow t_{\text{real}}$
if $t' \in Carry[d]$ **then**
del($Carry[d][t']$) ▷ Carried element has been resolved
end if
else if $s' \neq -1 \wedge t' = -1$ **then** ▷ s refers to element from d, but t does not
$Carry[d][t] \leftarrow s'$ ▷ make a note for t that it carries an unmerged s
end if
end for
end for
return C3 ▷ $= [(d_1, s_1, t_1),\ (d_2, s_2, t_2), \ldots]$

end function

indicate performance (NMI) by a mixing factor μ. Error bars indicate standard deviation over 100 randomly created graphs. Methods keep "their" color across different evaluation charts. The evaluation charts were created using R (R Development Core Team, 2011).

The variance of the NMI values in many cases is rather large. In order to differentiate the variance that comes from the differences between the individual graphs and that which stems from actual performance difference between the methods, an additional statistical test is performed – a so-called sign test. Instead of regarding the absolute performance values, the better algorithm for each individual graph is determined. This results in a "score" of $n{:}(100-n)$. A binomial test is performed against the null-hypothesis that both algorithms perform similarly well, i.e. that the chance of one algorithm outperforming the other on a single graph equals 50%. Hollow symbols indicate non-significant differences (p>0.05, meaning superior performance of one method in at least 61 out of 100 examples). It turns out in most cases that differences between algorithms are in fact significant even when absolute variance is high, confirming the hypothesis that this variance is rather due to differences between sample graphs.

4.1.3. Results

The non-partite approach performs well on the SIMPLE family of graphs: Increasing the number of edge-to-node ratio improves results while adding noise by increasing μ decreases performance, a pattern which will be visible across all datasets and algorithms. The approach however completely fails on the OVERLAPPING graphs, and, consequently, on the even more complicated CONTRADICTIVE dataset as well.

Let us examine the case of OVERLAPPING graphs in more detail. Figure 4.2(a) shows a graph from that family with 3 edges per node, 10 nodes per community and $\mu = 0$ – as previously shown in Figure 3.4(b). Figures 4.2(c) and 4.2(d) show the solution as returned by the non-partite method: The tags, originally in a single community (green in 4.2(a)), are assigned to one of two tag communities, depending on which document/user community they are slightly more strongly connected to. This is, however, the obvious solution for the non-partite method to return when we consider the structure of the graph without domain information (as "seen" by the method), shown in Figure 4.2(b). By letting go of the domain information, the non-partite approach has no reason to assume the tags (not recognizable as such) should not be split into two communities, and correctly simply splits the graph in two.

An obvious objection to a multi-partite modularity measure might be: Why not just project the data down onto a normal graph? The OVERLAPPING graphs are one (in my opinion the simplest possible) case in which this does not work. It demonstrates the practical implications of the first two challenges presented in Section 3.1.2: A method to reconstruct the intended community structure will need to support distinct community assignments for distinct domains – Challenge 1 – (and be able to process domain information in the first place) and furthermore support ambiguous correspondence relations between communities from different domains – Challenge 2. The next subsection introduces a method which supports this features.

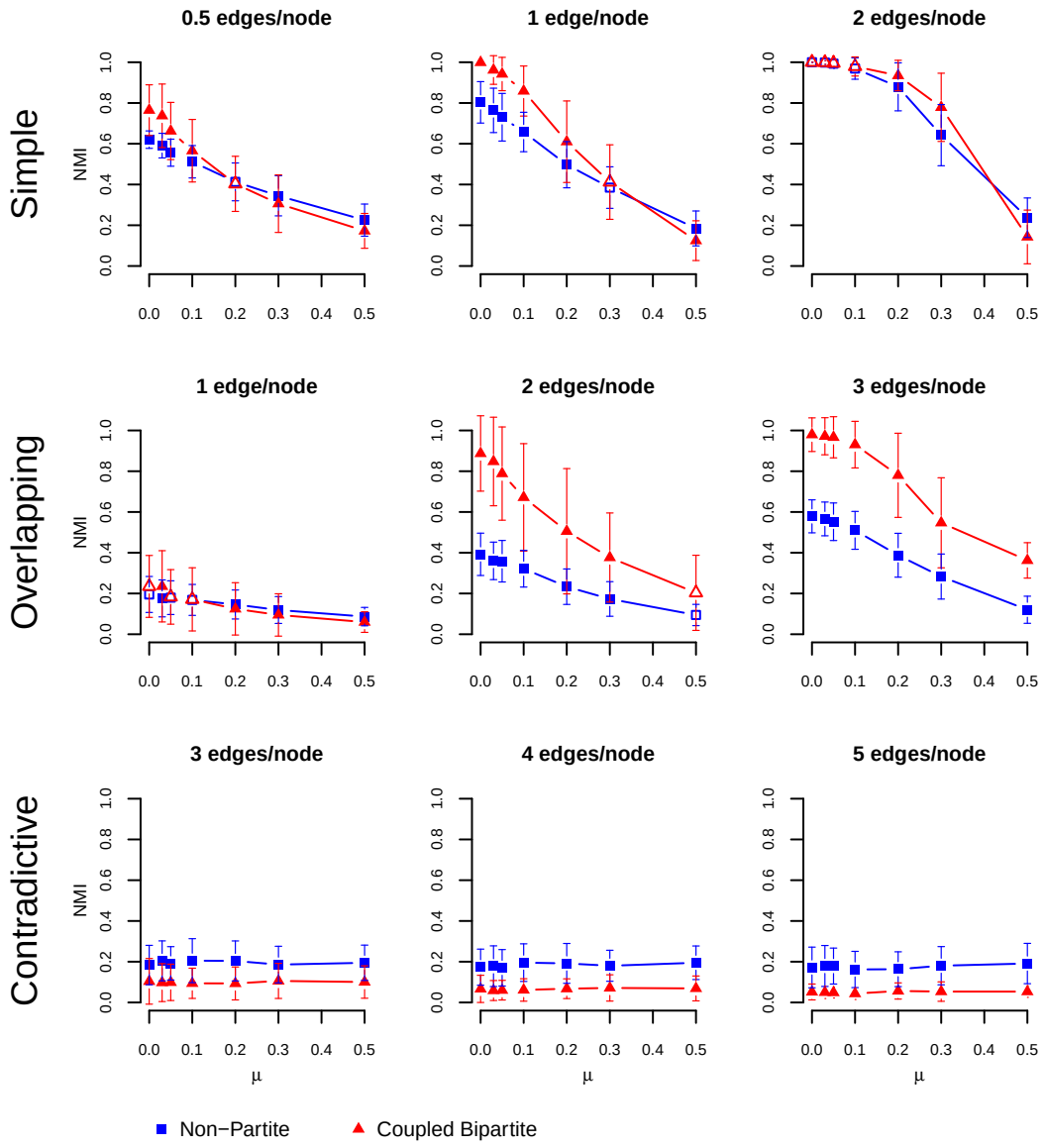


Figure 4.3.: Comparing the non-partite algorithm to the coupled bipartite approach: The non-partite approach performs well in the SIMPLE case, but fails both in the OVERLAPPING and CONTRADICTIVE situation. The coupled bipartite approach however succeeds in the OVERLAPPING case. See Section 4.1.2 for a detailed explanation of this chart.

4.2. Coupled Bipartite Modularity

In the previous section, regular community detection methods were made applicable to k, k-hypergraphs, at the cost of losing crucial structural information. Here, we examine a reduction to several bipartite graphs such that modularity measures adapted for bipartite graphs can be applied while retaining more of the original hypergraph's structure. After reviewing bipartite modularity measures, this section introduces the so-called coupled bipartite modularity and an optimization algorithm, then discusses its performance on the synthetic datasets.

4.2.1. Bipartite Modularity

Bipartite graphs are instances of k, k-hypergraphs with k=2. Therefore, they are also affected by the first two of the challenges mentioned above. Since 2007, at least three different generalizations of modularity have been proposed for handling them. We will discuss these measures in some detail – closely following the argumentation in (Murata, 2009) – as they play a crucial role for our proposed algorithm.

Barber (2007)'s modularity addresses the issue of connectivity by adapting the null model such that $P_{ij} = 0$ for nodes i, j from the same domain. This keeps the proposed modularity measure from punishing the non-adjacency of elements from the same domain. However, the model still assumes a shared community structure in both domains, not addressing the issue of different communities.

Guimera's modularity (Guimerà et al., 2007) only considers the community structure of one domain. It rewards nodes from the same community being connected to the same nodes from the other domain. Declaring the community structure of the second domain as irrelevant resolves both issues. However, in many cases, the community structure of both domains is of interest. Here, the modularity measure might be applied in both directions, which however is not an integrated solution and might complicate direct optimization.

Definition 4.1 (Murata's modularity) *For a given graph $G = (V, E)$ with M edges, an adjacency matrix A and a community assignment $\boldsymbol{\sigma}$, Murata (2009)'s bipartite modularity Q_{Mu} is defined as*

$$Q_{\mathrm{Mu}}(\sigma) = \sum_l (e_{lm} - a_l a_m),\ m = \operatorname*{argmax}_m(e_{lm}),$$

where

$$e_{lm} = \frac{1}{2M} \sum_{i \in V:\sigma(i)=l} \sum_{j \in V:\sigma(j)=m} A(i,j),\ \text{and}\ a_l = \sum_m e_{lm}.$$

For every community l from one domain, this identifies that community m from the other domain which l shares the most edges with (m being from the other domain is not explicitly stated, because the number of shared edges with communities from the same domain, by definition, is 0, so none of them will ever be the result of the argmax function). Let us call m the "corresponding community" of l. Q_{Mu} rewards a high number of edges shared

between these communities (e_{lm}) minus the expected number of shared edges according to the configuration model, $a_l a_m$. Rephrased in terms of community edges, (l, m) indexes that community edge incident to l that has the highest relative importance for l, and Q_{Mu} rewards (l, m) being of high density, normalized by a null model.

Since this modularity measure is the only of the three which explictly supports distinct communities for distinct domains, it is the best choice as the basis for the coupled bipartite modularity (a recent generalization by Suzuki and Wakita (2009) will be discussed later on in the context of further generalizations).

4.2.2. Definition

Coupled bipartite modularity is based on the decomposition of a 3,3-(or k, k) hypergraph onto 3 (or $k(k-1)/2$) bipartite graphs. As illustrated in Figure 4.1(c) - 4.1(e), each edge (d, u, t) of a hypergraph H results in an edge (d, u) in a first graph called H_{DU}, and an edge (d, t)/(u, t) in the two next graphs H_{DT} and H_{UT}. Multiple occurrences of a pair like (d, t) lead to a more strongly weighted new edge. We call those graphs the *bipartite projections* of H:

Definition 4.2 (Bipartite Projection) *Let i and j be indices of two distinct domains of a k, k-hypergraph $H = (V, E)$, i.e. values from K or names like D,U,T. For a hyperedge $e \in E$, let e_i be the element indexed by i. For two nodes v_i and v_j from the corresponding domains, the set*

$$E_{\mathrm{proj}}(v_i, v_j) = e \in E : e_i = v_i \wedge e_j = v_j$$

thus contains all hyperedges which connect v_i and v_j. The bipartite projection $H_{i,j}$ of H is then defined by the edges

$$E_{i,j} = \{(v_i, v_j) : E_{\mathrm{proj}}(v_i, v_j) \neq \emptyset\}$$

and the weight of each edge $(v_i, v_j) = |E_{\mathrm{proj}}(v_i, v_j)|$.

Definition 4.3 (Coupled Bipartite Modularity) *Let the* coupled bipartite modularity *of a 3,3-hypergraph H with bipartite projections H_{DU}, H_{DT}, and H_{UT} be the mean modularity of the bipartite projections under a community assignment σ:*

$$Q_{\mathrm{CB}}(\sigma) = \frac{1}{3} Q_{\mathrm{Mu}}(H_{DU}, \sigma) + \frac{1}{3} Q_{\mathrm{Mu}}(H_{DT}, \sigma) + \frac{1}{3} Q_{\mathrm{Mu}}(H_{UT}, \sigma)$$

or, more generally, for k, k-hypergraphs with bipartite projections $H_{i,j}$:

$$Q_{\mathrm{CB}}(\sigma) = \frac{2}{k(k-1)} \sum_{i \in K} \sum_{j \in K, j > i} Q_{\mathrm{Mu}}(H_{i,j}, \sigma)$$

So, the task of multi-partite community detection is reduced to several applications of Muarata's bipartite modularity. Each domain has its independent set of community assignments, and the joint quality of the assignments is computed as the average modularity they produce in the bipartite relations. Like this, information about the domain membership

Algorithm 3 The general greedy bottom-up modularity optimization algorithm

function OPTIMIZE(E,V,Q)

▷ E contains the edges of the hypergraph to be clustered
▷ $V = (V_1, \ldots, V_k)$ contains nodes of the hypergraph, by domain
▷ Q is the modularity function to optimize

$\boldsymbol{\sigma}^0 \leftarrow$ unique community ids for every node
$\Delta Q_d(s,t) \leftarrow G(\boldsymbol{\sigma}^0, \varphi = (d,s,t)) \quad \forall d \in K,\ (s,t) \in V_d \times V_d$
for $\tau = 1 \ldots |V| - k$ **do** ▷ $|V_d - 1|$ binary joins in every domain d
  $\varphi \leftarrow \operatorname{argmax}_{\varphi'=(d,s,t)}(G(\boldsymbol{\sigma}^{\tau-1}, \varphi')) = \operatorname{argmax}_{\varphi'=(d,s,t)}(\Delta Q_d(s,t))$
  $\boldsymbol{\sigma}^\tau \leftarrow \boldsymbol{\sigma}^{\tau-1} \circ \varphi$
  $\boldsymbol{\varphi}_{\text{dirty}} \leftarrow$ NEEDUPDATE$_Q(\boldsymbol{\sigma}^{\tau-1}, \varphi)$
  $\Delta Q_{d'}(s',t') \leftarrow G(\boldsymbol{\sigma}^\tau, \varphi') \quad \forall \varphi' = (d',s',t') \in \boldsymbol{\varphi}_{\text{dirty}}$
end for
return $\Sigma = (\boldsymbol{\sigma}^0, \ldots, \boldsymbol{\sigma}^{|V|-k})$
end function

of the individual nodes is kept. Although the third of the three challenges – hyper-incidence (see Section 3.1.2) – is handled by reduction such that information loss still occurs, keeping domain membership information and supporting distinct community models take care of the first two challenges.

4.2.3. General Bottom-Up Optimization of Modularity

Before discussing the specifics of optimizing coupled bipartite modularity, I will introduce the general optimization approach, i.e. how k-partite hierarchical clusterings are generated, for all modularity measures proposed from now on, unless noted otherwise.

Definition 4.4 (Modularity Gain) *Let $G(\boldsymbol{\sigma}, \varphi)$ be the change in modularity obtained from applying join φ to $\boldsymbol{\sigma}$:*

$$G(\boldsymbol{\sigma}, \varphi) = Q(\boldsymbol{\sigma} \circ \varphi) - Q(\boldsymbol{\sigma}).$$

Optimization works in a greedy bottom-up fashion. Each node starts as a single community ($\boldsymbol{\sigma}^0$). At each step, those two communities are merged which, across all domains, result in the highest gain G in modularity, leading to a new $\boldsymbol{\sigma}^\tau$. In analogy to (Clauset et al., 2004), the actual optimization is performed by keeping one community $\times$ community table ΔQ_d per domain d, storing the potential modularity gain obtained by joining two communities and, after a join, updating those entries of ΔQ which may have been affected by the join, as computed by a function NEEDUPDATE. Finally, the k-partite hierarchical clustering Σ defined by $(\boldsymbol{\sigma}^0, \ldots, \boldsymbol{\sigma}^{|V|-k})$ is returned. See Algorithm 3 for a description in pseudo-code.

The NEEDUPDATE function plays a crucial role for the performance of the algorithm. In order for the algorithm to be correct, it must at least return all joins φ (i.e. community pairs with domain index) for which

$$G(\boldsymbol{\sigma}^{\tau-1}, \varphi) \neq G(\boldsymbol{\sigma}^\tau, \varphi).$$

If it marks all pairs for updating, the algorithm is simplified to a fully brute-force approach. However, depending on the particular modularity, significant speed-ups can be achieved by efficiently predicting which pairs need updating.

Another aspect specific to the particular modularity function is how the actual modularity gain of a join is computed. Again, one might choose the brute-force approach and simply call the modularity function for the new community assignment. Knowledge about the behavior of specific modularity functions has been incorporated in the actual implementations instead.

Finally: Several optimisations have been left out of the algorithm description. For example, modularity gain is symmetric, since information about which community is merged into which is only relevant for syntactic reasons, not for modularity computation – so ΔQ is symmetric as well and redundant entries need not be computed nor stored twice. Keeping track of and returning the index τ_{optimal} of the best community assignment is another example of low-level optimizations that has been left out of the pseudo-code.

4.2.4. Optimization for the Coupled Bipartite Modularity

4.2.4.1. Minimizing the output of NEEDUPDATE

Given two communities s, t in a domain d that have just been joint, an upper bound for the set of pairs for which ΔQ needs to be updated is the set of all pairs containing a community out of one of the following three sets:

- S_1: the communities s, t being joint,
- S_2: any community l in a domain $d' \neq d$ with connections to S_2 (as s, t, or the new $s + t$ may be or become l's corresponding community), or
- S_3: any community o in d with connections to any $l \in S_1$ (because the merge of s+t may have changed whether o joining another community becomes or stays l's corresponding community).

Depending on the network's connectivity, set S_3 may grow rather large, requiring large portions in ΔQ_d to be recomputed. See Section 4.3.3 for a discussion on how taking into account the nodes in set S_3 is only required due to the non-local properties of argmax and can be abandoned once it is replaced by simple linear function.

4.2.4.2. Computing Modularity Gain

The actual computation of the modularity gain is optimized by keeping track of which communities do or could correspond to each other and only testing for changes in these relations. The exact procedure is rather intricate and omitted here both because it does not significantly contribute to the main line of argument and because it would mean turning a significant amount of Java into pseudo-code.

4.2.4.3. Choosing the Best Community Assignment

Once all elements have been merged into a single community, the community assignments obtained during the process form a hierarchical clustering, and the best individual community assignment needs to be selected if a definite assignment is required. By default, this would be the assignment with the highest associated modularity value. However, there may be reasons to choose other community assignments. For example, Murata's modularity has the property of returning 0.5 if all elements in a domain are assigned to the same cluster: If vertices from domain 1 and 2 all belong to community 1 and 2, respectively,

$$e_{1,2} = e_{2,1} = a_1 = a_2 = \frac{1}{2M} M = 0.5, \text{ and}$$

$$Q_{\mathrm{Mu}} = e_{1,2} - a_1 a_2 + e_{2,1} - a_2 a_1 = 0.5.$$

In the presence of some noise, this may be a value unsurpassed by any of the non-trivial solutions. This behaviour may be fixed by replacing, in the definition of e_{lm}, the term $\frac{1}{2M}$ by $\frac{1}{M}$, as could be argued to be more appropriate. This however affects the measure's overall performance in other ways. So instead, the trivial solution is simply ignored and the second-best one is chosen, greatly improving results.

Another possibility is to use another function than modularity altogether for the selection of the final assignment. Empirically, we have found that choosing the solution with the best compression as defined in Section 3.2.3.2 tends to find slightly better spots, evading both overly simplistic and overly complex solutions (i.e. assignments too high or too low in the clustering tree).

4.2.5. Related Work

Various other approaches for bipartite community detection exist that either modify modularity more strongly than the measures introduced in this section, e.g. (Ghosh and Lerman, 2009), or work in a completely different fashion, e.g. the multi-level approach by Liu and Murata (2009) or the bipartite cut-based community detection by Kumar et al. (2008). Since the following work relies on further generalizations of modularity, these approaches have not been taken into account, but might well be interesting candidates for coupled optimization as well.

Bekkerman et al. (2005) provide an earlier example for the general strategy of reducing multi-way interactions to several binary relations. Hartsperger et al. (2010) describe an approach based on Nonnegative Matrix Factorization that simultaneously approximates the adjacency matrices of several bipartite graphs, which however stem directly from a k-partite (non-hyper-) graph.

Closest to the coupled bipartite modularity is however the work by Lu et al. (2009) who perform a so-called tripartite clustering on social bookmarking data. It derives individual cluster assignments for documents, users, and tags, and it also works by reducing the tripartite structure to bipartite relations. The difference to the coupled bipartite modularity is that it uses a centroid-based approach. A centroid for a domain d represents the average connectivity of all cluster members, with one average vector per domain $d \neq d'$. The authors

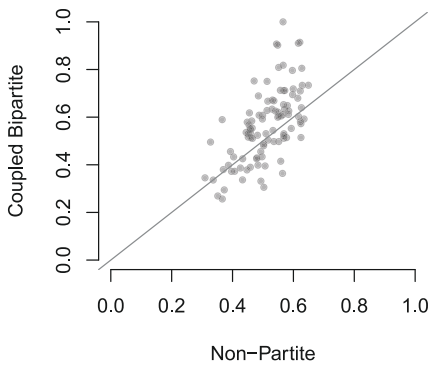


Figure 4.4.: Individual NMI scores for the non-partite and coupled bipartite approach, for each of the 100 graphs of the SIMPLE family with 0.5 edges/node and $\mu = 0.1$. The two data points at 0.1 in the top left chart in Figure 4.3 correspond to the means of these values.

then propose an algorithm which, for a given number of clusters per domain, minimizes the difference between the centroids and its members' actual connectivity. This approach was also evaluated on the synthetic datasets and the results will be discussed in the following.

4.2.6. Results

4.2.6.1. Coupled Bipartite Modularity

Figure 4.3 shows the performance of the coupled bipartite modularity. First of all, it excels on the SIMPLE graphs, outperforming the non-partite modularity in many cases. To shortly come back to the issue of significance: We find some of the aforementioned cases in which average performance values seem to be very close, particularly when taking into account the standard deviation as indicated by the error bars. Figure 4.4 shows the individual values behind the two means for one case in which the two means seem to be particularly close: $\mu = 0.1$ in the top left chart, i.e. the one regarding the sparsest SIMPLE graphs. For each graph, a single point is plotted with the non-partite method's performance as its x position and the coupled bipartite performance on the y position. We find that the coupled bipartite modularity not only reaches higher absolute values in some cases, but more importantly that the majority of the points lie above the diagonal, indicating a higher performance by the coupled bipartite modularity for that particularly graph. In fact, 64 out of 100 datapoints lie above the diagonal, a bit above the 61 required to turn the probability of equal performance given the data below 5%.

The method also performs well in the OVERLAPPING case: The domain information and the support for distinct community structures allow for a perfect reconstruction in many graphs. Keeping domain information, it can find that the best community assignment is for

all tags to remain in a single community. All document and user communities are now fully connected to "their" (the only) corresponding tag community, leading to high modularity values for the corresponding bipartite relations.

However, the information it receives in the CONTRADICTIVE case is too limited to allow for any sensible reconstruction: The bipartite connections indicate connections exist between any two communities – users from both communities have tagged documents from both communities and have used tags from both communities, just as tags from both communities have been assigned to documents from both communities. Therefore, the bipartite connections alone contain no information to tell apart different communities. This underlines the relevance of the third challenge (hyper-incidence).

4.2.6.2. Lu's Tripartite Clustering

Lu's algorithm requires the number of clusters as a parameter, as opposed to the coupled bipartite modularity – the ability to find the optimal number and size of clusters automatically is in fact one of the outstanding features of modularity. This poses the question of how to compare the algorithms fairly. Two different approaches are taken: First, three clusters per domain (i.e. too many) are given to the algorithm to test if may leave unrequired clusters empty. In the other scenario, the correct number of clusters are given to the algorithm – $2 \times 1 \times 2$ clusters for the OVERLAPPING dataset and $2 \times 2 \times 2$ for the others. Heuristics can be conceived to find the correct number of communities after generating results for various settings, so this approach might be better suited for comparison. The following results can be observed:

- Figure 4.5 shows the results for the first setting. Basically, no sensible results can be obtained without setting the right number of clusters.
- Figure 4.6 shows the results when providing the correct number of clusters. Still, the only time the tripartite clustering outperforms the coupled bipartite modularity is in the very sparse OVERLAPPING case where, without knowledge about the right amount of clusters, the modularity-based approach assigns all elements to a single cluster.
- Apart from the absolute performance values, the curves do follow the same basic shape. Both algorithms react positively to more edges and negatively to noise, and both fail on the CONTRADICTIVE dataset.

When comparing the performance of the two algorithms, one has to keep in mind that the coupled bipartite modularity was designed and optimized with the synthetic test cases in mind, whereas the tripartite clustering was applied out of the box using the code Caimei Lu was kind enough to send me. The algorithm also provides several hyperparameters which might be tuned to get better results. More relevant therefore is the fact that both methods show the same behaviour in general and, in particular, fail on the CONTRADICTIVE test graphs. This further strengthens the point that this is an inherent limit when considering only bipartite relations and further motivates the formulation of a multi-partite method, which will be developed in the next section.

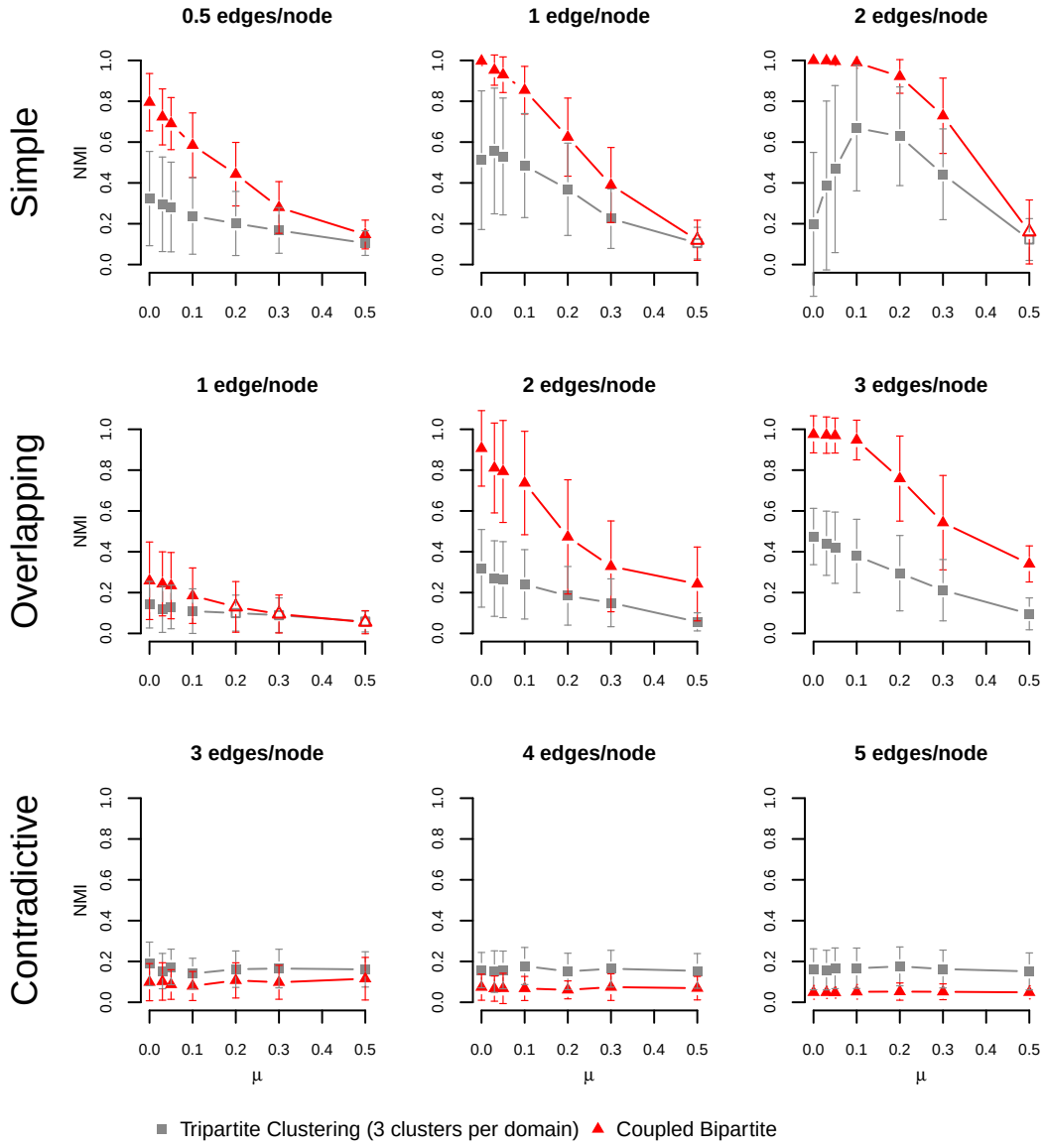


Figure 4.5.: Comparing the coupled bipartite modularity with Lu's tripartite clustering. For each graph, the number of clusters was set to 3, i.e. too high. The tripartite clustering, under these circumstances, almost never produces sensible results. See Section 4.1.2 for a detailed explanation of this chart.

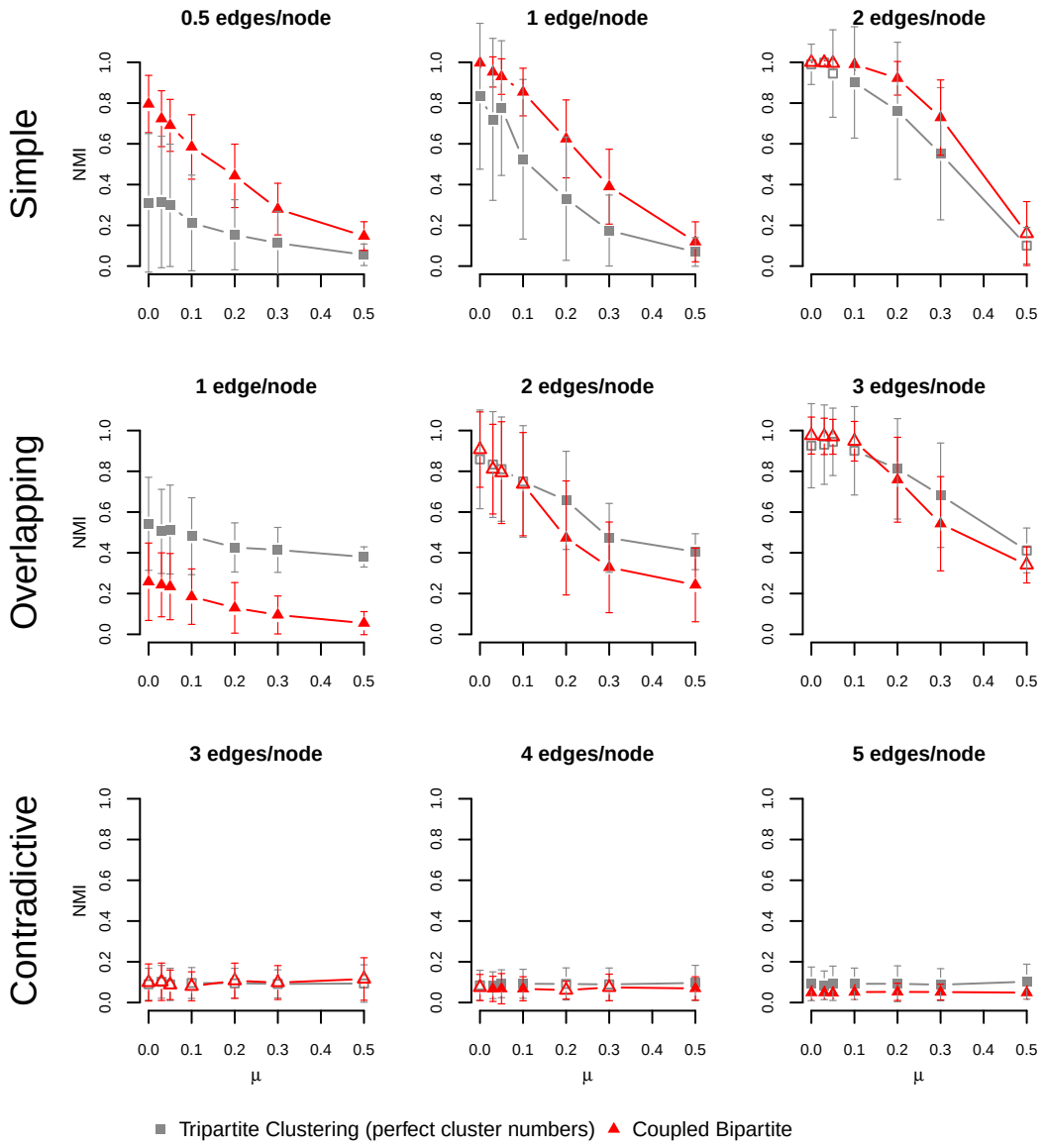


Figure 4.6.: Comparing the coupled bipartite modularity with Lu's tripartite clustering. The perfect number of clusters was given to Lu's algorithm. Still, it performs worse than coupled bipartite modularity except in few cases. See Section 4.1.2 for a detailed explanation of this chart.

4.3. Multi-Partite Modularity

In this section, a natively multi-partite modularity measure is proposed. After demonstrating how previous work leads to a natural generalization of the basic modularity formula, a second generalization allows the additional choice of arbitrary correspondence functions. An optimization algorithm is proposed and compared the the previous one and a brute-force approach in terms of runtime complexity. Then, its performance on the synthetic datasets is reviewed.

4.3.1. Definition

First of all, as stated by Clauset et al. (2004), modularity Q can be rewritten as

$$Q(\sigma) = \sum_{i \in C} (e_{ii} - a_i^2)\,,$$

where C is the codomain of σ, i.e. the set of communities,

$$e_{ij} = \frac{1}{2M} \sum_{v,w \in V : \sigma(v)=i \wedge \sigma(w)=j} A(v,w)$$

for an adjacency matrix A, and

$$a_i = \frac{1}{2M} \sum_{v \in V : \sigma(v)=i} k_v.$$

Instead of summing over all individual pairs of nodes, and only counting those assigned to the same community, this directly computes the aggregate actual (e) and expected (a) adjacency of each community, then sums over all communities. We can further rewrite this as

$$Q(\sigma) = \sum_{(i,j) \in C \times C} (e_{ij} - a_i a_j) f(i,j)\,,$$

where

$$f(i,j) = \delta(i,j).$$

This shows how identity is just one possible choice for the correspondence between communities. Murata's bipartite modularity Q_{Mu}, given by

$$Q_{\mathrm{Mu}}(\sigma) = \sum_{i \in C} (e_{ij} - a_i a_j), j = \operatorname*{argmax}_k (e_{ik})$$

can rewritten analogously as

$$Q_{\mathrm{Mu}}(\sigma) = \sum_{(i,j) \in C \times C} (e_{ij} - a_i a_j) f(i,j),$$

where

$$f(i,j) = \delta(j, \operatorname*{argmax}_k (e_{ik})).$$

So identity in Q is replaced by an argmax in Q_{Mu}. We can rewrite Q_{Mu} one step further as

$$Q_{\mathrm{Mu}}(\sigma) = \sum_{(i,j)\in C_1\times C_2} (e_{ij} - a_i a_j)(f_1(i,j) + f_2(i,j))\,,$$

where

$$f_1(i,j) = f(i,j) \text{ and } f_2(i,j) = f(j,i)$$

and C_1 and C_2 are the communities from the two different domains. Since elements from different communities in the same domain, by definition, never share edges, it follows that communities in the same domain never correspond to each other, such that the exclusion of pairs from $C_1 \times C_1$ and $C_2 \times C_2$ does not change the sum. Since each community occurs only as i or j, the correspondence function needs to be applied in both directions.

Using this formulation, the generalization to three domains becomes clear:

Definition 4.5 (Multi-Partite Modularity) *Let the* multi-partite modularity *of a 3,3-hypergraph H under a community assignment $\boldsymbol{\sigma}$ and a correspondence function f be defined as*

$$Q_{\mathrm{MP}}(\sigma) = \sum_{(i,j,k)\in C_1\times C_2\times C_3} (e_{ijk} - a_i a_j a_k)\big(f_1(i,j,k) + f_2(i,j,k) + f_3(i,j,k)\big),$$

where

$$e_{lmn} = \frac{1}{M} \sum_{i\in V:\sigma(i)=l} \sum_{j\in V:\sigma(j)=m} \sum_{k\in V:\sigma(k)=n} A(i,j,k)$$

and

$$a_i = \sum_{j,l} e_{i,j,l},\ a_j = \sum_{i,l} e_{i,j,l},\ a_l = \sum_{i,j} e_{i,j,l}.$$

or, more generally, for k, k-hypergraphs :

$$Q_{\mathrm{MP}}(\sigma) = \sum_{(i_1,\ldots,i_k)\in C_1\times\ldots\times C_k} \left(\left(e_{i_1,\ldots,i_k} - \prod_{j\in\{i_1,\ldots,i_k\}} a_j \right) \sum_{d=1}^{k} f_d(i_1,\ldots,i_k) \right).$$

Where necessary, the particular choice of correspondence function will be denoted after "MP", as in $Q_{\mathrm{MP|lin}}$.

The previous definitions are formulated to relate to the previous literature on mainly non-partite modularity. In the terminology of community edges introduced in Section 3.2.2, sums iterate over the community space $\mathbf{C}_{\boldsymbol{\sigma}}$, each index (i,j,k) refers to a community edge, and $e_{i,j,k} = \frac{1}{M}|(i,j,k)_\sigma|$. This allows a convenient reformulation for the general case as

$$Q_{\mathrm{MP}}(\boldsymbol{\sigma}) = \sum_{\mathbf{c}_{\boldsymbol{\sigma}}\in\mathbf{C}_{\boldsymbol{\sigma}}} \left(\left(\frac{|\mathbf{c}_{\boldsymbol{\sigma}}|}{M} - \prod_{d=1}^{k} \frac{a_{d,c_d}(\boldsymbol{\sigma})}{M} \right) \sum_{d=1}^{k} f_d(\mathbf{c}_{\boldsymbol{\sigma}}) \right).$$

4.3.2. Correspondence functions

What remains is the choice of the correspondence function f. $f_d(\mathbf{c}_{\boldsymbol{\sigma}})$, to summarize, describes how strongly the dth component of $\mathbf{c}_\sigma$, c_d, corresponds to the communities incident to $\mathbf{c}_\sigma$. As an abstraction of the correspondence function implicit in the original modularity's definition, its purpose is to provide a multi-partite and potentially continuous replacement.

In the following, general requirements for correspondence functions will be defined that follow from this. Then, two correspondence functions will be introduced that fulfill these requirements in different ways.

4.3.2.1. Requirements for correspondence functions

The following requirements make explicit properties of the originally employed identity function, $\delta(x, y)$ =1 iff $x = y$ and 0 otherwise, that should remain valid for its generalizations.

1. No correspondence without connectivity

   $$|\mathbf{c}_{\boldsymbol{\sigma}}| = 0 \Rightarrow f_d(\mathbf{c}_{\boldsymbol{\sigma}}) = 0$$

   If a set of communities do not share any edges, all mutual correspondences should be 0. This ensures that these sets do not contribute anything to the modularity and that the outer sum in fact loops over *non-empty* community edges only.

2. Full correspondence for full connectivity

   $$f_d(\mathbf{c}_{\boldsymbol{\sigma}}) = 1 \Leftrightarrow |\mathbf{c}_{\boldsymbol{\sigma}}| = a_{d,c_d}$$

   If a community c_d shares all its edges with a set of communities, its correspondence to this set should be 1. This in fact follows from requirements 1 and 3, but is noted here for clarity.

3. Correspondence per community must sum to 1

   $$\sum_{\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}} : c_d = i} f_d(\mathbf{c}_{\boldsymbol{\sigma}}) = 1$$

   The sum of all correspondence values between a given community i and other (sets of) communities should amount to 1, just as is the case with δ, where for a given community c, only (c, c) returns 1. This value may however be reached as the sum of various correpondence values < 1.

   As it turns out, bounding correspondence is the driving force behind the creation of communities. Imagine setting the correspondence to 1 for each community edge with a positive contribution to modularity. The ideal solution in this case is to keep every element in a single community: The community edges (which in this case are identical to the original set of edges) have an actual connectivity of $\frac{1}{M}$, whereas the expected connectivity remains minimal due to the small size of the involved "communities", i.e. the single nodes. Any join can only increase expected connectivity, decreasing overall modularity. While this is an extreme example, it highlights the importance of this final requirement.

4.3.2.2. Murata's Tripartite Modularity

Murata (2010a,b) has proposed a direct generalization of his approach to tripartite hypergraphs which can be seen as an instance of this formulation with

$$f_1(i,j,k) = \delta((j,k), \underset{(j',k')}{\operatorname{argmax}}(e_{ij'k'}))$$

and $f_2,\ f_3$ correspondingly.

This function fulfills the above requirements:

1. $|\mathbf{c}_{\boldsymbol{\sigma}} = (i,j,k)_{\boldsymbol{\sigma}}| = 0 \Rightarrow e_{ijk} = 0$. Since each community contains at least one node and each node is incident to at least one edge (since we do not consider nodes without edges as they trivially do not play a role for community detection), another (j',k') will be the result of argmax, assigning a correspondence of 0 to an empty $(i,j,k)_{\boldsymbol{\sigma}}$.
2. $|\mathbf{c}_{\boldsymbol{\sigma}} = (i,j,k)_{\boldsymbol{\sigma}}| = a_{1,i} \Leftrightarrow (j,k) = \operatorname{argmax}_{(j',k')}(e_{ij'k'}) \Leftrightarrow f_1(i,j,k) = 1$, i.e., a set of communities sharing all edges is automatically the result of argmax, resulting in a correspondence of 1.
3. For any community i, $f_1(i,j,k)$ is 1 exactly for 1 combination of (j,k) and 0 for any other.

4.3.2.3. Linear Multi-Partite Modularity

As proposed earlier by Suzuki and Wakita (2009) for bipartite cases, I further generalize correspondence to a real-valued measure, able to represent ambiguous relations, where a single community may correspond to several communities in a another domain at the same time:

$$f_1(i,j,l) = \frac{e_{i,j,l}}{a_i}, \quad f_2(i,j,l) = \frac{e_{i,j,l}}{a_j}, \quad f_3(i,j,l) = \frac{e_{i,j,l}}{a_l}$$

which, in terms of community edges, is the relative importance of $\boldsymbol{\sigma}$ for its dth component (see Def. 3.4):

$$f_d(\mathbf{c}_{\boldsymbol{\sigma}}) = r(\boldsymbol{\sigma}, d) = \frac{|\mathbf{c}_\sigma|}{a_{d,c_d}}.$$

If a document community i shares 4 of its 10 edges with a tag/user pair (j,l), and 6 edges with (j',l'), the correspondences provided by this function will be 0.4 and 0.6, respectively, instead of 0.0 and 1.0 as with the argmax-based approach.

This function also fulfills the requirements for a correspondence function:

1. $|\mathbf{c}_{\boldsymbol{\sigma}}| = 0 \Rightarrow f_d(\mathbf{c}_{\boldsymbol{\sigma}}) = \frac{0}{a_{d,c_d}} = 0$
2. $|\mathbf{c}_{\boldsymbol{\sigma}}| = a_{d,c_d} \Leftrightarrow f_d(\mathbf{c}_{\boldsymbol{\sigma}}) = \frac{a_{d,c_d}}{a_{d,c_d}} = 1$
3. Since $\sum_{\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}} : c_d = i} |\mathbf{c}_{\boldsymbol{\sigma}}| = a_{d,i}$ – the sum of edges connecting elements in c_d to any other set of communities equals the total number of edges of elements in c_d–,

$$\sum_{\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}} : c_d = i} f_d(\mathbf{c}_{\boldsymbol{\sigma}}) = \sum_{\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}} : c_d = i} \frac{|\mathbf{c}_{\boldsymbol{\sigma}}|}{a_{d,i}} = \frac{\sum_{\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}} : c_d = i} |\mathbf{c}_{\boldsymbol{\sigma}}|}{a_{d,i}} = 1.$$

4.3.3. Optimization

As will be shown below, the linear multi-partite modularity outperforms the argmax-based one. After finding this while comparing the algorithm using a brute-force maximization, I decided to only devise an optimized maximization method for the linear approach, which will be discussed in this section. Nevertheless, a specialized maximization approach for the argmax-based multi-partite modularity is easily conceivable, most likely as a generalization of the optimization for the bipartite approach described in Section 4.2.3.

4.3.3.1. Minimizing NEEDUPDATE

The optimization follows the general approach described in Algorithm 3 in Section 4.2.3, so performance optimization consists of finding a function NEEDUPDATE as to update as few pairs as possible (but still as many as necessary). As it turns out, it is easier to track the consequences of joins when using a linear correspondence function than when using argmax, which has undesirable non-local properties.

Consider four communities i, j, o, p in one dimension, and a single community l in another community. 3 edges connect elements in i and l, and 2 edges connect elements in each of j,o and p with elements in l. l has no further connections.

With argmax, i is the corresponding community for l. If i and j are joint, the joint $i + j$ remains the corresponding community. However, before the join, o and p together would have become the corresponding community for l. Afterwards, the four edges of $o + p$ are less than the five edges of $i + j$. So, any entries in ΔQ related to o and p have to be updated even though these communities were neither changed nor share connections with i or j – because l's future correspondence with regard to these communities changes due to the merge of i and j. This is the reason for considering set S_3 in the coupled bipartite optimization in Section 4.2.3.

With a linear correspondence function, l's correspondence to o and p is $\frac{2}{9}$ before and after joining i and j, and the correspondence to a potential future $o + p$ remains $\frac{4}{9}$. Thus, there is no need to recalculate the modularity gains through joins involving o or p. In the terminology of Section 4.2.3, set S_3 is no longer required, significantly decreasing computation time since S_3 tends to be the largest, involving all neighbours of i and j of distance 2 in domain d.

To conclude, the actual optimiziation again consists of of keeping k community $\times$ community matrices ΔQ and finding, at each step, the pair i, j in community d which yields the highest increase in modularity G. The difference is that afterwards, we only have to update those entries of ΔQ that involve any community from the sets

- S_1: the communities i, j being joint, or
- S_2: any community l in a domain $d' \neq d$ with connections to S_1.

4.3.3.2. Computing Modularity Gain

Computing the modularity gain for a join $\varphi = (d, s, t)$ is also made easier by the locality of the linear correspondence function. The only relevant community edges are those which

are incident to s or t. Their contribution before the join can be computed and subtracted, while the contribution of the new community edge with $s + t$ in their place is added.

4.3.4. Runtime Analysis

Algorithm 3 was presented as the general approach for greedy bottom-up optimizatin of modularity across several domains. It was argued that the actual runtime depends on the number of community pairs for which the modularity gain stored in ΔQ has to be updated – as computed by NEEDUPDATE –, as well as the speed of the actual updating. Now that the two main branches of modularity optimization problems – those involving argmax- based and those with linear correspondence functions – have been introduced, let us examine the runtime of the algorithm and its concrete implementations.

4.3.4.1. Theoretical Analysis

For a k, k-hypergraph $H = (V, E)$ with $N = |V|$ and $M = |E|$, $N - k - 1$ update steps are required – one element remains per domain, and no updates are required after the final step. For each step, the largest part of the computation time is spent computing the updates required after applying join φ^τ to $\boldsymbol{\sigma}^{\tau-1}$.

Let $T(H)$ designate the time to perform community detection on H. Then

$$T(H) \in \mathcal{O}\left(\sum_{\tau=1}^{N-k-1} |\textsc{NeedUpdate}(\boldsymbol{\sigma}^{\tau-1}, \varphi^\tau)|\textsc{UpdateCost}(H, \boldsymbol{\sigma}^\tau)\right)$$

I omit here components of lower complexity like the runtime of NEEDUPDATE, finding the maximum in ΔQ and changing the community assignments as they would vanish in the $\mathcal{O}$ notation anyhow. Three different realizations exist:

brute-force At each step, all gains are computed from scratch, each one by calling the modularity function on the result of performing the corresponding join.

argmax An adaptation for argmax-based modularity functions. The NEEDUPDATE function minimizes the set of required updates to all pairs that could be potentially influenced by changing argmax-relations, just as the actual update function can shortcut modularity computation by only considering potentially affected community edges.

linear An adaptation for modularity functions with linear correspondence functions. As argued in the previous section, this makes possible a NEEDUPDATE function which returns a considerably smaller set of update pairs, in particular because no pairs in the domain in which the join takes places need to be updated except those involving one of the two joined communities.

Let us now review each realization's concrete runtime behaviour.

Brute-Force At each step, all entries of ΔQ will be recomputed, for each domain d. Since the matrices are symmetric and the diagonal is not required, this leads to

$$|\textsc{NeedUpdate}_{\mathrm{brute-force}}(\boldsymbol{\sigma}, \varphi)| \propto \sum_{d=1}^{k} \frac{|C_d|(|C_d|-1)}{2} \in \mathcal{O}(N^2), \text{ where } \mathbf{C}_{\boldsymbol{\sigma}} = (C_1, \dots C_k).$$

This is a somewhat generous upper bound, since $|C_d| = |V_d|$ only in the first clustering step and at each step, $|C_d|$ is decreased by one for one of the domains. Still, the basic type of dependency is quadratic.

$$\textsc{UpdateCost}_{\mathrm{brute-force}}(H, \boldsymbol{\sigma}^{\tau}) \propto M \sum_{d=1}^{k} \log |V_d| \in \mathcal{O}(M \log N).$$

The computation of modularity requires looping over all edges and retrieving the community assignment of each involved node, which has logarithmic cost in a hashtable. It follows that

$$T(H)_{\mathrm{brute-force}} \in \mathcal{O}(N \cdot N^2 \cdot M \log N) = \mathcal{O}(N^3 \log N \cdot M).$$

Argmax The basic approach for the optimized updates consists of identifying potentially affected communities and recomputing only pairs of communities involving those. For each affected community in a domain d, the entire row, i.e. $|C_d|$ entries of the corresponding table have to be updated. For a join $\varphi = (d, s, t)$, this leads to

$$|\textsc{NeedUpdate}_{\mathrm{argmax}}(\boldsymbol{\sigma}, \varphi)| \propto 2|C_d| + \sum_{d' \in \{1,\dots,l\},\ d' \neq d}^{k} \left(|S_2| \cdot |C'_d| + |S_3| \cdot |C_d| \right),$$

where

$$S_2 = \{v \in V_{d'} : \exists e \in E : e_d \in \{s, t\} \wedge e_{d'} = v\}$$

and

$$S_3 = \{v \in V_d : \exists v \in V_d : \exists e \in E : e_d = v \wedge e_{d'} \in B\}.$$

Sets S_2 and S_3 have been introduced in more depth in the previous section. As we can see, their size depends on the connectivity of the graph, making the behaviour of this optimized version harder to quantify. Since in the worst case, however, these sets may contain all nodes in the corresponding domain, the only guaranteed bound is equivalent to the brute-force approach:

$$|\textsc{NeedUpdate}_{\mathrm{argmax}}(\boldsymbol{\sigma}, \varphi)| \in \mathcal{O}(2N + N^2 + N^2) = \mathcal{O}(N^2).$$

A similar argumentation holds for the update cost.

Linear As argued above, the advantage of using linear correspondence is that no further updates in domain d are required, therefore

$$|\textsc{NeedUpdate}_{\text{linear}}(\boldsymbol{\sigma}, \varphi)| \propto 2|C_d| + \sum_{d' \in \{1,\dots,k\},\ d' \neq d} (|S_2| \cdot |C_{d'}|) \in \mathcal{O}(2N + N^2) = \mathcal{O}(N^2).$$

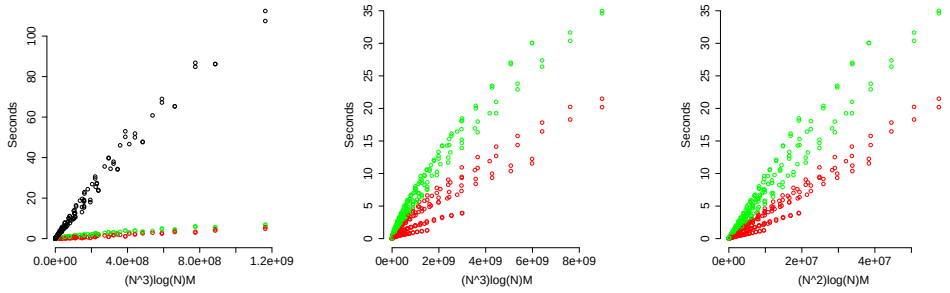
Again, the worst case behaviour is equivalent to the brute-force approach, even if the amount of recomputations is less or equal to the argmax approach.

To conclude, the runtime of the greedy bottom-up optimization in all three cases is in $\mathcal{O}(N^3 \log N \cdot M)$, even though significant speed-ups can be expected in the optimized versions. In the following, the actual time requirements for the different algorithms will be compared.

4.3.4.2. Empirical Analysis

For a comparison of actual runtimes and a validation of the theoretical results obtained in the previous section, time requirements for various synthetic hypergraphs were collected: SIMPLE,OVERLAPPING and CONTRADICTIVE graphs were generated with varying edge densities as used in the performance charts, and with varying numbers of nodes ranging from 2 to 17 for the brute-force approach, and 2 to 27 for the optimized approaches. The individual data points correspond to average values for a particular graph configuration over the different noise levels.

Figure 4.7(a) shows the time requirements for the three variants. The first thing to notice is that the optimized versions indeed run much quicker than the brute-force approach. Second, the runtimes of the brute-force approach seem in fact to depend linearly on the theoretical complexity derived in the previous section. The various "lines" with different



(a) Runtimes on a $N^3 \log N \cdot M$ scale.

(b) Runtimes of optimized algorithms only on a $N^3 \log N \cdot M$ scale.

(c) Runtimes of optimized algorithms only on a $N^2 \log N \cdot M$ scale.

Figure 4.7.: Runtime behaviour of the different optimization procedures (black=brute-force, green=argmax, red=linear). Figures b) and c) include data from larger graphs not feasible with the brute-force approach.

slopes correspond to different families of synthetic graphs, for which the various influences that were cancelled out in deriving the $\mathcal{O}$-notation have influences of different strength.

Figure 4.7(b) shows only the runtimes of the optimized approaches for better visibility. Figure 4.7(c) plots the same data on a $N^2 \log N \cdot M$ scale, i.e. leaving out one factor of N, showing a better linear dependency. This may indicate that even though the number of required updates is, in the worst case, proportional to N^2, it is in practice closer to N.

4.3.4.3. Conclusion

Greedy bottom-up optimization of modularity is a costly enterprise whose exact effort depends on the number of nodes, the number of edges, and the connectivity of the graph. Improvements furthermore depend on the actual connectivity described by the edges. Using such improvements, evidence could be presented supporting both that the worst-case runtime is in $\mathcal{O}(N^3 \log N \cdot M)$ and that the real runtime seems to be in $\mathcal{O}(N^2 \log N \cdot M)$ with, additionally, a smaller linear factor.

These results, on one hand, emphasize the beneficial effect of creating hand-tailored optimization procedures for specific modularity functions. On the other hand, much work remains to be done – a super-quadratic dependency on the number of nodes is prohibitive for larger graphs. For hypergraphs with hundreds of nodes and thousands of edges, community detection can easily take hours and days, so in order to provide real-time services or examine even larger hypergraphs, further speed-ups are required. A discussion of possible approaches is postponed to Section 6.2.

4.3.5. Results

Multi-partite modularity with a suitable correspondence function f considers all three challenges posed in Section 3.1.2. Like the coupled bipartite modularity measure, it allows for distinct community assignments per domain and has a generalized notion of equality suited for multi-partite relations. Furthermore, it meets the third challenge by a generalized notion of correspondence that can express a joint correspondence between k communities. The synthetic benchmark graphs were used again to confirm whether these theoretic advantages improve performance.

Figure 4.8 compares the performance of the linear multi-partite modularity against that of the coupled bipartite one. Figure 4.9 compares the linear and argmax approach. We observe four results:

1. The two multi-partite modularities are the first approaches to perform well on the CONTRADICTIVE graphs.

   This is the most important result. The multi-partite generalizations of modularity were set up to consider relations that can only be detected using the full tripartite information contained in the hypergraph. The success of the methods confirms that this goal is met.

2. They perform similarly convincingly on the OVERLAPPING dataset, the linear variant even outperforms the coupled bipartite approach.

This is another positive, but somewhat surprising result. The communities in the OVERLAPPING dataset can, in theory, be fully reconstructed without considering tripartite relations - so where does the advantage of the linear multi-partite modularity come from? It seems that by considering tripartite relations, wrong decisions can be avoided in ambiguous situations where a wrong bipartite join might offer the highest gain in the short term.

3. The linear variant consistently outperforms the argmax-based variant.

   One has to keep in mind that the goal of modularity optimization is to create community edges with a high density. Through the argmax function, only one community edge per community enters the equation. In all cases but the SIMPLE one, more than one relevant community edge exists for at least one (OVERLAPPING) or all (CONTRADICTIVE) communities (for reference, see Figure 3.3 – the black circles indicate the relevant community edges). The possibility to reward progress in different community edges in parallel is the crucial advantage of the linear correspondence function.

4. Neither of the two multi-partite methods perform particularly well on the SIMPLE dataset.

   This was, to me, the single biggest surprise I found during the work on this thesis. One might expect that a more complex approach works better on *all* possible situations – instead we lose performance on the simple situations after concentrating on the more complicated ones. The next section will be wholly dedicated to explaining and resolving this problem.

At this point, the goal of devising a multi-partite modularity measure has been reached. However, even using only synthetic data, we can see that this is not enough. The mathematically pure concept of linear multi-partite modularity needs to be additionally refined in order to deal with all possible situations it might encounter. This will be the topic of the next two sections.

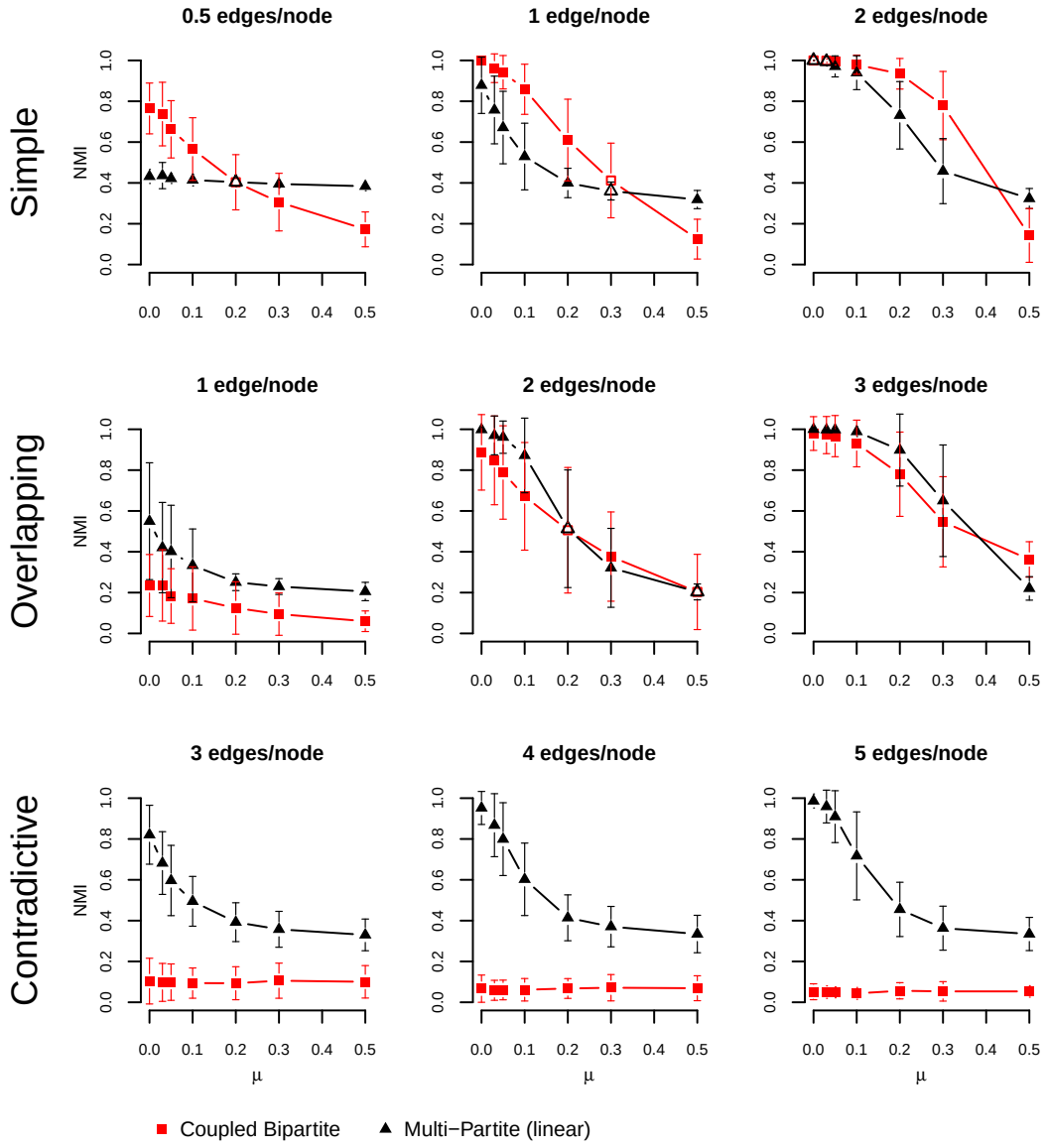


Figure 4.8.: Comparing the coupled bipartite algorithm to the (linear) multi-partite approach: The latter finds sensible solutions on the CONTRADICTIVE datasets, but performs poorly in the sparser SIMPLE settings. See Section 4.1.2 for a detailed explanation of this chart.

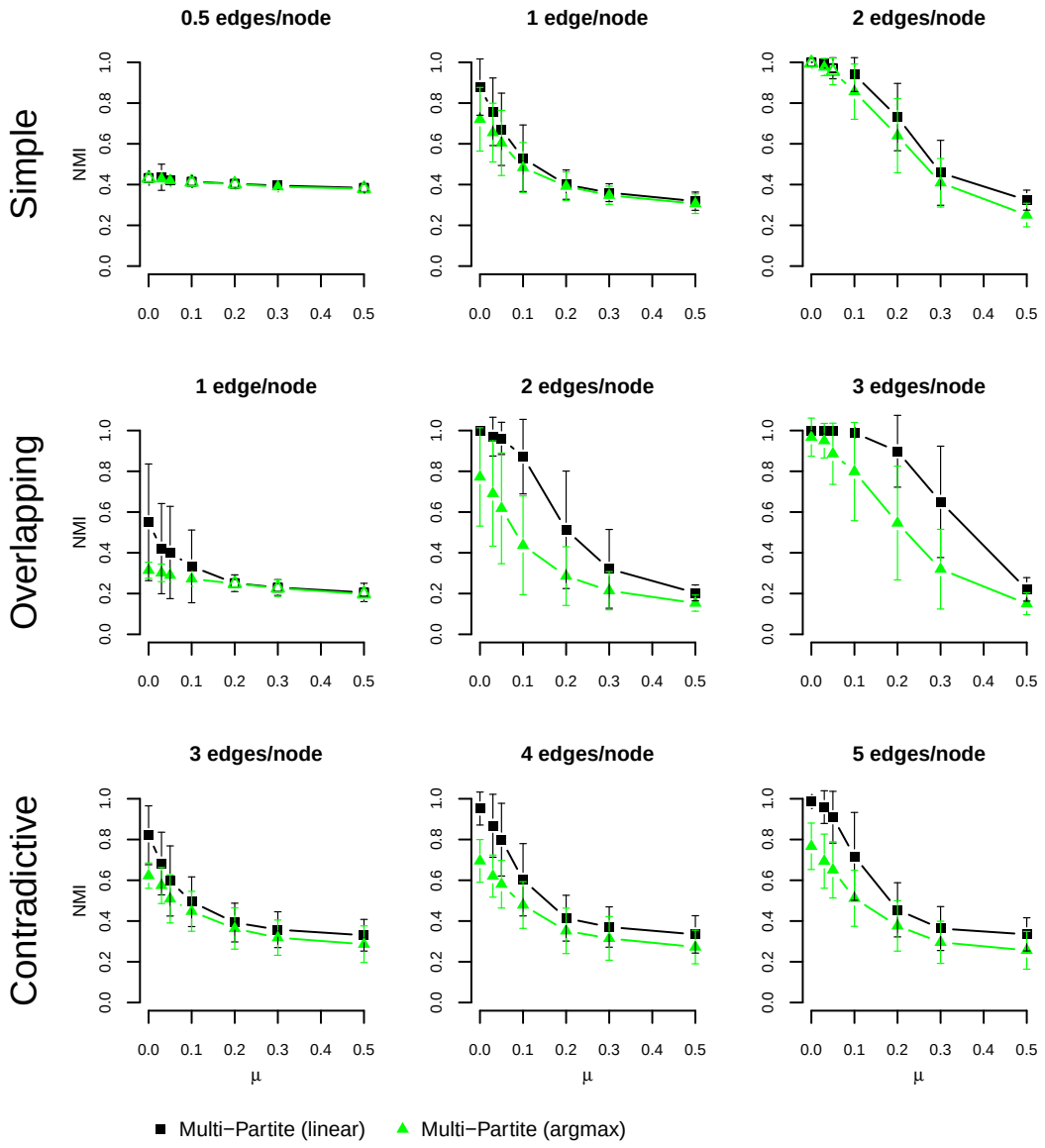


Figure 4.9.: Comparing the linear to the argmax-based multi-partite approach: The linear approach outperforms the argmax approach in all situations but those few where the approaches tie. See Section 4.1.2 for a detailed explanation of this chart.

4.4. Hybrid Modularity Optimization

As we have seen in the previous section, multi-partite modularity succeeds in extracting complicated situations, but sometimes fails in identifying the simple ones. In this section, I will first identify the cause of this behaviour, and then proceed to two alternative community detection methods, the latter of which, called hybrid modularity optimization, can handle all synthetic graph families with satisfying performance.

4.4.1. Why Multi-Partite Modularity Fails

Figure 4.10 shows a simple situation in which multi-partite modularity fails to create the expected result. Given an intermediate clustering $\boldsymbol{\sigma}^\tau$ in 4.10(a), we would expect d_1 and d_2 to be merged in a next join $\varphi^+ = (1, d_1, d_2)$ (Figure 4.10(b)), or u_1 and u_2. Instead, the best next step modularity-wise is to join d_1 and d_3 ($\varphi^- = (1, d_1, d_3)$, see Figure 4.10(c)), even though d_1 and d_2 share a tag, whereas d_1 and d_3 don't share anything.

4.4.1.1. Numerical analysis

To understand this strange behaviour, let us compute the modularity for each community assignment in detail. Note that modularity is computed as a sum over all non-empty community edges. For each community edge, the summands take the form $x \cdot f$, where x contains the actual minus the expected adjacency (let us call this "raw modularity") and f represents the correspondence.

$$Q_{\mathrm{MP}}(\boldsymbol{\sigma}) = \sum_{\underbrace{\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}}}_{\text{community edges}}} \Bigg(\underbrace{\left(\frac{|\mathbf{c}_{\boldsymbol{\sigma}}|}{M} - \prod_{d=1}^{k} \frac{a_{d,c_d}(\boldsymbol{\sigma})}{M} \right)}_{x} \underbrace{\sum_{d=1}^{k} f_d(\mathbf{c}_{\boldsymbol{\sigma}})}_{f} \Bigg).$$

Since in the following, particular values for x will repeat themselves, let

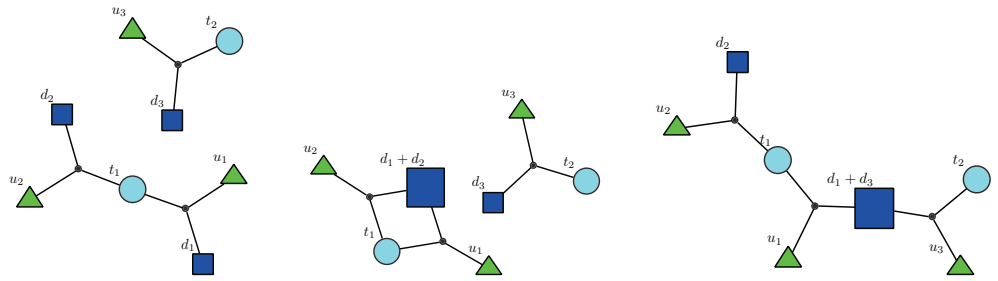
$a = \frac{1}{3} - \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{2}{3} = \frac{7}{27}$ be the raw modularity of a community edge that contains one edge between 2 communities which only have one edge and one which contains two,

$b = \frac{1}{3} - \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{3} = \frac{8}{27}$ the raw modularity when all incident communities have only one edge, and

$c = \frac{1}{3} - \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{2}{3} = \frac{5}{27}$ the raw modularity when two incident communities have two edges, and one has one.

Then, inserting the correspondence terms – which are either 1 or 0.5, depending on whether a community edge contains all or just half of a community's edges – we obtain

$$\begin{aligned}
Q_{\mathrm{MP}}(H, \boldsymbol{\sigma}^\tau) &= 2a(1+1+0.5) + b(1+1+1) &&= 5a + 3b = \frac{59}{27}, \\
Q_{\mathrm{MP}}(H, \boldsymbol{\sigma}^\tau \circ \varphi^+) &= 2c(0.5+1+0.5) + b(1+1+1) &&= 4c + 3b = \frac{44}{27}, \quad \text{and} \\
Q_{\mathrm{MP}}(H, \boldsymbol{\sigma}^\tau \circ \varphi^-) &= 2a(1+1+0.5) + c(0.5+1+0.5) &&= 5a + 2c = \frac{45}{27}.
\end{aligned}$$



(a) Community structure given by some $\boldsymbol{\sigma}^\tau$

(b) Community structure given by $\boldsymbol{\sigma}^\tau \circ \varphi^+$, a desirable next join

(c) Community structure given by $\boldsymbol{\sigma}^\tau \circ \varphi^-$, the actual next join

Figure 4.10.: Given an assignment function $\boldsymbol{\sigma}^\tau$ at some point in the clustering of a hypergraph, $\boldsymbol{\sigma}^\tau \circ \varphi^+$ and $\boldsymbol{\sigma}^\tau \circ \varphi^-$ represent two possibilities to continue the clustering.

So $\boldsymbol{\sigma}^\tau \circ \varphi^-$ really is a better choice than $\boldsymbol{\sigma}^\tau \circ \varphi^+$, analytically. This discrepancy with the intended behavior lies in the role of the two community edges $(d_1, u_1, t_1)_{\boldsymbol{\sigma}}$ and $(d_2, u_2, t_1)_{\boldsymbol{\sigma}}$. As they are transformed into $(d_1 + d_2, u_1, t_1)_{\boldsymbol{\sigma}\circ\varphi^+}$ and $(d_1 + d_2, u_2, t_1)_{\boldsymbol{\sigma}\circ\varphi^+}$, correspondence decreases and expected adjacency goes up due to $d_1 + d_2$'s increased size, so their contribution to modularity drops from $5a$ to $4c$. The intuitive advantage, the fact that d_1 and d_2 share connections with t_1, remains unrewarded by the modularity measure. By focussing on tripartite relations, it cannot capture the improvement in *binary* relations that this join would bring. The join φ^-, on the other hand, looks just as well or badly connected under this measure, but has the advantage of creating only one community edge connecting two communities of weight 2, thereby slightly outperforming φ^+, which contains two such community edges.

4.4.1.2. Generalized Explanation

More generally speaking, linear multi-partite modularity only increases when two former community edges are collapsed onto each other. Let us formulate and prove this assumption formally – this not only ensures the generality of the phenomenon, but also plays a crucial role for the decision criterion of hybrid modularity which will be defined below.

Lemma 4.1 *For a join $\varphi = (1, s, t)$ (assuming w.l.o.g., for notational convenience, that $k = 3$ and that the join is taking place in domain 1) applied to $\boldsymbol{\sigma}$,*

$$\big(\forall (i,j) \in C_2 \times C_3 : (|\mathbf{c}_{\boldsymbol{\sigma}}(s,i,j)| = 0 \vee |\mathbf{c}_{\boldsymbol{\sigma}}(t,i,j)| = 0)\big) \Longrightarrow G_{\mathrm{MP|lin}}(\boldsymbol{\sigma}, \varphi) < 0.$$

This means that if no pair of non-empty community edges $(s,i,j)_{\boldsymbol{\sigma}}$, $(t,i,j)_{\boldsymbol{\sigma}}$ exists that will be merged into a single $(s+t,i,j)_{\boldsymbol{\sigma}\circ\varphi}$, the gain in linear multi-partite modularity gain is negative, regardless of partial matches such as $(s,i,j)_{\boldsymbol{\sigma}}$ and $(t,i,j')_{\boldsymbol{\sigma}}$, as present in the example.

Please note that I constrain myself to the linear version here. A proof for the argmax variant would be decidedly more complex due to the involved non-local effects, while the

basic approach remains the same, showing that actual connectivity remains the same while expected connectivity is increased and connectivity, at best, stays the same.

Proof 4.1 *Let $Q(\mathbf{c}_{\boldsymbol{\sigma}})$ denote the individual contribution of each community edge, such that*

$$Q(\boldsymbol{\sigma}) = \sum_{\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}}} Q(\mathbf{c}_{\boldsymbol{\sigma}}).$$

Using this terminology (and replacing $Q_{\mathrm{MP|lin}}$ by Q for readability),

$$\begin{aligned}
& G(\boldsymbol{\sigma}, \varphi) && < 0 \\
\iff\ & Q(\boldsymbol{\sigma}) && > Q(\boldsymbol{\sigma} \circ \varphi) \\
\iff\ & \sum_{\mathbf{c}_{\boldsymbol{\sigma}} \in \mathbf{C}_{\boldsymbol{\sigma}}} Q(\mathbf{c}_{\boldsymbol{\sigma}}) && > \sum_{\mathbf{c}_{\boldsymbol{\sigma}\circ\varphi} \in \mathbf{C}_{\boldsymbol{\sigma}\circ\varphi}} Q(\mathbf{c}_{\boldsymbol{\sigma}\circ\varphi}) \\
\iff\ & \sum_{\mathbf{c}_{\boldsymbol{\sigma}}=(s,i,j),(d,i,j) \in \mathbf{C}_{\boldsymbol{\sigma}}} Q(\mathbf{c}_{\boldsymbol{\sigma}}) && > \sum_{\mathbf{c}_{\boldsymbol{\sigma}\circ\varphi}=(s+t,i,j) \in \mathbf{C}_{\boldsymbol{\sigma}\circ\varphi}} Q(\mathbf{c}_{\boldsymbol{\sigma}\circ\varphi}) && (4.1)
\end{aligned}$$

$$\begin{aligned}
\impliedby\ & \forall (i,j) \in C_2 \times C_3 : \\
& Q((s,i,j)_{\boldsymbol{\sigma}}) + \underbrace{Q((t,i,j)_{\boldsymbol{\sigma}})}_{=0 \text{ per condition}} && > Q((s+t,i,j)_{\boldsymbol{\sigma}\circ\varphi}) && (4.2)
\end{aligned}$$

$$\begin{aligned}
& && \big(\underbrace{e_{s,i,j}}_{=:x>0} - a_s \underbrace{a_i a_j}_{=:y>0}\big) \cdot && \big(e_{s,i,j} + \underbrace{e_{t,i,j}}_{=0 \text{ per condition}} - (a_s + a_t) a_i a_j\big) \cdot \\
\iff\ & \left(e_{s,i,j}\left(\frac{1}{a_s} + \underbrace{\frac{1}{a_i} + \frac{1}{a_j}}_{=:z>0}\right)\right) && > \left((e_{s,i,j} + e_{t,i,j})\left(\frac{1}{a_s + a_t} + \frac{1}{a_i} + \frac{1}{a_j}\right)\right) \\
\iff\ & (x - a_s y)\left(\frac{x}{a_s} + z\right) && > (x - (a_s + a_t)\, y)\left(\frac{x}{a_s + a_t} + z\right) && (4.3) \\
\iff\ & a_t && > 0
\end{aligned}$$

Step 4.1 is possible because the summands of the two sums are identical except for those community edges which are affected by join $\varphi = (1, s, t)$, i.e. those involving community s or t before the join or the new community $s + t$ after the join.

Step 4.2 makes an even stronger argument than necessary by stating that the inequality not only holds for the whole sum, but for each individual set of involved community edges (s, i, j), (t, i, j) and $(s + t, i, j)$. Per condition, either $Q_{\mathrm{MP|lin}}((s,i,j)_{\boldsymbol{\sigma}}) = 0$ or $Q_{\mathrm{MP|lin}}((t,i,j)_{\boldsymbol{\sigma}}) = 0$; w.l.o.g. we assume the latter from step 4.2 on and below.

As can be seen in step 4.3, a_t can only decrease the right hand side of the inequality. As a_t refers to the weight of a non-empty community, it is always positive.

It follows that joins that create only bipartite improvements, e.g. $\varphi = (1, d_1, d_2)$ where community edges $(d_1, u, t)_{\boldsymbol{\sigma}}$ and $(d_2, u', t)_{\boldsymbol{\sigma}}$ exist – like φ^+ –, never contribute positively to linear multi-partite modularity.

Due to the greedy optimization strategy, this neglect may have have severe consequences. Since $\boldsymbol{\sigma}^\tau \circ \varphi^+$ is rejected and a wholly different path is chosen via $\boldsymbol{\sigma}^\tau \circ \varphi^-$, the way to a fully merged community edge $(d_1 + d_2, u_1 + u_2, t_1)_{\boldsymbol{\sigma}\circ\varphi^+\circ\varphi'}$ – which *would* be rewarded by the modularity measure – is blocked in our example. This demonstrates how this problem affects not only a single, but also the future clustering steps.

As pointed out by my colleague Wendelin Böhmer, there is a certain analogy to fitting a function: You can generally do more with a higher-order function class, but you still need the lower-order terms. I will now proceed with two possible approaches towards integrating the "lower-order" coupled bipartite modularity.

4.4.2. Mixed Multi-Partite Modularity

Using the knowledge that multi-partite modularity fails in some cases and coupled bipartite modularity in others, it seems promising to create a modularity measure which integrates both measures' input. A simple attempt to achieve this is to define a function which mixes the two in a linear fashion.

Definition 4.6 (Mixed Multi-Partite Modularity) *Let the* mixed multi-partite modularity *of a k, k-hypergraph H under a community assignment $\boldsymbol{\sigma}$, a correspondence function f and a mixing factor β be defined as*

$$Q_\beta = \beta Q_{\mathrm{MP}}(H, \boldsymbol{\sigma}, f) + (1 - \beta) Q_{\mathrm{CB}}.$$

This introduces the question of how to set β. Figure 4.11 shows the results on the synthetic test graphs for various values of β. We find that

- On the SIMPLE and OVERLAPPING dataset, a linear interpolation between the two methods basically creates a linear interpolation in performances. The higher we set β, the better the performance becomes on the OVERLAPPING examples and the worse it gets in the SIMPLE case.
- For the CONTRADICTIVE case, the behaviour is more non-linear. At $\beta = 0.5$, usable results can be achieved, whereas below that threshold, results are basically useless – even though the curve for $\beta = 0.25$ looks considerably higher than the one for $\beta = 0$, the fact that the former does not react to noise or increased edge density tells us that for both values, the results returned are a static baseline result.
- So, if certain losses in quality for the first two cases can be accepted, any value $\beta > 0.5$ can provide acceptable results for all three families, while even higher values can improve performance on OVERLAPPING and CONTRADICTIVE, at the cost of performance on SIMPLE.

These results show that there is hardly any synergetic effect by combining the two modularity measures in this way. Instead β seems to allow us to put a weight on which type

of phenomena we would like to be detected more effectively. So can we inspect a real-life dataset, find out how it is structured and then find a good value for β? While one might try different values for β and choose the best result based on a quality measure different than modularity, e.g. the compression ratio, it is important to keep in mind that the synthetic test sets do not aim to represent real graphs. Instead, they should be seen as building blocks, representing in an isolated fashion various phenomena which can simultaneously occur in real data. So, instead of asking whether a whole graph is globally "more SIMPLE" or "more OVERLAPPING", it would be more desirable to analyse the local situation at each clustering step and choose a the optimal approach based on that.

4.4.3. Hybrid Modularity Optimization

There are situations in which multi-partite modularity outperforms the coupled bipartite modularity, and there are other situations in which the opposite is the case. As it turns out, these situations can be defined rather clearly, allowing a more targeted decision between the two than the global mix defined in the previous section.

4.4.3.1. Definition and Optimization

Hybrid modularity optimization, as the name suggests, differs from the previous modularities in that it does not actually provide a new definition for modularity, but instead changes the optimization scheme. So far, a single modularity measure was used to identify the optimal join at each step τ, given a current community assignment $\boldsymbol{\sigma}$. Here, instead, at each step, a decision is made as to which modularity to use for finding the next join.

Basically, the relative advantages of multi-partite and coupled bipartite modularity can be easily defined for a given situation:

- If there is a possibility to create a join that merges two community edges, e.g. $(d, u, t)_{\boldsymbol{\sigma}}$ and $(d', u, t)_{\boldsymbol{\sigma}}$, Q_{MP} identifies this, whereas Q_{CB} may not favor such a join over one involving only a bipartite improvement, e.g. joining d and d' when, only (d, u, t) and (d', u', t) exist. This follows from the definition and is confirmed by the two approaches' performance on the CONTRADICTIVE dataset.

  In these situation, we would like to base the decision wholly on Q_{MP}.

- If no merging of two community edges is possible, Q_{MP} is not able to distinguish whether joins create at least bipartite improvements or not, as shown in the example involving φ^+ and φ^-, whereas this is exactly what Q_{CB} examines.

  In these situations, we would like to base the decision wholly on Q_{CB}.

Lemma 4.1 provides a simple decision criterion to decide whether, at a given clustering step τ with intermediate community assignments $\boldsymbol{\sigma}^\tau$, the algorithm is facing the first or the second situation. If there is a join φ such that $G_{\mathrm{MP}}(\boldsymbol{\sigma}^\tau, \varphi) > 0$, we can safely apply φ because this means that $\boldsymbol{\sigma}^\tau$ is in the domain where Q_{MP} yields good results. Otherwise, the next join is chosen using Q_{CB}. Algorithm 4 highlights the changes to the original optimization algorithm 3.

Algorithm 4 Changing the original to a hybrid modularity optimization algorithm

function OPTIMIZE(E,V,Q)
  initialize ΔQ_{MP} and ΔQ_{BP}
  ...
  for $\tau = 1 \dots |V| - k$ **do**
    ...
    $\varphi \leftarrow \mathrm{argmax}_{\varphi'=(d,s,t)}(G_{\mathrm{MP}}(\boldsymbol{\sigma}^{\tau-1}, \varphi')) = \mathrm{argmax}_{\varphi'=(d,s,t)}(\Delta Q_{\mathrm{MP},d}(s,t))$
    if $G_{\mathrm{MP}(\boldsymbol{\sigma}^{\tau-1},\varphi)<0}$ **then**
      $\varphi \leftarrow \mathrm{argmax}_{\varphi'=(d,s,t)}(G_{\mathrm{CB}}(\boldsymbol{\sigma}^{\tau-1}, \varphi')) = \mathrm{argmax}_{\varphi'=(d,s,t)}(\Delta Q_{\mathrm{CB},d}(s,t))$
    end if
    ...
  end for
  return $(\boldsymbol{\sigma}^0, \dots, \boldsymbol{\sigma}^{|V|-k})$
end function

Lemma 4.1, by the way, does not mean that a negative $G_{\mathrm{MP}}(\boldsymbol{\sigma}, \varphi)$ necessarily implies that no community edges will be merged by φ. Another decision criterion would therefore be to check for merged edges directly and switch to Q_{CB} only if no such edges are found, i.e., using the multi-partite modularity even in some cases where it yields a negative modularity gain. Experimental results (not shown here) indicate a minor shift in performance from SIMPLE to CONTRADICTIVE cases. Since the difference is minimal, I will restrict myself to the approach defined above in the following.

Regrettably, hybrid modularity optimization comes with theoretical and practical costs: On the theoretical side, it is no longer possible to provide in closed form the quality function that is being optimized. On the practical side, we need to keep both G_{MP} and G_{CP} updated after each join, i.e. the computational costs of the two modularities involved add up. In the following, it will be examined whether the quality of the results justifies these costs.

4.4.3.2. Using Linear Coupled Bipartite Modularity

The replacement of the argmax-based by a linear correspondence function can also be applied on the coupled bipartite modularity: This simply means that the modularity applied on the individual bipartite graphs is, instead of Murata's bipartite modularity, Suzuki and Wakita (2009)'s bipartite modularity. While the argmax-based coupled bipartite modularity works better if used alone – compare red lines in Figure 4.11 and A.6 – the linear coupled bipartite modularity in fact works better as a component in the mixed and hybrid settings, while benefitting from the speed improvements associated with linear correspondence functions. The performance charts for the mixed and hybrid functions with argmax-based bipartite modularity are shown in the Appendix (Figure A.6) – in the following however, mixed and hybrid approaches of linear multi-partite modularity refer to using a coupled bipartite modularity with linear correspondence.

4.4.4. Results

Figure 4.11 compares the results of the hybrid modularity optimization with those of the mixed modularity for various values of β, including $\beta = 0$ and $\beta = 1$ for purely bipartite/multi-partite modularity.

- On the SIMPLE dataset, the hybrid optimization performs comparable to the coupled bipartite approach, particularly in the sparse regime (0.5 edges/node) in which multi-partite approaches fail to find any structure. In the denser regimes, it is sometimes outperformed by the medium-β approaches, but these differences seem rather small compared to the qualitative difference between finding a solution or not. These results indicate that the detection works in situations in which multi-partite modularity is inadequate.
- On the CONTRADICTIVE dataset, hybrid optimization yields results comparable to $\beta = 0.5$, i.e., it struggles in the sparser environments but does retrieve the desired structure once enough information is available. These results indicate that the detection works as well in situations in which coupled bipartite modularity is inadequate.
- It is the OVERLAPPING dataset in which hybrid optimization really shines. It outperforms all other approaches by a rather large margin particularly in the sparser regimes. It is one thing for the hybrid optimization to basically find the right mix adaptively between the two modularities, as shown in the previous two points. It is a different thing, however, to outperform the two constituent modularities as clearly as in the results for 1 edge/node. This indicates that the analysis of the two measures' strengths and weaknesses has really led to a synergetic effect and creates results impossible to achieve for any single measure alone.

To summarize, hybrid modularity optimization achieves the best aggregate performance over all datasets and seems to be the community detection approach of choice unless a quality measure in closed form is required or time performance is particularly crucial.

4.4.5. Conclusions & Outlook

This concludes the improvements of modularity towards performance on the synthetic test datasets. The fact that the solution for the multi-partite/bipartite trade-off lies in a change of the optimization routine suggests that further improvements might be possible by tweaking that routine. One possibility might be to look two joins ahead, such that intermediate joins like φ^+ which would lead to a good result in the next step can be correctly judged. An elaborate theoretical analysis would however be necessary to avoid the combinatorial explosion of the associated search space.

This section has strongly focussed on $3,3$-hypergraphs. A hybrid community optimization for k-partite hypergraphs for $k > 3$ could work exactly as described here, using the k-partite and the coupled bipartite approach. A more elaborate approach would involve defining $k-1$-partite projections of k-partite hypergraphs, generalizing the bipartite projections defined in Def. 4.3. Like this, community edges sharing $k-1, k-2, \ldots$ communities could be detected.

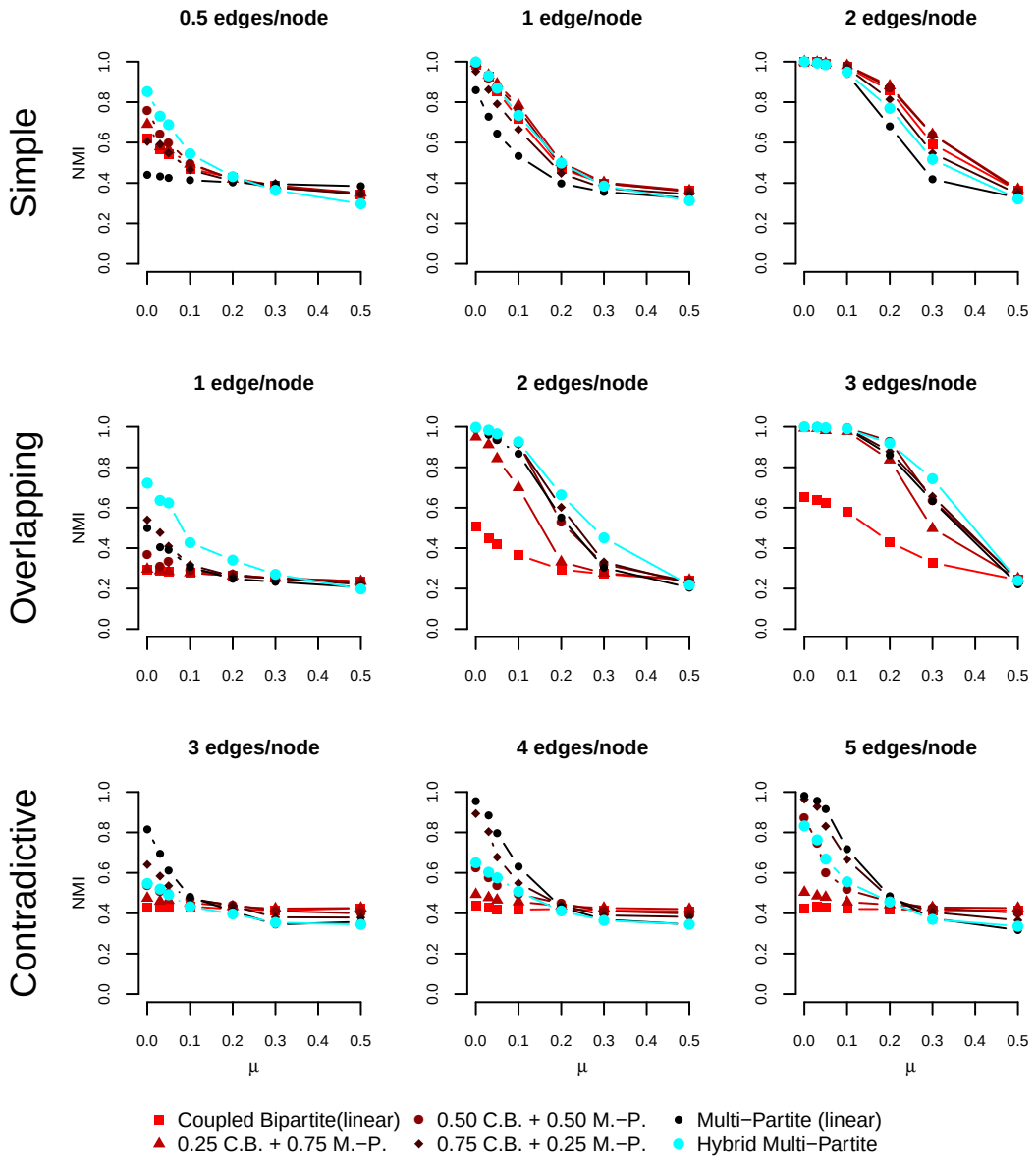


Figure 4.11.: Comparing the hybrid approach to linear interpolations between the linear coupled bipartite and linear-multi-partite modularity: The hybrid approach is the only one which performs reasonably well under all conditions. Error bars omitted for clearness. See Section 4.1.2 for a detailed explanation of this chart.

4.5. Balanced Multi-Partite Modularity

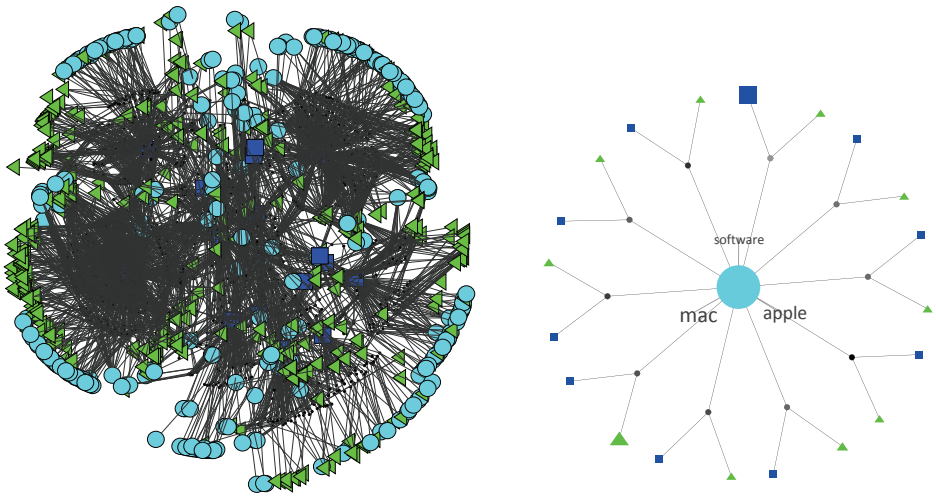
Some phenomena encountered applying multi-partite modularity on real hypergraphs are not caught by the synthetic datasets. This section will discuss a final adjustment to linear multi-partite modularity to deal with a particular crucial issue that comes up in certain hypergraphs. Parts of this section anticipate concepts properly introduced only in the next chapter on exploring real datasets. However, as these issues require an adaptation of the modularity measure, they have to be addressed here.

I will first describe the problem using a sample hypergraph, then analyze it more generally and propose a new measure, balance, to quantify its occurence. Two changes to multi-partite modularity are proposed to mitigate the undesired effects: dampened modularity involving a dampening parameter α and balanced modularity which is parameter-free. The beneficial influence of these changes will be demonstrated on three prototypical hypergraphs.

4.5.1. Problem Description

When performing community detection on real hypergraphs, we want to obtain *useful* community assignments. While usefulness is a fuzzy term and its exact interpretation may differ between applications, two phenomena have frequently occured that can safely be considered not useful even without a particular application in mind.

First, for many hypergraphs, solutions tend to be suggested in which all elements of one domain are grouped into a single community. This may be a sensible solution in some cases, but concrete examples exist where community structure could clearly be distinguished visually in the original hypergraph, and yet all elements were joint into a single community.



(a) A sample hypergraph H_{medium}

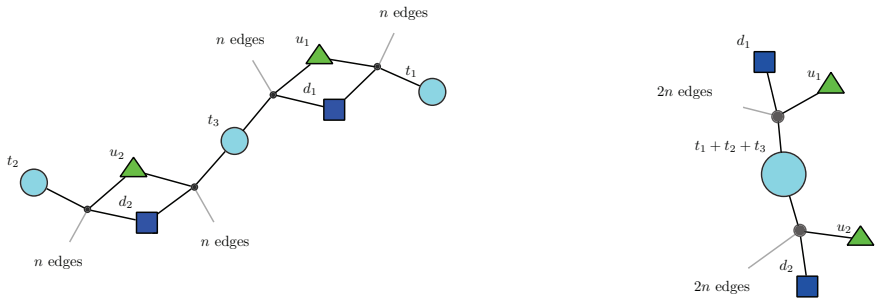
(b) Community assignment of linear multi-partite modularity optimization

Figure 4.12.: A hypergraph from obtained data and a problematic community assignment

As an example, consider the hypergraph shown in Figure 4.12(a), containing all edges involving documents of a particular user from the Delicious dataset (a "user expansion" – see Section 5.1 for more sophisticated subgraph extraction techniques and associated use cases). Even though there are clearly discernible groups of tags (turqoise circles) only associated with some of the documents – basically all of the outer tags, whereas the more centrally positioned tags seem well connected to all documents –, the optimal community assignment for multi-partite modularity assigns all of them to a single community. Figure 4.12(b) shows a visual representation of the clustering (see Section 5.2 for a detailed description of this type of visualization) and gives an intuitive idea of why this is not particularly informative.

Second, the situation cannot be easily resolved by manually splitting the clusters. Since all nodes eventually end up in a single community, the cause for the problem mentioned above could be that this happens earlier in one domain than in the others. If this was the only problem, it could be fixed by manually expanding that single community, i.e. undo some of the final joins, or, thinking in terms of dendrograms, "going deeper" into the clustering tree. However, in the problematic cases we're interested in here – when the single community doesn't legitimately represent a pattern in the data – this does not help: The giant communities tend to split into a tiny spin-off and a mostly unchanged new giant community, a symptom of individual nodes being added to this single community all through the clustering process. See 4.15(c) for the dendrogram of the tag domain that corresponds to the solution presented in Figure 4.12(b).

The problem is thus not a single bad solution – the whole clustering tree is pathologically formed. The fact that nodes, in such cases, can just as well be all joint into a single community, is a mere symptom. Consider two communities of users u_1 and u_2, each of which has tagged on community of documents, d_1 and d_2, respectively. Half of the edges used by each user community connect the documents to individual tag communities t_1 and t_2, the other half associates the documents with a shared tag community t_3. This is basically an



(a) $\boldsymbol{\sigma}^{\tau}$: t_1, t_2 and t_3 are disjunct communities.

(b) $\boldsymbol{\sigma}^{\tau+2}$: t_1 and t_2 have been merged into t_3.

Figure 4.13.: A hypothetical hypergraph under two different clusterings: t_1 and t_3 are either individual tag clusters or all joint into a single, central tag cluster. Linear multi-partite modularity assigns a higher score to the latter case.

abstraction of the situation shown in Figure 4.12(a): A central set of tags which seems to be applied to a lot of documents, and a number of "arms" connected to only parts of the documents.

There are two relevant possibilities to cluster this data. The first one – let us refer to it as $\boldsymbol{\sigma}^{\tau}$ – is to keep distinct communities for t_1 and t_2, as shown in Figure 4.13(a). The other possibility $\boldsymbol{\sigma}^{\tau+2}$ (i.e. two steps further) is to merge the two outer tag communities into the central one, as shown in Figure 4.13(b). The first option appears more desirable. From a user's point of view, it tells a clearer story. Looking at the visualization of the second option, one might get the impression that the two user communities only differ with respect to their associated documents and use the same set of tags. From an analytic point of view, one might expect that a clear separation of the tags by usage would yield better results than merging them and creating implicit connections between elements which do not share a connection. However, once again we can observe how the intuitions about the modularity are not matched by its actual behaviour: It turns out that $Q_{\mathrm{MP|lin}}(\boldsymbol{\sigma}^{\tau}) = \frac{9}{8}$, whereas $Q_{\mathrm{MP|lin}}(\boldsymbol{\sigma}^{\tau+2}) = \frac{10}{8}$. On the bright side, this means we now have an atomic example of the current problem at hand.

4.5.2. Numerical Analysis

Let us take a look at the more general case of m instead of two "arms". For $\boldsymbol{\sigma}^{\tau}$, Q is the sum of the contributions of the m outer community edges (called Q_{outer} in the equation) and the m inner community edges, Q_{inner}, whereas the modularity of $\boldsymbol{\sigma}^{\tau+2}$ is computed as the sum of the contribution from m identical community edges Q_{merged}. The sum of edge weights $M = 2mn$, but all occurrences of n are in fact cancelled out in the actual computation. So, the modularity for the clustering in which the arms remain unmerged is

$$\begin{aligned}
Q_{\mathrm{MP|lin}}(\boldsymbol{\sigma}^{\tau}) = & \quad m \cdot Q_{\mathrm{outer}} + m \cdot Q_{\mathrm{inner}} \\
= & \quad m\left(\frac{1}{2m} - \frac{1\cdot 2\cdot 2}{8m^3}\right)\left(1+\frac{1}{2}+\frac{1}{2}\right) && (4.4)\\
& + m\left(\frac{1}{2m} - \frac{2\cdot 2\cdot m}{8m^3}\right)\left(\frac{1}{2}+\frac{1}{2}+\frac{1}{m}\right) && (4.5)\\
= & \quad \frac{3}{2} - \frac{3}{2m^2}\,,
\end{aligned}$$

whereas the modularity for clustering all arms into the central cluster is

$$\begin{aligned}
Q_{\mathrm{MP|lin}}(\boldsymbol{\sigma}^{\tau+m}) = & \quad m \cdot Q_{\mathrm{merged}} \\
= & \quad m\left(\frac{2}{2m} - \frac{2\cdot 2\cdot 2m}{8m^3}\right)\Big(\underbrace{1+1+\frac{1}{m}}_{x}\Big) && (4.6)\\
= & \quad 2 - \frac{1}{m} - \frac{1}{m^2}.
\end{aligned}$$

The difference between the desired and the undesired outcomes converges to 0.5 as the number m of arms increases (to put these numbers into scale, please note that I did not

normalize the linear multi-partite modularity here, so its maximum is k or 3 instead of 1). The more arms exist, the more beneficial it is to eliminate them and create a central tag cluster. This is clearly not the intended behavior.

The cause of the problem is marked with an x. As the tag communities are merged, documents and users are connected to a single tag community containing lots of tags they are in fact unconnected to. However, this only decreases correspondence from the perspective of the tag community (the $\frac{1}{m}$ term in x). The correspondence of the bloated community edge for the document and user communities, on the other hand, goes up to 1: All their incident edges are contained in a single community edge. Even though the expected adjacency increases, the increase in correspondence is more than enough to mitigate both this and the tag community's poor correspondence value. In other words, the optimization, in certain situations, leads to community assignments that are bad for one domain, as long as they are good for the other domains. The key to resolving the described problems is therefore to enable modularity to detect such types of imbalances and punish the corresponding solutions accordingly. Below, two approaches towards this goal are introduced.

4.5.3. Balance

Before the changes to the modularity measure are introduced, we require a measure to identify whether these approaches perform correctly. Since the very problem is that modularity yields high values for undesirable solutions, it cannot be used for evaluation.

One possibility is to inspect the solutions manually and see if communities in one domain are collapsed into one. The ultimate goal is to prevent exactly this from happening – except where justified – however, it seems desirable to have a quantitative measure at hand, e.g., for automatic evaluation.

The measure proposed here, balance, inspects the entire hierarchical clustering obtained, and scans for the existence of giant communities.

Definition 4.7 (Balance) *Let the* balance *in the dth domain of a k-partite clustering $\Sigma = (\boldsymbol{\sigma}^0, \ldots, \boldsymbol{\sigma}^{\tau_{\max}})$ be*

$$\mathrm{bal}_d(\Sigma) = \sum_{\tau=0}^{\tau_{\max}} \frac{\mathrm{bal}_d^\tau}{\tau_{\max} + 1} ,$$

where

$$\mathrm{bal}_d^\tau = \frac{a_{d,\mathrm{secondmax}}(\boldsymbol{\sigma}^\tau)}{a_{d,\max}(\boldsymbol{\sigma}^\tau)} ,$$

and $\max$ *and* $\mathrm{secondmax}$ *refer to the community in d with the highest and second-highest number of incident edges (a) for a given community assignment $\boldsymbol{\sigma}^\tau$.*

So, a low value for bal_d means that over many steps, the community with the most incident edges has a lot more such edges than the next-biggest one. By considering all steps, this measure distinguishes "legit" big communities, merged from several big ones in a late stage, from pathological giant communities created by continually adding nodes to the same community, as shown in Figure 4.15(c) for the example discussed initially.

4.5.4. Dampened Multi-Partite Modularity

Since the cause of giant communities seems to be a lack of concern about weak correspondence values for single communities, the so-called dampened multi-partite modularity replaces the simple sum of all correspondence values with a softmin function:

Definition 4.8 (Dampened Multi-Partite Modularity) *Let the* dampened multi-partite modularity *of a 3,3-hypergraph H under a community assignment $\boldsymbol{\sigma}$, a correspondence function f, and a damping factor α be defined as*

$$Q_{\mathrm{MP},\alpha} = \sum_{(i,j,k)\in C_1\times C_2\times C_3} (e_{ijk} - a_i a_j a_k)\,\mathrm{softmin}_\alpha(f_1(i,j,k), f_2(i,j,k), f_3(i,j,k))\,,$$

or

$$Q_{\mathrm{MP},\alpha} = \sum_{\mathbf{c}\in C_{\boldsymbol{\sigma}}} \left(\left(\frac{|\mathbf{c}_{\boldsymbol{\sigma}}|}{M} - \prod_{d=1}^{k} \frac{a_{d,c_d}(\boldsymbol{\sigma})}{M} \right) \mathrm{softmin}_\alpha(f_1(\mathbf{c}_{\boldsymbol{\sigma}}), \ldots, f_k(\mathbf{c}_{\boldsymbol{\sigma}})) \right)\,,$$

where

$$\mathrm{softmin}_\alpha(x_1,\ldots,x_n) = \sum_{i}^{n} x_{\min} + \alpha(x_i - x_{\min}), \qquad x_{\min} = \min(x_1,\ldots,x_n).$$

The parameter α controls how strongly one weak correspondence value should influence the overall correspondence value of a single community edge, and thereby its contribution to modularity. A value of 0 amounts to a hard minimum function, whereas a value of 1 leaves the original sum function unchanged.

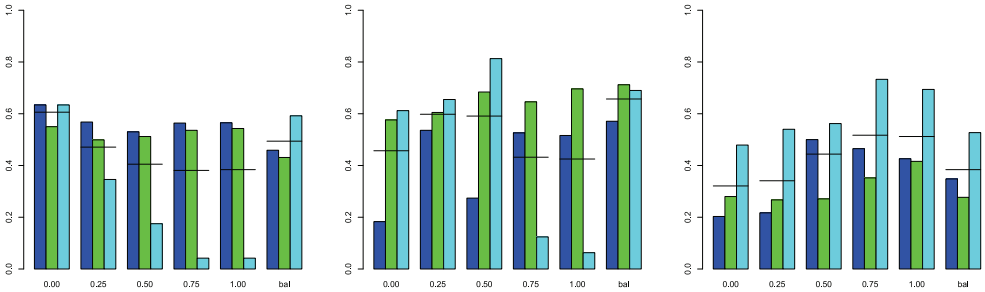
4.5.4.1. Choosing α

Since neglecting information from other domains is generally detrimental to overall performance, α should be set only as low as necessary. In practice, I have tried out several values for α per graph (0, 0.25, 0.5, 0.75, 1.0), retrieved the minimal balance over all domains, and chosen the result with the highest minimal balance.

Often, comparing the balances on the first 20% of the clustering already serve as a good predictor for which α value will be the best on the whole dataset. Searching for a more systematic way of finding α, another approach evolved that will be introduced in the next section. Before, however, let us look at the effects of dampening.

4.5.4.2. Results

For evaluation, I have chosen three typical user-expansion graphs that respond best to different α settings: H_{low} (Figure A.7(a)), H_{medium} (Figure 4.12(a)), and H_{high} (Figure A.7(b)). Figure 4.14 shows the balance in each domain, for each α value, for all three hypergraphs. The highest minimum balance for the graphs is found at $\alpha = 0$, $\alpha = 0.25$ and $\alpha = 1$,



(a) Results for H_{low} (b) Results for H_{medium} (c) Results for H_{high}

Figure 4.14.: Balance per domain (blue=documents, green=users, turqois=tags) for three hypergraphs and dampened modularity with various values of α and balanced modularity. Vertical lines indicate average balance.

respectively. The compression (not shown), used as a quality measure for the clustering independent from modularity, mainly stays invariant, indicating that tuning balance does not disturb the identification of dense clusters.

Figure 4.15 shows the clustering results for H_{medium} in more detail for various values of α. Each row shows a visualization of the final community assignment, the development of bal for tags over time (since this is the domain creating the problems in the undampened version), as well as the dendrogram of the tag domain.

- The first row shows the connection between a single tag cluster, very low balance values and a dendrogram that describes a clustering process characterized by joining tiny elements into a single giant community.
- The second row shows the results for the optimal choice of α, 0.25 (Figure 4.14(b) shows that this value yields the highest minimum balance). We see discrete clusters, a better balance and a dendrogram that shows two big and one small cluster.
- The third row shows the results for $\alpha = 0$. Even though balance is even higher in the tag domain and we see an even more refined dendrogram, Figure 4.14(b) shows that now, balance for the document domain drops drastically.

The fourth row will be discussed in the next section. See Figures A.8 and A.9 in the Appendix for corresponding analysis charts of the other two hypergraphs. The effect of decreasing α on the synthetic test charts were also tested, see Figure 4.16. As expected, low values of α cause a slight decrease in performance. However, the corresponding benefits cannot properly evaluated on these datasets because no giant communities are created. To conclude, the above results show that

- Balance is a suitable measure to detect the emergence of giant communities, which cause long phases of very low individual balance values.

- Using the dampening approach, decreasing α can be used to remove the giant communities.
- Different hypergraphs have different optimal α values. Trying out different values and choosing the one which yields the highest minimum balance seems like a feasible approach, but is neither theoretically well-founded nor particularly efficient in real applications.

The next section will introduce an approach to remove the need for choosing a particular α.

4.5.5. Balanced Modularity

There is another way to look at the behavior leading to giant communities. It is indeed problematic that "sacrificing" correspondence for one domain while optimizing the two others may lead to superior modularity values. More specifically however, we can see that it is in particular the decaying correspondence of the *largest* involved community that is characteristic for the emergence of giant communities. So, an alternative to weighting the individual correspondence values per community towards the minimum is a weighting by the size (i.e. the number of incident edges) of the involved community. This way, a low correspondence value for the biggest community will drag down the contribution of the whole community edge without the need to explicitly address the minimum value.

4.5.5.1. Definition

Consider a community edge $\mathbf{c}_{\boldsymbol{\sigma}} = (i, j, k)_{\boldsymbol{\sigma}}$ for which the three involved communities contain 10, 10 and 20 edges, respectively, i.e. $a_i = a_j = 10,\ a_k = 20$. I propose a linear weighting scheme such that in this case, the individual contribution of the correspondence values if $\frac{1}{4}$, $\frac{1}{4}$ and $\frac{1}{2}$. This amounts to changing the correspondence part of the modularity function from

$$f_1(i,j,k) + f_2(i,j,k) + f_3(i,j,k)$$

to

$$f_1(i,j,k)\frac{a_i}{a_i + a_j + a_k} + f_2(i,j,k)\frac{a_j}{a_i + a_j + a_k} + f_3(i,j,k)\frac{a_k}{a_i + a_j + a_k}$$

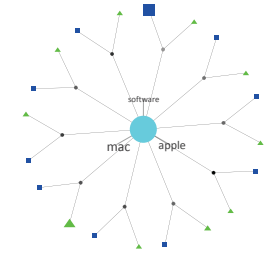
or, more generally, to

$$\sum_{d=1}^{k} f_d(\mathbf{c}_{\boldsymbol{\sigma}})\frac{a_{d,c_d}(\boldsymbol{\sigma})}{\sum_{d'=1}^{k} a_{d',c_{d'}}(\boldsymbol{\sigma})}.$$

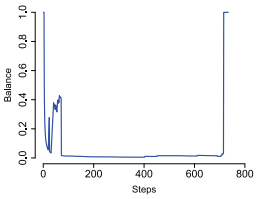
For the linear correspondence function, such a balancing results in a correspondence term of

$$\sum_{d=1}^{k} \frac{|\mathbf{c}_{\boldsymbol{\sigma}}|}{a_{d,c_d}(\boldsymbol{\sigma})}\frac{a_{d,c_d}(\boldsymbol{\sigma})}{\sum_{d'=1}^{k} a_{d',c_{d'}}(\boldsymbol{\sigma})} = \frac{k|\mathbf{c}_{\boldsymbol{\sigma}}|}{\sum_{d'=1}^{k} a_{d',c_{d'}}(\boldsymbol{\sigma})}.$$

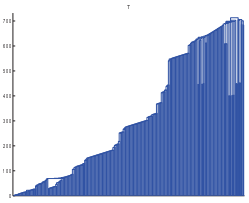
This leads to the following definition:



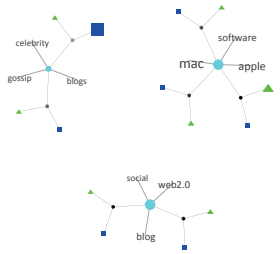
(a) Visualization for $\alpha = 1$



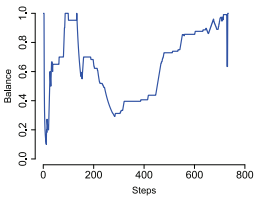
(b) Balance chart for $\alpha = 1$



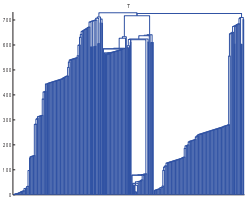
(c) Tag dendrogram for $\alpha = 1$



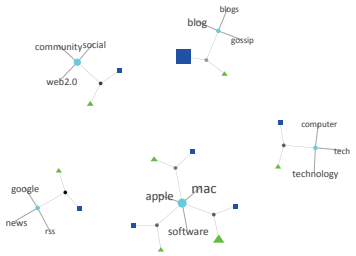
(d) Visualization for $\alpha = \frac{1}{4}$



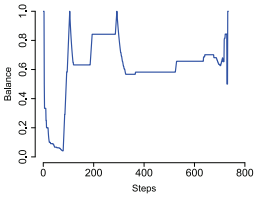
(e) Balance chart for $\alpha = \frac{1}{4}$



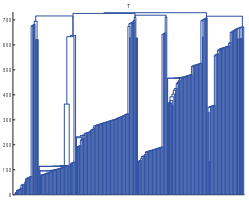
(f) Tag dendrogram for $\alpha = \frac{1}{4}$



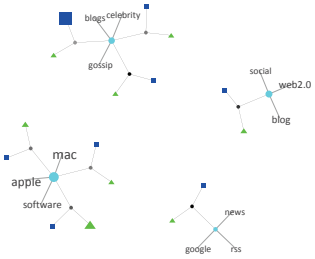
(g) Visualization for $\alpha = 0$



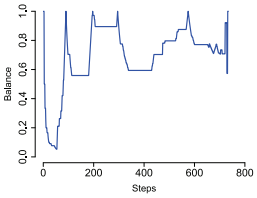
(h) Balance chart for $\alpha = 0$



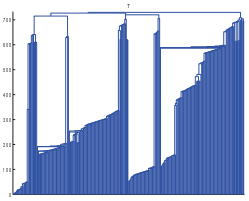
(i) Tag dendrogram for $\alpha = 0$



(j) Visualization – balanced



(k) Balance chart – balanced



(l) Tag dendrogram – balanced

Figure 4.15.: H_{medium} – results for the dampened modularity with various values of α and the balanced modularity

Definition 4.9 (Balanced Linear Multi-Partite Modularity) *Let the* balanced linear multi-partite modularity *of a 3,3-hypergraph H under a community assignment $\boldsymbol{\sigma}$ be defined as*

$$Q_{\text{MPbal|lin}} = \sum_{(i,j,k)\in C_1\times C_2\times C_3} (e_{ijk} - a_i a_j a_k) 3 \frac{e_{i,j,k}}{a_i + a_j + a_k},$$

or

$$Q_{\text{MPbal|lin}} = \sum_{\mathbf{c}_{\boldsymbol{\sigma}}\in\mathbf{C}_{\boldsymbol{\sigma}}} \left(\left(\frac{|\mathbf{c}_{\boldsymbol{\sigma}}|}{M} - \prod_{d=1}^{k} \frac{a_{d,c_d}(\boldsymbol{\sigma})}{M} \right) \frac{k|\mathbf{c}_{\boldsymbol{\sigma}}|}{\sum_{d=1}^{k} a_{d,c_d}(\boldsymbol{\sigma})} \right).$$

4.5.5.2. Re-Evaluating the Example

Going back to the initial example: Does the version that leaves the m "arms" with specific tags unmerged receive a higher balanced modularity than the one with a large, central community?

Replacing linear by balanced modularity, the correspondence term $(1 + \frac{1}{2} + \frac{1}{2})$ in Equation 4.4 becomes $\frac{3}{5}$, the term $(\frac{1}{2} + \frac{1}{2} + \frac{1}{m})$ in Equation 4.5 becomes $\frac{3}{4+m}$, and the term $(1 + 1 + \frac{1}{m})$ becomes $\frac{1}{2m}$. Already, it becomes visible that correspondence in the latter case converges towards 0 as m becomes large – the central tag community becomes larger, compared to the single document and tag communities, so its decreasing correspondence receives more weight. While a similar process takes place for the correspondence of the inner edges in the unmerged case, the outer edges are unaffected by the choice of m, which guarantees a positive correspondence value. Plugging in the numbers into the modularity computation, we obtain

$$Q_{\text{MPbal|lin}}(\boldsymbol{\sigma}^{\tau}) = 3\left(0.1 - \frac{2}{5m^2} + \frac{1}{8+2m} - \frac{1}{8m+2m^2}\right)$$

and

$$Q_{\text{MPbal|lin}}(\boldsymbol{\sigma}^{\tau+m}) = 3\left(1 - \frac{1}{m}\right)\frac{1}{2+m}.$$

$Q_{\text{MPbal|lin}}(\boldsymbol{\sigma}^{\tau}) > Q_{\text{MPbal|lin}}(\boldsymbol{\sigma}^{\tau+m})$ for all $m > 2$. It might be preferable to obtain this result for $m \geq 2$, but it seems an acceptable compromise that the balanced modularity favors merging two arms, as the size difference between the central tag community and the document and user communities has not yet grown that large. Since for large m, $Q_{\text{MPbal|lin}}(\boldsymbol{\sigma}^{\tau})$ converges to 0.3, but $Q_{\text{MPbal|lin}}(\boldsymbol{\sigma}^{\tau+m})$ to 0, balanced modularity can be considered a successful adaptation at least to the toy example: The solution maintaining the "arms" gets more stable the more such arms exist.

4.5.5.3. Results

Figure 4.14 shows, in the last column for each graph, the balance values obtained by the balanced linear modularity measure. The bottom rows in Figures 4.15, A.8 and A.9 show the visualization, tag balance charts and tag dendrograms obtained with this measure. A number of observations can be made from this data:

- For those hypergraphs which would require a dampening – low – α, i.e. H_{low} and H_{medium}, balanced solutions are found. While it is unclear whether higher balance is always better, what is important is that very low balance values are removed. This happens in both cases.
- For H_{medium}, it is possible to "over-dampen" by choosing $\alpha = 0$ which results in an unbalanced document domain (see the left-most bar in Figure 4.14(b)). This problem does not occur using balanced modularity optimization.
- For H_{high}, which has the most balanced results for $\alpha = 1$, i.e. the undampened version, numerical results (Figure 4.14(c)) look slightly worse than for the balanced version, compared to the undampened one. Even though the visualization of the balanced result in Figure A.9(j) arguably provides a successful overview of the dataset, this finding might encourage a certain caution in balancing graphs which do not need to be balanced.

Figure 4.16 completes the examination of the balanced, hybrid modularity: Its results on the synthetic datasets are not quite as good as the undampened version and resemble those of dampened modularity with medium α settings. This further strengthens the hypothesis that balancing should only be applied where necessary.

4.5.6. Conclusion

In this section, I have discussed a problem that occurred on real data, and provided, after a theoretical discussion of its origins, two solutions, the dampened and the balanced modularity measure. Even though the problem could be solved by the dampened modularity, it requires a parameter, implying a full clustering process for every candidate setting. Balanced modularity, on the other hand, works without a parameter, and performs comparably to finding a good α value, in particular if such a value would be low, i.e. balancing is required in the first place.

In practice, I would recommend clustering an unknown hypergraph without any dampening in the first place. If balance in one of the domains turns out to be critically low – in the examined graphs, giant communities were characterized by a balance below 0.2 – or the giant community is visually identified, I would recommend applying the balanced modularity. Only if this measure yields unsatisfactory results – which could not be observed on the, admittedly small, set of examples – dampened modularity can be used as a last resort to fine-tune the obtained results.

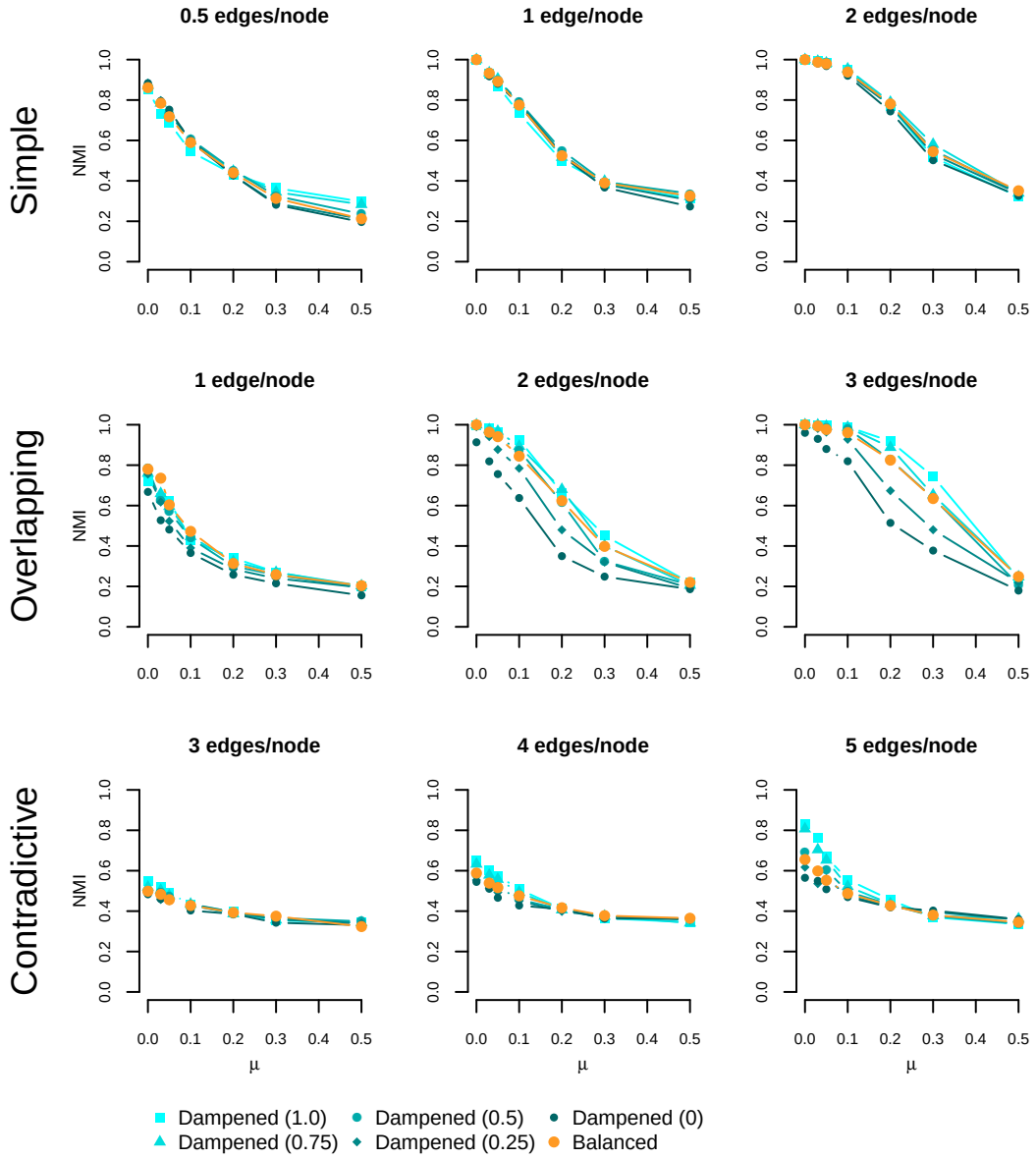


Figure 4.16.: Comparing the effect of dampening for different values of α vs the balanced approach. The underlying modularity measure is the hybrid multi-partite one (shown in light turqouise, as $\alpha = 1$ removes the dampening). Low values of α lead to a decay in performance, particular in the OVERLAPPING case. The balanced approach suffers some performance loss as well, compared to the undampened version, but not as severe. See Section 4.1.2 for a detailed explanation of this chart.

Table 4.1.: An overview about the introduced modularity measures and their capabilities.

	SIMPLE	OVERLAP	CONTRADICTIVE	Balanced	param.-free
Non-Partite	✓				✓
Coupled Bipartite	✓	✓			✓
Multi-Partite		✓	✓		✓
Mixed	✓	✓	✓		
Hybrid	✓	✓	✓		✓
Dampened	✓	✓	✓	✓	
Balanced	✓	✓	✓	✓	✓

4.6. Discussion

4.6.1. Summary

In the first three sections, it was demonstrated how increasingly higher-dimensional relations in the data in fact require higher-dimensional community detection algorithms. Also, the relevance of the theoretical challenges from Section 3.1.2 could be proven on actual data: Distinct community assignments, ambiguous correspondence relations, and different ternary generalizations of correspondence could all be shown to matter even in the rather simplistic examples.

While the first three sections basically build the way to the formulation of a truly multi-partite modularity measure, the last two sections describe the struggles of this theoretically motivated measure when facing real data. Hybrid modularity integrates modularities of different order, which turned out to be important after seeing the multi-partite modularity fail on the SIMPLE datasets. Balanced modularity finally resolves the problem of single giant communities, an undesired effect that could only be found on real hypergraphs.

Table 4.1 sums up the developments described in this chapter.

4.6.2. Conclusion

This concludes the chapter with the primary theoretical contributions of this thesis. Taking a step back, what happened in the last two chapters is that starting from a general problem statement and a number of abstract challenges, a method was devised and refined to the point where very specific situations have been analyzed and accounted for. It was my concern to document not only the final formulations of the modularity functions but the thoughts that led up to those formulations and my interpretation of the behaviour of these functions in various situations.

The next logical step is now to see how the devised methods perform on real data, and how their functionality can be put to use in application contexts. This is the topic of the next chapter.

5. Community Detection on Real Data

This chapter highlights possible applications of the previously introduced multi-partite community detection on hypergraphs obtained from real-world social bookmarking datasets.

Section 5.1: Use Cases and Sub Graphs considers possible use cases for community detection in social bookmarking datasets. These involve restricting the attention to the neighborhood of a particular item of interest; therefore these use cases correspond to patterns for extracting subgraphs around these items.

Section 5.2: Interactive Exploration of Community Structure introduces the interactive exploration tool `mpce` (Multi-Partite Community Exploration). It allows browsing k-partite hierarchical clusterings, i.e. the exploration of the full clustering tree instead of a static, single solution. It serves as an example of a navigational application as well as to visually compare the solutions of different community detection algorithms.

Section 5.3: Comparison of Community Detection Algorithms examines the practical differences between the different community detection algorithms when applied on the proposed use cases. This involves both the inspection of individual results and a more general analysis, examining average properties over many results.

Section 5.4: Network Analysis in a Real Social Network contains a report on a project with VZ.net, the company operating Germany's largest social networks StudiVZ, MeinVZ and SchülerVZ.

The software described in Section 5.2 is provided as a downloadable software package, `mpce` (Multi-Partite Community Exploration) – see Appendix A.1.4 for details.

Parts of this chapter have been published in Neubauer and Obermayer (2011).

5.1. Use Cases and Subgraphs

When we set out to explore the community structure of social bookmarking sets, the goal was to uncover the suspected information contained. Now, with the methods in place, how can they be used in practice to make that information relevant? Furthermore, how can we judge, in the absence of ground truth, whether a given community assignment is "good", or better than another one? These questions are "softer" than the question which algorithm performs better on a benchmarking dataset. Having concrete use cases however might facilitate such judgements. To answer these questions, two use cases will be introduced, and the corresponding solutions of different community detection algorithms will be explored. Those use cases are:

- "I want to get an overview over all documents I have assigned a given tag to, using tags from other users."
- "I want to get an overview of what a particular tag is all about."

The hypothesis is that these overviews can be generated by extracting relevant parts of the full bookmarking dataset, performing community detection on those parts and letting the users explore its solution. In the following, the subgraphs related to each use case will be defined. In the next section, I will propose a tool for the visualization and exploration of the obtained solutions, and the third section will conclude this topic by an in-depth examination of those results.

5.1.1. User/Tag Expansions

5.1.1.1. Definition

The subgraph defined here is related to the use case of exploring all of a user u''s documents that she has tagged with a tag t'. Assuming that all edges in the social bookmarking dataset are in E, these documents are given by

$$D(u', t') = \{d : (d, u', t') \in E\}.$$

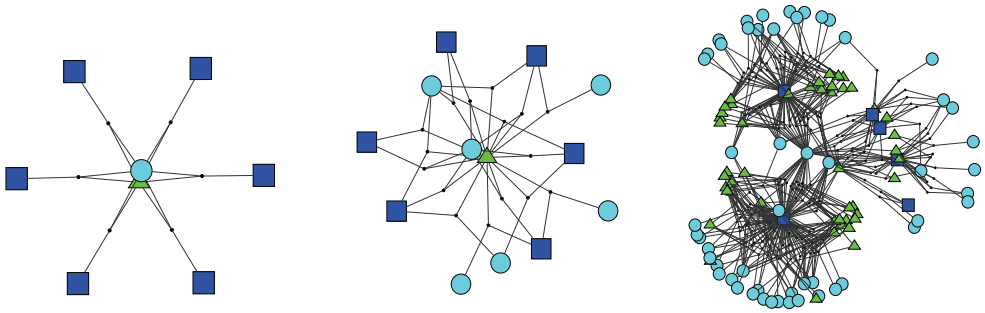
One possibility to extract related edges might be to simply retrieve

$$E_1(u', t') = \{(d, u', t') \in E : d \in D(u', t')\}.$$

Figure 5.1(a) shows a graph created for the tag "programming" and a user who has tagged only 6 documents with this tag. We might also fetch the edges concerning all other tags that user u' has applied on those documents:

$$E_2(u', t') = \{(d, u', t) \in E : d \in D(u', t')\},$$

shown Figure 5.1(b). However, a much richer description of the tagged documents can be gained by using the social aspect of social bookmarking systems, retrieving the edges with *all* users' tags for the documents in question.



(a) E_1: A user, the tag "programming", and 6 documents tagged "programming" by the user (6 edges)

(b) E_2: All other tags also used by the user on the initial 6 documents (16 edges)

(c) E_{DU}: All other tags (52) given by users (47) to the initial 6 documents (216 edges)

Figure 5.1.: Real data: Expansion around a user/tag pair

Let the *user/tag expansion* around a user u' and a tag t' be defined as the hypergraph given by the edges

$$E_{DU}(u', t') = \{(d, u, t) \in E : d \in D(u', t')\}.$$

Figure 5.1(c) demonstrates how this adds some complexity even for the small sample dataset. At the same time, some structure already becomes evident from the layout in the graph, grouping four of the documents together and isolating the remaining two. It remains to be seen how community detection can help to maintain this additional information while reducing the complexity of the representation.

5.1.1.2. Multi-Partite p-Cores

A recurring practical problem when dealing with subgraphs obtained by e.g. user/tag expansions is constraining their size, given the runtime constraints of community detection algorithms discussed in the previous chapter. One way to reduce the size of a subgraph is to remove the "outer" nodes, restricting analysis to those elements which are more strongly connected to other elements in the graph. This inner part is known as the "core" of a graph.

A p-core (Seidman, 1983) is a subgraph of a graph G containing all elements that are connected to at least p elements which are also in the core. It can be computed by repeatedly removing all elements with degree $< p$ until no more elements are removed. p-cores have been used, for example, for the decomposition of large networks (Alvarez-Hamelin et al., 2005). In Hotho et al. (2006b), a p-core of a tagging dataset is used for testing methods requiring a highly connected graph. Here, I generalize the concept of p-cores to the multi-partite case, requiring the all elements of a multi-partite p-core are connected to at least p elements *from each other domain*, i.e. each documentsin a multi-partite 2-core needs to be connected to at least two tags and two users. In (Neubauer and Obermayer, 2008),

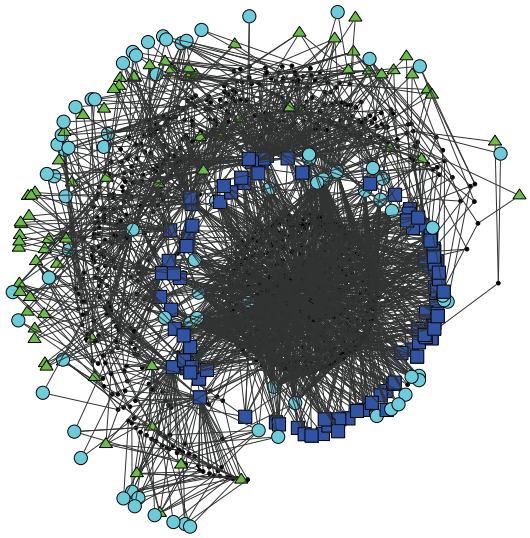


Figure 5.2.: The 2-core of a user/tag-expansion

we define an even more general definition of cores. It allows to specify that, e.g. each documents needs to be connected to n tags and m users. Since these refinements however did not turn out to be practically relevant, they are not formally defined here. The multi-partite p-core, however, has regularly been used during the inspection of various subgraphs and appears to be a suitable way of reducing a graph's size while keeping the gist of its contents. Figure 5.2 shows a 2-core of a user/tag-expansion, displaying the characteristic lack of "dangling" nodes on the borders of the graph.

5.1.2. Constrained Tag Expansions

The second use case is to quickly obtain an overview over the contents associated with a particular tag t' without paying attention to a single user in particular. The simplest way to obtain an associated subgraph might be to simply obtain all edges $e \in (d, u, t') \in E$. However, this will create widely differing graphs depending on the usage of t'. While the multi-partite p-cores introduced above provide a way of pruning the obtained graphs to more manageable sizes, I here propose here an alternative way for controlling the subgraph's size. The goal is to influence it during the creation of the graph already. It has various benefits: From a performance point of view, it provides more fine-grained control over the actual number of edges and elements, enabling better prediction of the runtime required for community detection. From a usability point of view, the resulting clusterings will be more uniform, providing more consistent visualizations to the user over different tags. Finally, from a methodological point of view, being able to produce many graphs of rougly the same dimensions will allow for a more expressive quantitative comparison later on.

The "constrained tag expansion" aims to fulfill these requirements. It is defined via the four parameters t', n_T, n_D, n_E. Casually speaking, we first find the top n_D documents

tagged with t'. Then the top tags associated with those top documents are identified (creating the union of the top n_T from each document), and finally edges are added user by user until we pass the threshold of n_E edges.

More formally, we start by retrieving all documents

$$D(t') = \{d : \exists d, u, t' \in E\}.$$

Let $D(t')[n_D]$ be the n_D documents in $D(t')$ tagged by the most users. Now, let us obtain the top tags for each of those top documents.

$$T(d') = \{t : \exists d', u, t \in E\}$$

Let $T(d')[n_T]$ be the n_T tags used on d' by the most users.Combining the tags obtained in this way leaves us with a set of top tags around our initial tag t':

$$T(t') = \bigcup_{d' \in D(t')[n_D]} T(d')[n_T].$$

This set can contain up to $n_D \cdot n_T$ tags, although it is more likely that some documents share top tags and a smaller number is obtained. Having identified both the top documents and the top tags, the edges between them are given by

$$E(t') = \{(d, u, t) \in E : t \in T(t') \wedge d \in D(t')[n_D]\}.$$

We might be done now, but we additionally want to control the number of edges in this graph. This can be achieved by incrementally adding edges, grouping edges by the users that have created them and adding them by those groups until the maximum number of edges is reached. Since the goal is to find the relations between tags and documents, users having tagged several documents are preferable. All users having tagged the top documents with the top tags can be obtained

$$U(t') = \{u : (d, u, t) \in E(t')\}.$$

These users are sorted in descending order based on the number $|D(u')|$ of top documents they have tagged:

$$|D(u')| = |\{d' : (d', u', t) \in E(t')\}|.$$

The constrained tag expansion is then defined by the set of edges

$$E(t')[n_e] = \bigcup_{u' \in U(t')} (d, u', t) \in E(t') \ ,$$

where $U(t')$ contains the minimum number of users taken from the ordered list that is required for $|E(t')[n_E]| \geq n_E$.

This construction may appear a bit artificial, but it is the result of several iterations of subgraph constructions, trying to achieve graphs of comparable shape and size across many tags while ensuring real-world applicability. As we will see later on, the obtained graphs are simple enough to be clustered in mass, but complex enough to reveal differences between different algorithms.

5.2. Interactive Exploration of Community Structure

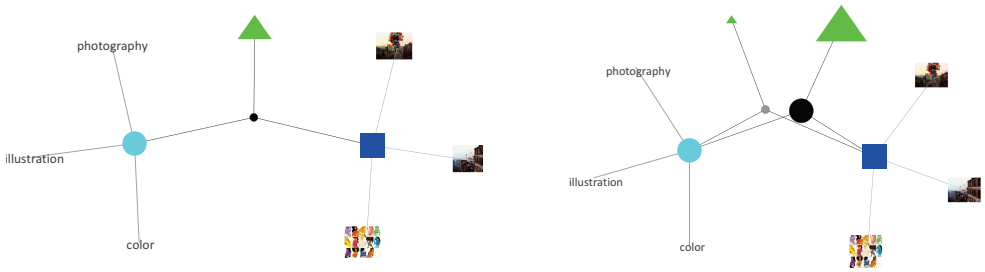
This section introduces the tool `mpce` (Multi-Partite Community Exploration) which interactively visualizes k-partite hierarchical community assignments, and demonstrates its output on a sample hypergraph.

5.2.1. Multi-Partite Community Exploration

`mpce` is a piece of software written in Java on top of the Processing (`processing.org`) framework, which facilitates the creation of interactive, animation-based programs. Besides abstracting graphics and animations, one of its advantages is the abundance of existing plug-ins. Here, the dynamic graph visualization is implemented using the Traer Physics plug-in (`murderandcreate.com/physics`). Using this plug-in, the graph layout is rendered in real-time as the user interacts with the system, with nodes modeled as particles, edges modeled as springs and non-adjacent nodes kept apart by simulating repulsive forces. Other functionalities include exporting into PDF files, which has been used for the screenshots in this section.

`mpce` exploits the hierarchical clustering generated by bottom-up community detection algorithms. Instead of just considering the optimal spot in-between as returned by community detection algorithms, `mpce` lets users undo this development in a step-wise fashion. This can be compared to using a file system explorer, starting from the root directory. The root directory corresponds to the root in the clustering tree, where each element belongs to the same community. By expanding it, it is split into the two subcommunities it was merged from.

Additional complexity comes from the fact that users are exploring not one, but three trees at the same time, one for each domain. As introduced in Section 3.2, the output of k-partite community detection algorithms is a k-partite hierarchical clustering: At each step, this clustering describes the merging of two communities in one domain. Users can either



(a) $\boldsymbol{\sigma^{253}}$ (b) $\boldsymbol{\sigma^{252}}$

Figure 5.3.: View before (a) and after first split (b). A set of 256 nodes has undergone 253 binary joins, resulting in one community per domain. Accordingly, (a) represents the community assignment $\boldsymbol{\sigma^{253}}$, i.e., the top of the clustering trees, and (b) represents $\boldsymbol{\sigma^{252}}$.

follow the order of the clustering – by pressing the space key, the next community according to the clustering is split. Alternatively, they can click on any community to split it even out of turn.

Figures 3.1(a) through 3.1(e) on page 49 have been produced using `mpce`. Since users would start from the final community assignment with all elements in one community per domain, repeatedly pressing space would take the user from 3.1(e) to 3.1(a).

More formally, in the terminology of Section 3.2:

- Each state of the tool visualizes a k-partite community assignment $\boldsymbol{\sigma}$.
- When using the default order (by pressing space to perform the next split), the community assignments are taken from the k-partite hierarchical clustering Σ given as the input, starting with the last assignment $\boldsymbol{\sigma}^{N-k}$ for N nodes and k domains and decreasing the index at each step.
- When splitting communities manually, a new community assignment is created, copying the current one and changing it by finding the next split of the current communities according to Σ.
- Communities are connected by edges represented as black or gray circles. These correspond to community edges $C_{\boldsymbol{\sigma}}$ induced by the currently displayed community assignment $\boldsymbol{\sigma}$. In all cases shown here, a minimum relevance of $\alpha = 0.2$ has been set for better readability, such that the actual set of community edges is given by $C_{\boldsymbol{\sigma},\alpha}$.
- The size of each circle is scaled by the number of edges $|c_{\boldsymbol{\sigma}}|$ of the community edges $c_{\boldsymbol{\sigma}}$ it represents. The color scales with its density $\tau(c_{\boldsymbol{\sigma}})$, with black indicating the currently highest value.

5.2.2. Example

Let us explore the user/tag-expansion around a the tag “color” and a user who has used this tag on 101 documents from the Visualize.us dataset (see Section 1.3.4). There are 76 users

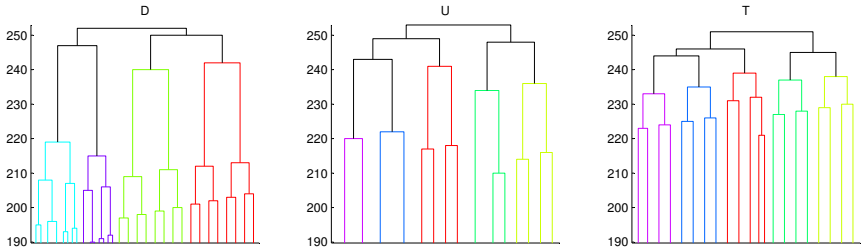


Figure 5.4.: Tree state at $\boldsymbol{\sigma}^{\mathbf{243}}$, showing the top quarter of the clustering trees. At each y-position, exactly one join takes place. Here, the highest 11 joins have been reversed - the resulting communities are encoded by colour.

who have used 79 different tags on those documents in altogether 1035 assignments.

Figure 5.3(a) shows `mpce`'s initial output after processing the output of the community detection process (see Table 3.1 on page 46 for the general structure of this output). The "last" (taking a bottom-up point of view) community assignment $\boldsymbol{\sigma}^{\mathbf{253}}$ is shown, i.e., all elements from the same domain have been assigned to a single community. Now, we can gradually descend the clustering trees. Proceeding one step from $\boldsymbol{\sigma}^{\mathbf{253}}$ to $\boldsymbol{\sigma}^{\mathbf{252}}$ results in the last join of the 3-partite community assignment being reversed, which, as we can see in 5.3(b), has been a join of two user communities.

How should these visualizations be read? First of all, communities can be annotated with their most important elements – the ones with the most edges –, as done here for images and tags. The size of element and community edges represents the number of contained elements/edges (sizes are normalized at each step and not representative across different steps). This information should enable the user to get a quick idea of the overall structure and where to find items of interest, particular as the structure further unfolds.

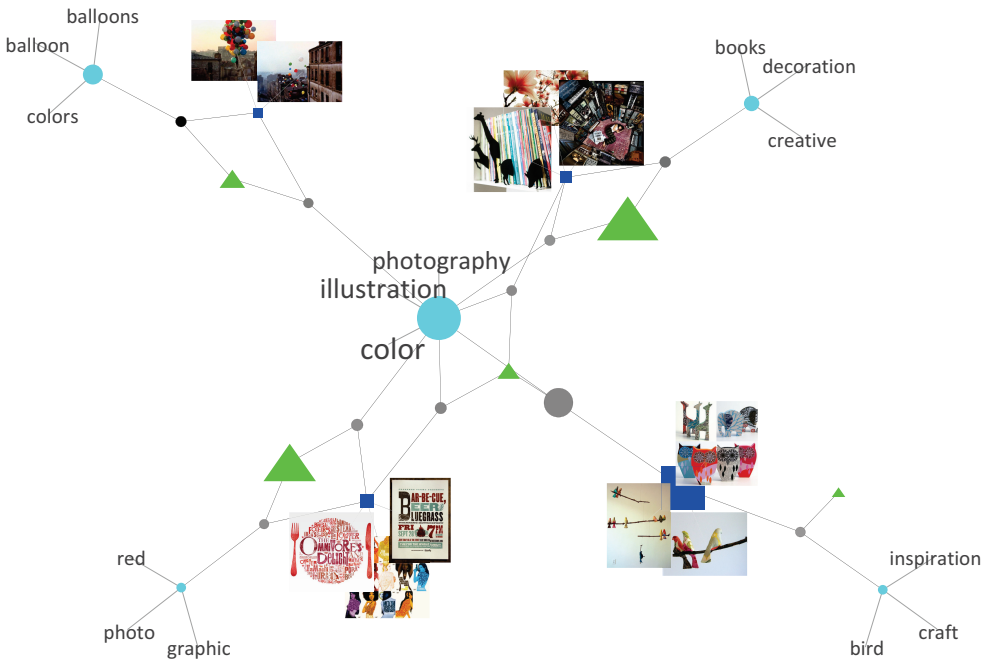


Figure 5.5.: Interface state at $\boldsymbol{\sigma}^{\mathbf{243}}$, i.e. 11 binary community splits – the single communities have been split into 4 document, 5 user, and 5 tag communities. community edges are plotted for relative importances $\alpha \geq 0.2$.

Figure 5.5 shows the output 11 expansion steps into the community structure. Documents, users, and tags have been split in a pretty balanced fashion, creating four "arms", each with a distinct triple of strongly connected document/user/tag communities, but still connected to a large central tag community (plus, for two of them, connected to a small, central user community).

It is well possible that one of the documents of the upper left document community C_d (the balloons) is in fact tagged with one of the tags from the lower left community C_t(for example, "photo"). If this was the case, there should be a connection via a small, gray community edge between the corresponding communities. This is where enforcing a minimal *relative importance* of $\alpha = 0.2$ comes into effect: if there is a non-empty community edge (C_d, C_u, C_t) (for any user community C_u), its relative importance is smaller than 0.2 for any of the involved communities, i.e., not drawing this connection neglects less than 20% of incident edges, for all communities. In short, α can be adjusted to achieve a trade-off between completeness and clarity in the resulting visualization. I found that a value of $\alpha = 0.2$ yields well-readable results in many cases.

Figure 5.4 finally shows the top quarter of the clustering trees representing the hierarchical clustering under exploration, at the state displayed in Figure 5.5. Starting from the top of the tree, we have reversed the ten highest joins implied by these trees. Figure 5.3(b) showing the user community being split first corresponds to the root of the middle tree (the "U" dimension) being higher than the roots of the other trees. Exploring the next step of the induced community structure would mean undoing the 11-highest join in the trees. Of course, explorations beyond the linear order implied by these trees are possible. A user might be interested in a particular community and expand it manually, overriding the next step implied by the trees, basically expanding a subtree earlier than it would have normally been.

This concludes the example of how community assignments, and in particular a hierarchical set of community assignments, can be visualized, and how such a visualization might be useful for navigational purposes. The remaining question is whether different algorithms result in meaningful differences with respect to these visualizations.

5.3. Comparison of Community Detection Algorithms

Let us first compare a pair of individual results on a user/tag expansion, demonstrating clear differences in the nature of resulting visualizations. In the next step, constrained tag expansions are considered, first again on an individual example, then – as made possible by the uniform size of the subgraphs – in a more quantitative fashion.

5.3.1. Examining a User/Tag-Expansion

Here, the results of non-partite and coupled biparite community are compared. Using the Delicious dataset, a user/tag expansion has been created like the one shown in Figure 5.1 – using also the tag "programming", but bookmarks of a different user. Instead of six, this user has tagged 261 documents with "programming". In order to obtain a feasible dataset, the graph is reduced to its elements which are connected to at least three different elements from each other domain (the multi-partite 3-core, see Section 5.1.1.2). This results in a final dataset of 182 documents, 910 users and 250 tags linked by 17401 edges.

5.3.1.1. Results

Figures 5.6(a) and 5.6(b) are the results of applying non-partite and the (argmax-based) coupled bipartite modularity optimization on this example. Again, only community edges of relative importance $\alpha \geq 0.2$ are displayed.

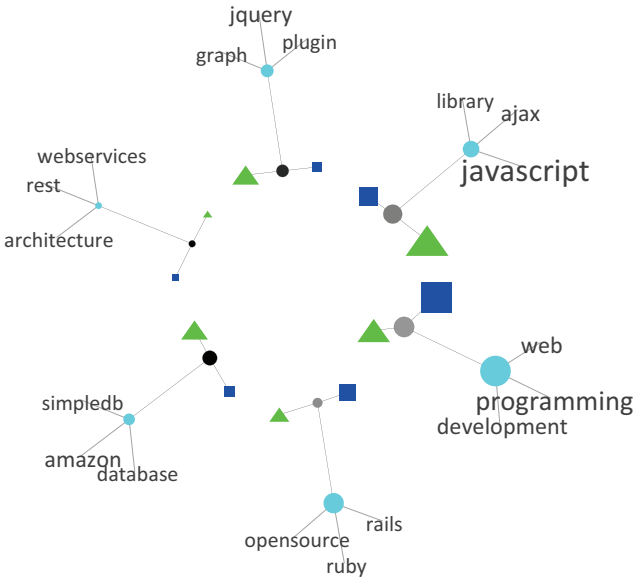
It must be noted that optimization of the coupled bipartite modularity runs for several hours whereas applying Clauset's fast optimization on the non-partite graph requires only seconds. All the more, it is crucial to determine if this effort, or further work on the optimization, is rewarded by better results.

The most striking difference between the two solutions – next to the different number of communities – is that the non-partite method yields isolated triples of document/user/tag communities, whereas the coupled bipartite solution also contains relevant community edges connecting elements from different triples. More concretely, even though four triples of strongly connected communities from each domain exist in the coupled bipartite version, each document and tag community is also connected to the central user community.

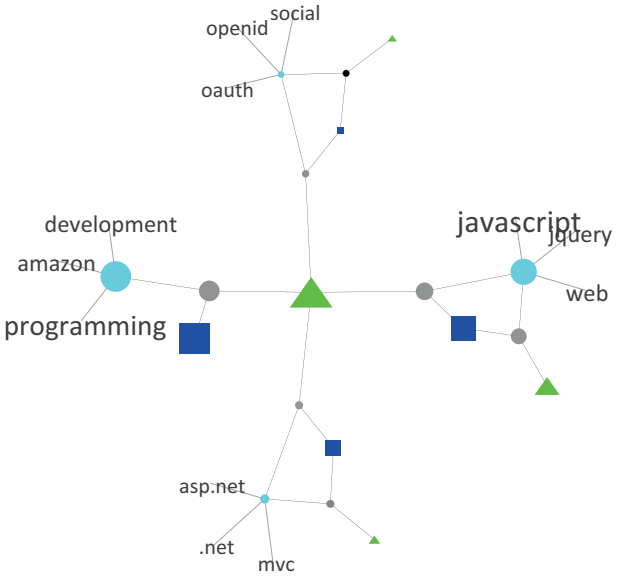
5.3.1.2. Evaluation

Which community assignment is better? In the absence of ground truth that could give us a quantitative answer, this issue is up to discussion and different positions can be held. I would, however, like to argue in favor of the result produced by the coupled bipartite approach.

First of all, there is *some* quantitative evidence after all. Examining the amount of edges removed by neglecting edge communities of insufficient relevance, it turns out that 2769 edges or ˜16% of edges are removed for the coupled bipartite solution, whereas 10284 edges or ˜59% are removed with the same cut-off threshold α for the non-partite solution. This is a problematic measure to use by itself: It could be optimized by the trivial solution of joining all elements into the same community (yielding a single edge community and 0%



(a) Communities found by the non-partite approach



(b) Communities found by the coupled bipartite approach

Figure 5.6.: Communities detected in a user/tag expansion around “programming” in Delicious ($p = 3$, $\alpha = 0.2$)

lost edges), and also the different numbers of communities play a role. Accounting for these factors would conceivably lead to a measure similar to modularity. Still, the big difference between these values might indicate that the coupled bipartite approach actually captures some essential structural properties that the non-partite approach does not.

Our knowledge about the algorithms' behaviour on synthetic data further supports this suspicion: The non-partite approach is forced to group documents, users, and tags into shared communities, not being able to tell the difference between them. The large number of suppressed edges could be caused exactly by edges diffusely distributed across the borders of these artificial communities, indicating that the found distribution is due to the limitations of the applied model and not due to the underlying structure of the data. Opposed to that, the coupled bipartite approach possesses the flexibility to model edges beyond closely linked community triples. Like this, different pairs of document/tag communities can all be strongly connected to the central user cluster and yet remain unmerged.

This flexibility allows for what in my view makes the more qualitative argument in favor of the coupled bipartite approach: While both community assignments seem to make sense in terms of the grouped tags, the coupled bipartite result additionally reveals insights between the different communities, telling an additional "story" about the data:

- There is a community of documents mainly about backend programming with tags like "programming", "development", and, not displayed here, "webservice", "database", etc. The corresponding document and tag communities (left) are connected to the central user community ("programmers") in the middle via a large community edge.
- There is another bunch of documents, largely tagged with tags like "javascript", "jquery" etc – i.e., GUI/design-oriented technologies. The rightmost community edge connects this document/tag community pair to the rightmost user cluster ("designers").
- The center/right community edge is what makes things interesting: It connects the central community of "programmers" to the GUI-related right communities; without, however, there being a corresponding community edge from the "designers" to backend technologies.

So we started by trying to organize our bookmarks, and ended up learning something about the users connected to those bookmarks. The bulk of users is interested in the topic of the central tag, programming, and part of this interest includes frontend technologies. Another set of users that is only interested in that frontend part. If we were, e.g., interested in learning more about design, further exploring these users' bookmarks might be a good next step. Although these insights may not appear spectacular, they do seem plausible and reveal another level of structure of the underlying data.

This may indicate that an individual treatment of the different domains can lead to qualitatively new insights, compared to a simpler treatment or complete ignorance of the user dimension.

In the following, an example is presented that shows relevant differences between four different detection algorithms, as well as a more quantitative examination of these differences.

5.3.2. Examining Constrained Tag Expansions

The example in the previous section has provided evidence that the theoretical considerations stated in Section 3.1.2 do have practical impact, and that an algorithm taking them into account leads to qualitatively different solutions than one which doesn't. However, this example had two shortcomings: First of all, meaningful differences could only be presented between the non-partite and the coupled bipartite case (multi-partite solutions have therefore been omitted), and it focussed on a single hypergraph, making the evidence rather anecdotal.

These shortcomings will be fixed here: Four different algorithms will be examined on various examples. Let us briefly review these algorithms to make clear their individual relevance.

Non-Partite Modularity (Section 4.1) – the baseline algorithm that does not take into account the partiteness of the source data,

Coupled Bipartite Modularity (Section 4.2) – the algorithm which takes into account partitions, but does not work on "real" tripartite data structures,

Hybrid Multi-Partite Modularity (Section 4.4.3) – the algorithm which uses tripartite modularity unless in cases that are identified to be problematic, in which it switches back to using the optimal step provided by the bipartite one, and

Balanced Multi-Partite Modularity (Section 4.5.5) – which extends the hybrid multi-partite modularity by adapting the correspondence function to avoid the occasional pathological giant components the former occasionally produces.

Using constrained tag expansions, another single example will be examined first that exemplarily highlights the characteristics of all involved algorithms. In the second part, numeric results based on the examination of 250 randomly chosen tags will be presented that emphasize the generality of these differences. All expansions discussed here were created for the top 30 documents, top 5 tags per document, and a maximum of 1000 edges. Tags for which 30 documents or 1000 edges could not be obtained were not considered.

5.3.2.1. Individual Analysis

Let us start by examining a single example of a constrained tag expansion. Not only is it helpful for the future discussion to have an idea of how clusterings for a constrained tag expansion look like – this example also nicely highlights the differences between the algorithms involved.

The example I have chosen is the tag "Charity". This tag was used by 2866 users on 2342 distinct tags, creating 4210 edges. Only 154 users' edges relating to the top tags needed to be added to obtain the required 1000 edges. Please refer to Figures 5.7 to 5.10 for the solutions of the different algorithms – the actual clustering step used for the assignment displayed was once again selected by optimal minimum description length.

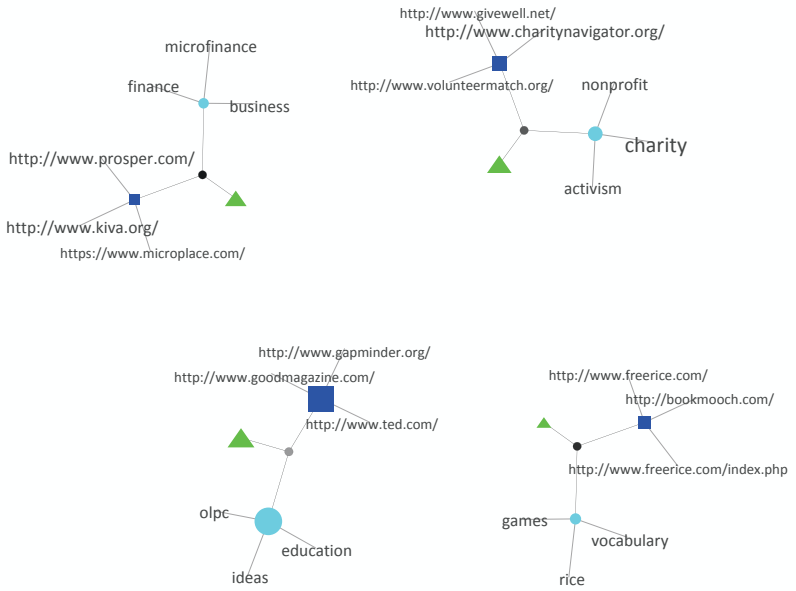


Figure 5.7.: Community structure for the constrained tag expansion around the tag “Charity”, produced by the non-partite modularity

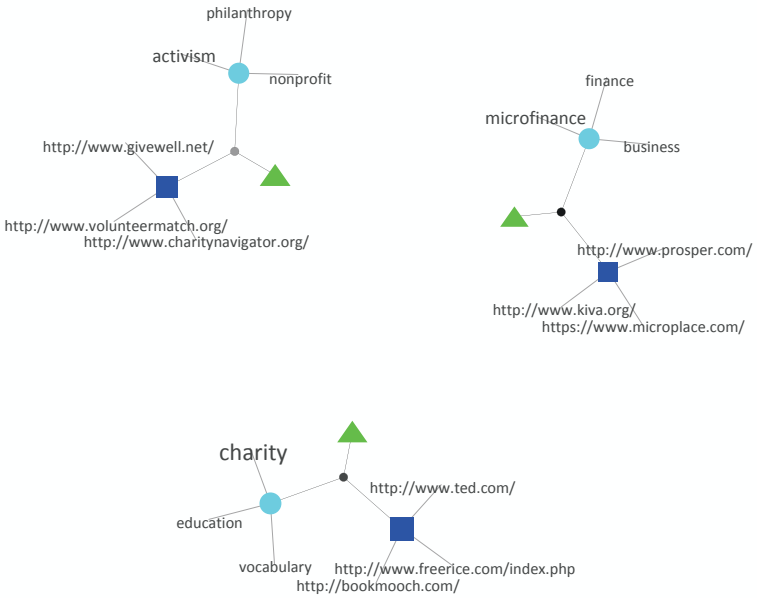


Figure 5.8.: Community structure for the constrained tag expansion around the tag “Charity”, produced by the coupled bipartite modularity

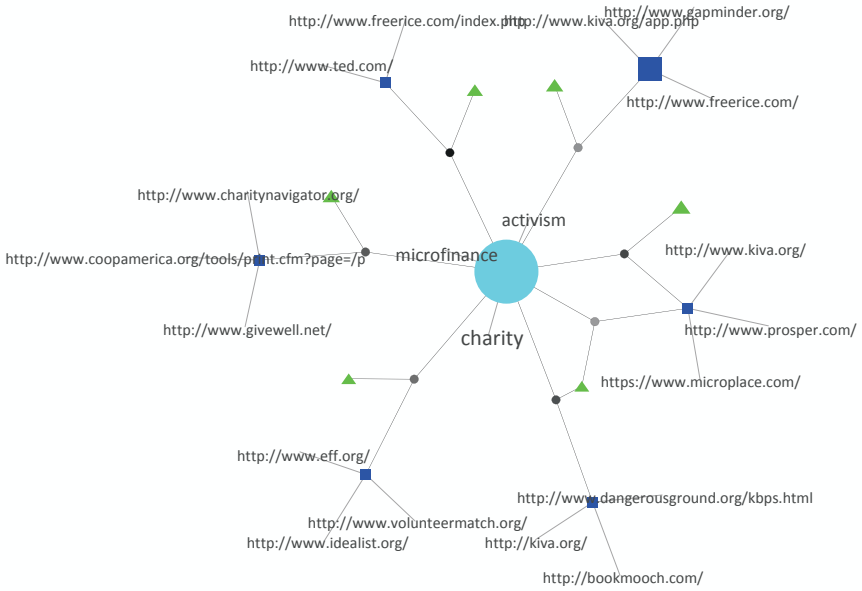


Figure 5.9.: Community structure for the constrained tag expansion around the tag “Charity”, produced by the multi-partite modularity

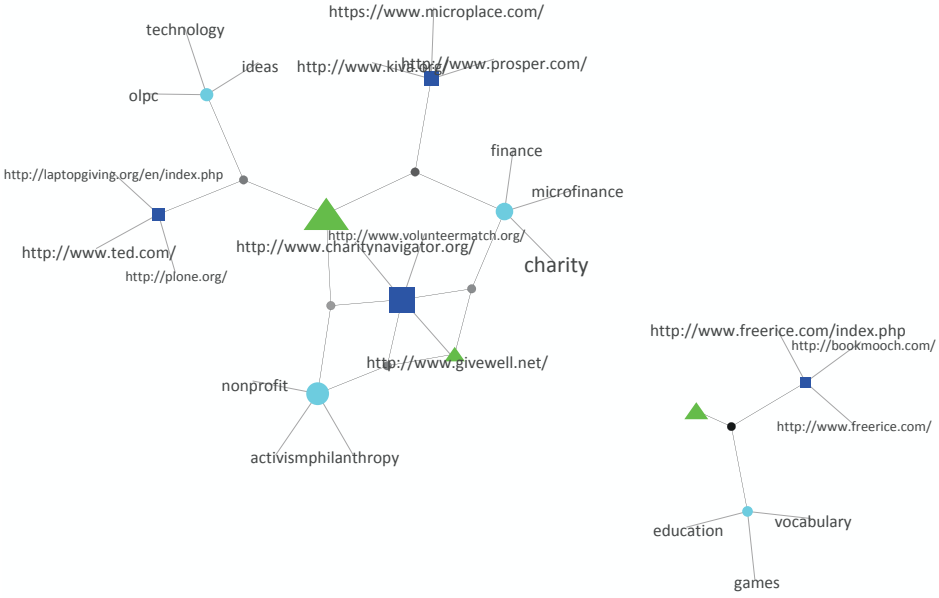


Figure 5.10.: Community structure for the constrained tag expansion around the tag “Charity”, produced by the balanced multi-partite modularity

Non-Partite The non-partite approach cleanly identifies four clusters. A nice feature of the tag "Charity" is the fact that these clusters, which most algorithms somewhat agree upon, correspond to distinctly identifiable topics. First of all, there is the microfinance community (top left corner). Then, there is a community of sites that seems to offer opportunities for charitable activities (top right corner). There is a community of sites more related to the bigger picture (bottom left corner), like the "one laptop per child" initiative or gapminder.org, a data visualization tool for various statistics related to measures of well-being in different nations. Finally, there's a more "consumer-oriented" community of sites (bottom right corner), involving freerice.com, donating rice for correctly answering vocabulary questions.

As in the previous example, the non-partite approach does a good job of identifying the individual topics. However, it fails again to create relations between the different clusters, for the same reasons discussed previously. This strengthens these earlier assumptions, and validates once again the need for more sophisticated approaches.

Coupled Bipartite Even though it was successful in identifying inter-cluster connections in the previous example, the coupled bipartite approach this time fails to do so. The only difference to the non-partite solution is that it has merged the two clusters of general information and consumer-oriented sites, which are arguably less specific than the two other ones.

Hybrid Multi-Partite The multi-partite approach displays the problematic behaviour discussed in Section 4.5.5: The optimal solution consists in collapsing all tags into a single community. Since the tags now cannot help in comparing sites, the remaining clustering is purely user-based. For example, freerice.com/index.php and freerice.com/ are clustered into two distinct communities – depending on which exact URL users have bookmarked. Clearly, this defies the purpose of community detection and can easily be identified as the weakest solution of all four.

The specific example of these two similar URLs stimulates further thought: Should the two have been merged into a single document during import? Or should the convergence of two such similar URLs into a single community be quantitatively evaluated as a quality measures? I believe however that it is legitimate to keep the two separate, and that the effect is overall too rare to meaningfully measure it on a larger scale.

Balanced Multi-Partite The (again, implicitly hybrid, since the switching heuristics of the hybrid approach are kept in place) balanced multi-partite approach was specifically designed to counter the problem of the giant community, and that is what it does. The four content communities found by the non-partite approach are successfully identified again. Beyond that however, there are some interesting inter-cluster relationships to be found. I will stick here to the most prominent feature: Three clusters are highly interconnected (and two of the previous user communities have been merged into a single one), and the one that remains isolated is the one with the more consumer-facing sites. In my view, this is an intuitively appealing solution, since users interested in financing microcredits or

Table 5.1.: Compressed size (MDL) relative to random community ordering

	Compressed Size	Std. Dev.
Non-Partite	38.7%	3.2%
Coupled Bipartite	38.2%	3.3%
Hybrid Multi-Partite	37.3%	3.5%
Balanced Multi-Partite	37.2%	3.6%

donating together probably form a different target audience than those stopping by at, e.g., freerice.com to play a sort of game in their browsers.

Beyond the concrete example, it is nice to see the characteristics of the different algorithms exposed so saliently. We can see the balancing playing an important role, and the multi-partite analysis it enables resulting in informative new connections between the triples of document/user/tag communities already identified by the simpler algorithms. Let us now proceed and find out whether these results are actually scalable.

5.3.2.2. Collective Analysis

Even though the exemplary analysis of individual results looks promising, it does not provide a solid basis for answering the question how well the algorithms are really doing on real-world data. I cannot examine all possible examples, and of those examples I have examined, I have admittedly presented ones with particularly compelling stories, and not the ones where only trivial differences could be determined between the different algorithms. Let us therefore proceed to a more quantitative methodology and compare the statistics of the clusterings for 250 tags randomly chosen among those which qualified, i.e. had sufficient documents and edges attached.

Compression Table 5.1 shows the average size of the compressed description of the adjancency tensor using the MDL scheme described in Section 3.2.3.2. Even though the numbers lie in a rather close range, it can be seen that the coupled bipartite approach consistently outperforms the non-partite one, while the multi-partite approaches outperform the bipartite one. The standard deviations are pretty high, but it is again hard to judge in how far these variations are due to the differences between graphs, as was the case for the synthetic datasets. Therefore, a sign test is applied in the same fashion as described in Section 4.1.2, counting the number of times one algorithm scores better than another. Both hypotheses (the bipartite modularity outperforming the non-partite one, and the multi-partite modularities outperforming the bipartite and non-partite ones) are highly significant (p<0.001).

Community Sizes Considerable effort has been spent trying to control the number of resulting communities. Figure 5.11 plots the distributions of the sizes of both the smallest and the highest number of communities. If a solution consists of 3 document, 4 user, and 5 tag communities, this would contribute to these charts as one count for 3 in the “minimum”

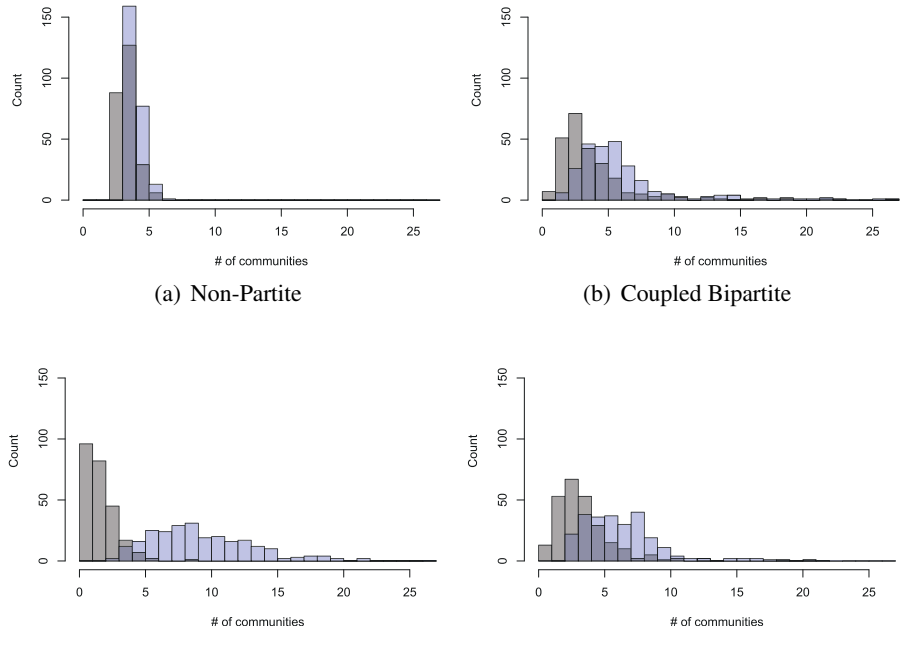
series and one count for 5 in the "maximum" series – i.e. in which domain the highest or lowest number of communities is found is not taken into consideration.

We find that the overwhelming majority of solutions, for the non-partite approach, has two or three communities. Even though this is a nice number of communities to start from for display and exploration purposes, it further strengthens the suspicion that the results obtained from this modularity function are largely driven by its algorithmic properties.

The coupled bipartite approach, in contrast, has a more plausible (i.e. wider) distribution, and even the occasional outliers with very many communities help underline the trust that an exceptional structure in the data will be mirrored by the solution.

For the hybrid multi-partite approach, it turns out that the tendency for giant communities, at least on graphs with the structure of the tag expansions, is more than anecdotal: Almost 100 out of 250 graphs end up with a solution that incorporates a single community in one of the domains. As we have seen above, this easily correlates with less than optimal clusterings in the other domains, and there also seems to be a tendency to create large numbers of communities in these other domains, as indicated by the sprawling distribution of maximal community sizes.

The balanced multi-partite modularity repairs this problem just as reliably as the hybrid one produces it. The number of single-community solutions almost drops to zero, and the overall distribution of community sizes looks very similar to the bipartite solutions.



(a) Non-Partite (b) Coupled Bipartite

(c) Hybrid Multi-Partite (d) Balanced Multi-Partite

Figure 5.11.: Histograms of minimal (gray) and maximal (blue) number of communities of all three domains, for different modularities.

5.3.2.3. Conclusion

The quantitative analysis confirms the anecdotal findings obtained from examining individual cases:

- Significant differences exist between the solutions of the different algorithms, both in terms of compression quality and the distribution of the number of communities obtained.
- The theoretically most sophisticated modularity measures, taking into account the full tripartite structure of the source data, outperform the simpler modularity measures in terms of compression quality.
- The coupled bipartite and the balanced multi-partite modularity seem to have the most "healthy" distributions concerning the number of modularities, assuming that both an almost fixed number of resulting communities (as produced by the non-partite modularity) and a high number of single-community results (as produced by the hybrid multi-partite modularity) are undesirable.

Whether the measures applied here are the last word in terms of judging multi-partite community assignments may be debatable – some alternative approaches will be discussed in the outlook. However, in the absence of ground truth, it may be legitimate to examine empirically good solutions, try to abstract their features, and then examine those features on a larger scale. Using this methodology, and regarding the specific type of graph examined, it appears reasonable to assume that the balanced multi-partite modularity is in fact the best modularity measure to use.

This concludes the main discussion of multi-partite community detection algorithms dealing with real data. The use cases introduced in the beginning of this chapter should have provided both an example of potential applications, as well as a basis to compare the different algorithms by. I would like to believe that the initial questions about relevant differences and real-world applications of the introduced algorithms can be answered positively. A more in-depth conclusion of the overall findings will be provided in the following chapter. Before, however, I will report the findings of an industrial collaboration, where some of the concepts introduced here (even though adapted to the concrete settings) could be applied.

5.4. Network Analysis in a Real Social Network

During the final stages of my dissertation, I've had the great opportunity to work in a joint project with VZ.net, the company running the largest German social network StudiVZ/MeinVZ (for university students/others) and SchülerVZ (a protected network for pupils of minor age). The goal of this pilot project was to explore how applicable and useful graph-theoretic techniques are when dealing with real-world challenges on graphs as serious as the ones created through the usage of VZ.net's services. I will discuss findings on the general properties of this graph as well as an experiment performed with bipartite community detection. Before, however, let me review the special privacy considerations that have to be considered in this setting.

5.4.1. On Privacy

Special care was taken in order not to violate the privacy rights of users during the examination of the data. The data provided for research was a dedicated export containing only the necessary information, i.e. an adjacency list of user ids – in an internal format that I couldn't use to bring up the corresponding profile page on the web – without further information in the case of the friend list, and a list of user/"Edelprofil" pairs (see below) for the community detection experiment. For comparison purposes, a subgraph was provided containing only users registered as living in Berlin, but this, as well, was performed by VZ.net employees without me getting access to personal information at any point. Furthermore, all of the data was processed on-site at VZ.net on their machines, and the only data that has ever left their network are the aggregate results shown here. Also note that in the community detection experiment, we only investigate the resulting clusters of public profiles, not the resulting clusters of users. Finally, even with these precautions in place, the SchülerVZ data were completely off-limits due to even higher privacy standards applied for the data of minors.

5.4.2. Connected Components of the Friendship Graph

The central feature of social network sites is the ability to "friend" other users: One user initiates a friend request, and if the other user accepts, the two users are now registered as friends, resulting in each one showing up in the other's "friend list", and gaining extended permissions regarding each other's data. Interpreting each friendship as an edge between two user nodes result in a "friendship graph", which in the export of the combined StudiVZ and MeinVZ data from early 2010 consists of around 8.5 million nodes with around 330 million edges, and 250000 users with 3 million users in the Berlin subgraph.

The first question was whether the friendship graph is connected. It almost is: the largest connected component contains 8564919 out of 8571918 users, which is more than 99.9%. In the Berlin subgraph, 232258 out of 233082 users (or more than 99.6% of users) belong to the largest components, with the next-largest components containing 29 and 11 users, respectively. 4762 and 561 users, respectively, were entirely unconnected. This result of almost each user being connected to each other user through a path of common friends was basically expected as a general property of sufficiently densely populated random graphs

Table 5.2.: Average number of users reached after number of hops, including standard deviation (both rounded)

Hops	Full graph		Berlin graph	
1	77	74	27	34
2	8181	9622	1281	1782
3	529257	510520	32535	28612
4	4808269	1734597	122252	38340
5	2893922	1668272	63620	41399
6	313498	637191	10998	20267
7	11267	44795	1336	6717
8	398	939	171	2273
9	41	57	22	432
10	4	7	7	61
11	1	1	6	19
12	1	0	6	7
13			4	2

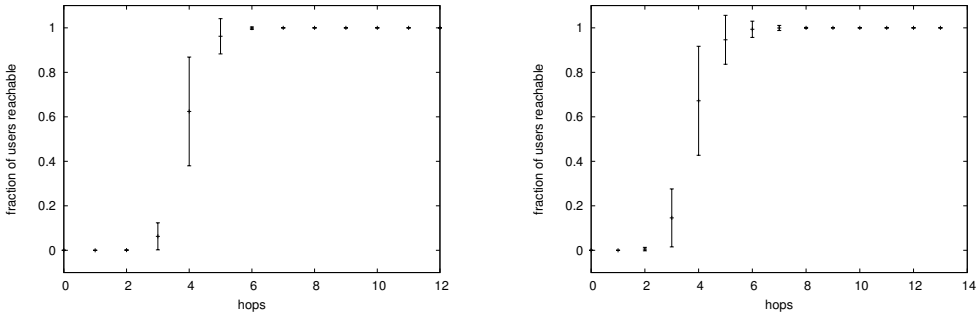
(see Chapter 2). Still, confirming this property was a useful first step in getting to know the tools (in particular the graph database Neo4J) and tricks required to deal with a graph of this size.

5.4.3. Characteristic Path Length of the Friendship Graph

In a next step, the characteristic path length – the average shortest path distance between two arbitrary nodes – of the two graphs was examined. For a single node, the shortest path length to each other node v_i is computed by performing a breadth-first search from that node, knowing that when a node v_j is encountered after n steps, this is the shortest path length to that node, since it would otherwise have appeared earlier. Ideally, the characteristic path length is computed by getting this distribution for each node and computing the true average. For the full graph, however, a single breadth-first traversal, as required to compute the distances from a single user to all other users, took hours to compute. Therefore, we just let the program run for over a week and used the data from the nodes completed during that time. These were mere 354 searches for the full graph and 68670 searches for the Berlin graph. The results are displayed in Figure 5.12, showing for each number of hops the fraction of all users that could be reached, averaged over all examined users, with error bars indicating standard deviation among users. The raw numbers can be found in Table 5.2.

5.4.3.1. Small World Properties

The most evident result is the low number of 'hops' (traversed edges) required to reach almost every user in the graphs. In both cases, after 6 hops, more than 99% of the user population can be reached in both cases, whereas after 4 hops, more than half of it can be



(a) Full graph (b) Berlin graph

Figure 5.12.: Cumulative Distribution of Path Lengths: Number of users reachable by distance

reached on average. The characteristic pathlength, i.e. the average of all observed pathlengths between pairs of nodes is 4.35 and 4.23 for the full and Berlin graph.

Networks like this with a very short characteristic path length in relation to the number of their nodes are called "small world" (Watts, 1999) networks. In a social context, this means, in simpler terms, that everyone knows everyone else through a short chain of friends. This is a well-known effect in social networks that has entered popular culture as the "Six degress of separation" phenomenon. To see how this happens, consider a network in which two people are connected if they are within 50 meters of each other at this very moment. This is a network without small world properties – to reach a person that is 50km away, you would have to hop through around 1000 different people (leaving aside issues of connectivity). If we now instead assume us connected to everyone we have ever been closer than 50m to, a single person in Berlin that has ever been to Munich will dramatically decrease the social distance between everyone in Berlin and Munich. Given that social ties possess a certain constancy over time, it seems safe to assume that online social networks are much closer to the second type of network than to the first one.

Still, there is a somewhat unexpected effect to be observed in the data: The distributions of the two graphs look very similar. Shouldn't we expect that two people in Berlin are on average connected by fewer connections than two people in Germany? This expectation can be formulated more formally: The small world property states that characteristic path lengths grow logarithmically to the number of nodes, i.e. the relation between characteristic pathlength p and the number of nodes N can be described as $N \sim b^p$ for a base b. One might assume that the full graph and the Berlin subgraph have the same properties and thus the same base parameter b, which should then however result in a lower characteristic path length in Berlin. Putting in the numbers for the two graphs, we get values for b of around 39 and 18 for the full and Berlin graph, respectively. The whole point of the small world property is that b does not change significantly as the graph grows, leading to only small increases in path lengths over time. Why does it change here? There are two possible perspectives on that:

- b can be interpreted as the average branching factor, or the number of previously unencountered users each new user in a path contributes. As we cut out the non-Berlin users from the graph, this number is bound to decrease – i.e. there are fewer people to reach, but there are also fewer people contributing shortcuts.
- While the first explanation would explain the change of b, the fact that the characteristic path length stays almost identical could also mean that geographic factors play such a small role in the friendship graph that sampling Berlin users is almost like randomly sampling users from the graph.

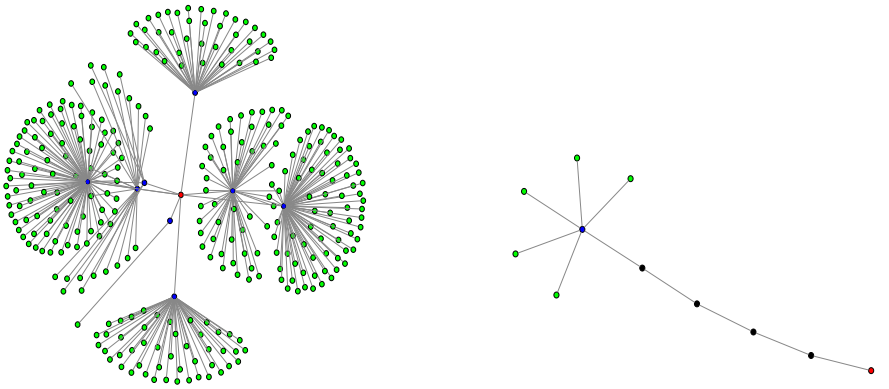
We have to be precise about the predictions of the small-world property: What it really says is that the full graph should have about the same base b now as it used to have when it was, say, only 250000 users large. What the above results now show in any case is that cutting out a subgraph of that size creates a new network that doesn't necessarily resemble that earlier-stage full graph.

Short-Range Effects Does this mean that regional proximity does not play any role for social distance? This certainly seems to contradict our intuitions about how we're more likely to meet people with shared aquaintances when in the same city! Taking a closer look at the charts, we find that there *is* a difference between the two which however is not captured by the global measure of characteristic path length and the number of hops needed to connect to almost everyone. This difference lies in the fraction users found through few connections. In the Berlin graph, 0.5% of the users can be reached via two hops, and 14.3% after three hops, whereas only 0.1% and 6.3% can be reached on the full graph. This means e.g. that the probability of being connected to a random person via only two friends becomes five times higher when only considering users in Berlin. It is conceivable that our personal experiences are shaped more strongly by those short-range effects than by the vast number of distant people that do not get much closer on average.

5.4.3.2. Pathological Path Lengths

On the other end of the path length spectrum, the maximum number of hops required to form a path between two users is 12 and 13, respectively. The distribution shows us that only a tiny fraction of users is further away than six hops from any other user, so we wondered how such extreme path lengths could possibly come into existence.

One might assume that the user creating these values are simply poorly connected users. However, it is not enough to be poorly connected – in order to arrive at extreme path lengths, you'd have to be connected to other poorly connected users which in turn only have poorly connected friends, and so on. Figure 5.13(a) shows a poorly connected, but "normal" user in red. Even though that user has only few friends, those friends have a more average number of friends, and the original user's average path length is at most only one hop larger than that of the more well-connected friends. We therefore examined single users with such high distances, and found a peculiar setup. Figure 5.13(b) shows the neighborhood of one of these users, representative for the other extreme cases as well. The user and those close to him (black) are not only poorly connected, they are in fact only connected to one or two



(a) A poorly connected, but "normal" user, directly connected to well-connected friends

(b) An poorly connected, irregular user, connected to regular users only through a number of other irregular users (black)

Figure 5.13.: Neighborhoods of different types of users with few friends. Such users (red), when connected to the giant component, at some point are connected to users (blue) with a high number of friends (green).

users at all, the ones required to form this long chain. Further examination of these users by the VZ employees showed that these users all seemed to have been created by the same person, maybe for fun or testing purposes, or for more malign intents – even though it is not clear how setting up this kind of connectivity could help such intents.

From a graph-theoretic point of view, this example confirms that it is very unlikely for a node in a small-world network to be all that far away from the core of the network – in the concrete case, one may have to make conscious efforts to create such a node. From an applications point of view, this finding follows up on the themes on spam detection discussed in Chapter 2: Non-standard behaviour creates non-standard graph properties so salient that single abusive users may be identified by looking at the global properties of a multi-million node graph. While this might be an interesting aspect to explore for operators of network-based services, it also underlines the importance of privacy considerations when dealing with networked data.

Conclusions On one hand, the results on path lengths confirm typical assumptions about the small-world behaviour of a social graph: The friendship graph of StudiVZ has a very short characteristic path length, and extreme deviations towards larger path lenghts point to anomalies in the underlying behaviour. On the other hand, the results also provide insights that correct two assumptions one might mistakenly hold about small-world graphs:

- Sampling from a graph is not the same as looking at the graph at an earlier point of time, and the statements made by the small-world property apply to the latter case only.
- Two networks may seem similar looking at their characteristic path length or the

number of hops required to reach almost every node in the network. The subjective experience of connectedness may however be much more closely related to the probability of meeting someone only few hops away. Since this is a feature that is overshadowed in those measures by the vast majority of only distantly connected people, it may be that subjective "small-world" experiences are largely decoupled from the measures associated with this term in network analysis.

These results bring up further questions that could be examined in a follow-up project:

- How well does the friendship graph follow the small-world property, i.e. how does the evoluation of b look like for different points in time?
- Do other regional subgraphs also exhibit the property of unchanged characteristic path length but higher probability of short-distance connections? It could be imagined that there are different classes of subgraphs, depending on absolute size but also regionally varying behaviours.

5.4.4. Bipartite Communities between Users and Profiles

As the second part of the project, we investigated the applicability of multi-partite community detection in a real-life setting. The platform provides the possibility of maintaining so-called Edel (noble) profiles – fan pages for public figures, events, products and the like. These profiles behave slightly different than normal user profiles – they don't show up, for example, as normal users' friends, but as items these users like. Also, they receive dedicated support by VZ.net, which, among other services, includes sorting them into a category tree. This contrasts them from, e.g., groups, which do not have to correspond to persons, events or such, can be created by any user, exist in much larger numbers, and are not categorized as finely.

5.4.4.1. Background

Bipartite community detection between users and entities like Edel profiles or groups has a number of interesting applications in the current setting. First of all, the identified clusters can serve for recommendations – if users already like elements of one cluster, chances are they will respond positively to recommendations from the same cluster. An automatic grouping can also facilitate a later manual assignment of categories to these groups, which may be interesting for commercial purposes. On the other partition, grouping users by similar interests without being restricted on users actually liking exactly the same things opens up possibilities for, e.g., friendship suggestions. In short, obtaining a sensible bipartite community structure could be the basis for future applications – if the results are actually sensible.

5.4.4.2. Experiment

The assumption is that an algorithm which creates sensible results on Edel profile networks – which we can test given the existing categorizations – might also provide valuable insights

into, e.g. the structure of the groups, where we have no such ground truth. We therefore decided to apply bipartite community detection to Edel profiles, in particular to those of major political Edel profiles, i.e. those of the seven largest German political parties or politicians from those parties, kept in a "Politics" subtree of the category tree. This is a clear-cut and easily interpretable domain, and has the advantage of keeping the numbers of connections at a managable level. As of early 2011, there were 446 Edel profiles and 260134 users, with 402259 assignments between the two sets, leading to an average of 901.93 fans per profile and 1.55 liked profiles per user. The most liked profile had 64919 fans, and the most active user was a fan of 401 profiles.

We applied an optimization of Murata (2009)'s bipartite modularity with the *mpcd* software package described in Appendix A.1.3, which can handle the bipartite case as a special case of the originally intended, more general k-partite usage. The number of nodes and edges could be significantly reduced by merging users with identical sets of liked profiles, such that two users who both like exactly one profile would be represented as a single user with an edge of double weight to the commonly shared profile (and vice-versa, if two profiles happened to be liked by an identical set of users). Such a "packing" of the graph was previously considered for the tripartite case, but exactly matching connections to two other domains seemed too rare after a short inspection of actual hypergraphs. In the bipartite case however, such identities were found often enough to help reduce the size of the graph. In order to further speed up computation, a next step in compressing the graph was to replace the need for identical connections by a minimum similarity, grouping users whose vectors of liked and unliked profiles showed a cosine similarity of at least 0.5. This finally reduced the number of edges to around 40000 and proved managable for the current algorithm.

5.4.4.3. Results

It has to be noted that the runtime of *mpcd*'s community detection does not scale well, as was already examined theoretically in Section 4.3.4. The community detection on the packed graph took three weeks to finish on a powerful server. However, finding the truly optimal community assignments is computationally intractable to begin with, and in order to evaluate the benefits of searching further shortcuts or introducing new approximations, the center of our attention was the quality of the results.

Figures 5.14 and 5.15 show the results, with the former showing the number of profiles in each cluster, and the latter showing the number of aggregate "likes" in each cluster, each time grouped by party. Several observations can be made:

- The optimal community assignment found consists of six clusters of political profiles.

  As described above, we focussed on the seven major political parties in Germany. However, two of them, CDU and CSU, form the "CDU/CSU" faction at the federal level (regionally, there is no CDU in Bavaria and no CSU outside of it) and are commonly regarded as a single political entity. The reason why there are seven parties, six clusters is that CDU and CSU are grouped together in cluster 1.

- The differentiation between clusters is clear-cut.

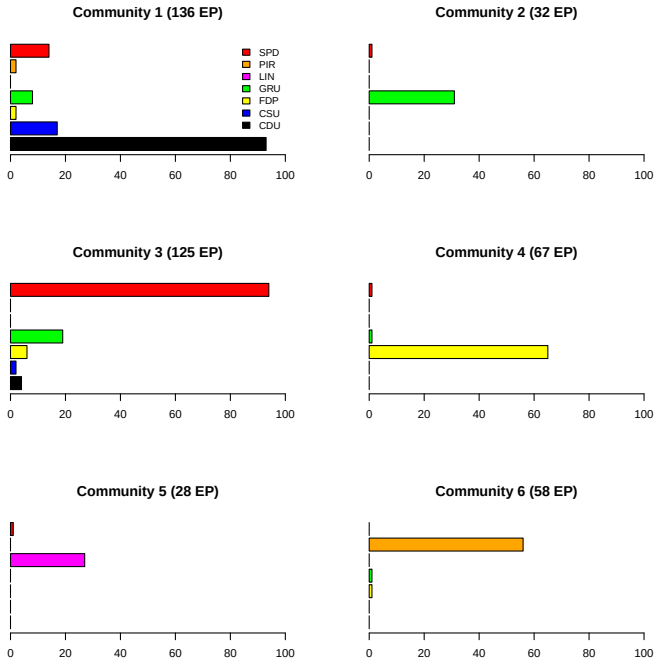


Figure 5.14.: The six resulting communities and the number of profiles in each, by party.

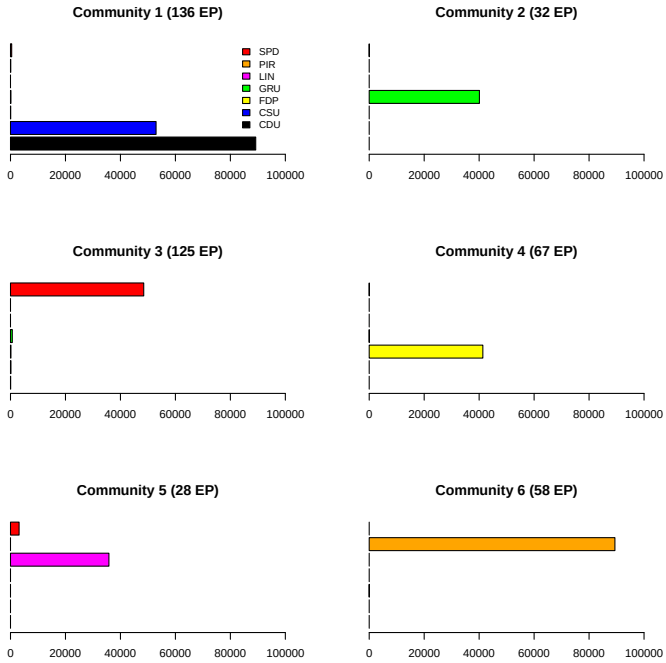


Figure 5.15.: The six resulting communities and the number of times users liked a profile in each, by party.

Each cluster corresponds to one political faction very clearly. Even though there is the occasional "foreign" profile in some clusters, these are profiles which are not liked by many users, as Figure 5.15 shows; those foreign profiles have a diminishingly small aggregate number of users liking them. Conceivably, that small number of connections is also the reason for their misplacement in the first place. The only exception is a single SPD representative in the "Die Linke" ("the Left") cluster 5 who has a high number of fans. She turns out to be Antje Krug, who used to be featured as a candidate in the German political gameshow format "Ich kann Kanzler" ("I can chancellor" [sic]). Judging from online resources of VZ.net (2009) and the SPD (Köcher, 2009), Krug in fact obtained those fans during that show and prior to joining the SPD. Thus discounting this outlier, we can conclude that the political profiles are clustered in such a way that the connections between users and profiles are almost perfectly divided by political faction.

5.4.4.4. Conclusions

The detection of communities in the political profiles could successfully retrieve the underlying structure. The only "mistake" in comparison to the manual category assignments is the merge of CDU and CSU, which, as discussed above, is well in accordance with public perception of the two.

This result is generally good news for a possible application on unlabelled data such as groups. Even though thematic clusters may not be pronounced as clearly in groups as they are in the political landscape, it seems that existing patterns can be identified.

Of course, for a large-scale application of this technique, further optimization of the community detection algorithm would be required in terms of runtime behaviour. The graph can be arbitrarily reduced in size by lowering the threshold for merging similar users. A more sophisticated line of work would require further examination of the actual graph structures, looking for ways to reduce the number of recomputations after each join, possibly leaving certain pairs of possible joins "dirty" (i.e. un-updated) when it can be estimated that their change does not change the favourite in the next step.

5.4.5. Summary

The results of this project can be summed up by three major points:

1. The friendship graphs behaves as expected, showing a giant connected component and a low characteristic path length. Extreme deviations can be traced back to anormal usage patterns.

2. A regional subgraph shows properties remarkably similar to those of the full graph, in terms of path lengths. However, the intuitively expected increase in small world-phenomena is mirrored by a relative increase of short-range connections, which however are overshadowed in the global measure typically applied.

3. Bipartite community detection retrieved an existing category structure rather flawlessly. Given further work on speeding up the process, real-world applications on unlabelled data are conceivable.

For me, this project has confirmed that the abstract measures used in network analysis can in fact be used to better understand a complex system as the graph structures maintained at VZ.net. Revealingly, a lot of the effort was spent on very concrete technical problem-solving to deal with the sheer amount of data – an important point to take home. I hope that, at the same time, some of these rather abstract results will find a way back into daily business – I could imagine, e.g., path length distributions to be useful for certain operational decisions.

6. Conclusions

6.1. Results and Discussion

6.1.1. Summary

This thesis, at its core, is about generalizing concepts defined on binary relations to ternary and theoretically even higher-order relations. In particular, it is concerned with concepts for decomposing the resulting structures into meaningful subparts for easier analysis. The practical background for this generalization is the emergence of datasets like social bookmarking data which are most naturally represented as 3-partite 3-uniform hypergraphs.

The starting point was the identification of spammers in social bookmarking data. The results on hyperincident connected components showed that a) structural information in these datasets is closely tied to the behaviour of its users, and b) a theoretically sound treatment of the underlying hypergraphs may generate insight where a simplification to normal graphs does not. Encouraged by these results, I further pursued the theme of hypergraph decompositions, heading into the more general direction of community detection. In the first of the three chapters dedicated to this topic, this task was formally defined for k, k-hypergraphs, along with a set of challenges faced when dealing with these structures, and synthetic examples to benchmark progress on the way to solving them. In the next chapter, several methods were developed to solve this task, from a simple (non-partite) baseline to a final (balanced multi-partite) solution incorporating both theoretical considerations and answers to practical issues encountered. Finally, a tool for visualizing and exploring the obtained results was introduced and the methods were compared in different situations, highlighting their practical differences when applied on real data.

To conclude, this thesis provides, on the practical side, insights into the structure of social bookmarking datasets as well as tools for their further exploration. On the theoretical side, existing gaps are filled by generalizing concepts well-established on graphs, connected components and modularity, to k, k-hypergraphs.

6.1.2. Discussion

6.1.2.1. On Generalizations

The main theoretical contribution of this thesis is the generalization of modularity to k-partite, k-uniform hypergraphs. Several interesting things happened during the creation of this generalization, or rather, these generalizations.

Simple generalizations... Back in Section 2.2, during the generalization of connected components, life was easy. The original concept was rephrased in a way that made explicit the formerly hidden parameter tying it to the 2-dimensional case, and then that parameter could be scaled up to accomodate the higher-dimensional cases.

. . . don't always work It became evident in Section 4.3 that a similar approach was not good enough for the generalization of modularity. The original measure to be generalized neglects, for each community, all outgoing edges except the ones connecting to the most strongly connected other community. A simple generalization would neglect all edges except those connecting to exactly the most strongly connected *pair* of communities from the two other domains. This led to a restrictively high number of edges that were not taken into account by the naive generalization. This issue could be solved by replacing the original argmax-based, binary selection heuristic by a linear correspondence function. Still, it is worth noting that the process of generalization exposed a structural weakness in the original method that only escalated when taken to higher dimensions.

Losing on the simpler cases Another issue, addressed in Section 4.4, was the effect that the higher-order method, while performing well on the test cases involving higher-order correlations, failed to handle the simpler cases correctly. The solution here was to combine the simpler and the advanced method into a hybrid one. To bring up again the analogy of my colleague Wendelin Boehmer, a quadratic function can fit curves a linear function cannot, but one still keeps the linear term around in case one needs to fit a line.

Undesirable optima The last issue I want to discuss is the one resolved in Section 4.5, where it turned out that the multi-partite modularity function would find unexpected and undesirable optima by joining all elements from one domain into a single giant community. The solution is to encode the desired property of balanced community sizes into the objective function. Still: This property came "for free" in the bipartite case, and only in the higher-dimensional case the possibility of joining everything in one domain and then only optimizing the remaining two ones became an (undesired) possibility.

I guess the bottom line is that although sometimes, an elegant formal generalization on paper may do the trick, all sorts of unexpected effects may appear when an algorithm of sufficient complexity is moved out of its original domain (but then of course, that's the whole fun of it).

6.1.2.2. On Synthetic Datasets

One might argue that the results on the synthetic examples basically show what they were designed to show: Higher-dimensional relations being reconstructable only by algorithms taking into account these relations. The respective advantages, one might continue, might have been shown purely analytically as well. However, it is not quite as easy, and in my opinion the observed examples are valuable for at least three reasons.

1. Concrete examples help to solidify the image of the problem at hand. For example, I was certain that multiple bipartite projections lose information contained in a hypergraph long before the CONTRADICTIVE example was made up. However, it was

hard to actually put a finger on when exactly this would happen. As we see now, it is actually a pretty specific case in which tripartite structure really is important. More generally, the examples helped putting the problem into a format which could be viewed, discussed, and played with.

2. Numerous ideas have been tried out during the creation of this thesis that, in theory, treated the tripartite structure in a similarly native way, but failed to work on the examples. So the synthetic datasets were not made up after the fact to show that everything works, but have been around all the way, shaping the development of the proposed algorithms. In that sense, they might be understood as machine learning equivalents of unit tests in software engineering.

3. Certain phenomena could be observed that probably would have been very hard to anticipate by a purely analytic approach. A simple example is the behaviour of the different algorithms under the influence of noise. A more significant example is the poor performance of the multi-partite modularity measures from Section 4.3 on the SIMPLE dataset. Never would it have occured to me that a more advanced representation of the data might actually miss relations that are obvious to a simpler one.

Of course, not every possible challenge could be cast as a synthetic dataset. The creation of pathological giant communities that led to the definition of the balanced modularity only became apparent once real datasets were examined. Still, the simplified and artificial example with different "arms" was a helpful step towards identifying the underlying problems, underlining the importance of the synthetic examples as condensed problem descriptions and test cases.

To conclude, I believe the synthetic datasets have proven themselves to be valuable tools in devising the proposed algorithms. By making them available per download, I hope they will be helpful for further developments as well.

6.1.2.3. On Algorithms

The work on multi-partite community detection was theoretically motivated by the challenges put forward in Section 3.1.2: distinct community structure, correspondence instead of equality, and hyper-incidence. Now that these challenges have been resolved by the proposed algorithms: Was it worth it? Are the results sufficiently better?

The most significant difference is observed when switching from the non-partite baseline to any partition-aware method, even the coupled bipartite one. This is not much of a surprise, because even this bipartite method takes into account the first two challenges already. By removing the implicit restriction to group documents, users, and tags together, new solutions become possible that apparently better fit the actual structure contained in the datasets (different numbers of communities among domains, connections between the main clusters), as shown e.g. in Section 5.3.1.

The difference between the bipartite and the multi-partite modularity measures is more subtle. The compression figures examined in Section 5.3.2 show that the multi-partite methods bring the same improvement relative to the bipartite method as the bipartite one brings

in relation to the non-partite one, yet the difference is sometimes hard to spot. In the same section, we see a case where the (balanced) multi-partite method finds interesting inter-cluster connections that the bipartite misses, which may hint at an increased sensitivity of the multi-partite methods. Yet, I could not find an example in which the difference between the two is as striking as the one to the non-partite method. This, however, may also include an insight about the underlying data, to be discussed in the following.

6.1.2.4. On Social Bookmarking Data

The practical motivation for the work on community detection in hypergraphs was to uncover the hidden knowledge contained in social bookmarking data. What has generally been learned about these datasets?

Cognition does matter. Social bookmarking collects traces of rather high-level cognitive behaviour, and it is nice to see that this actually shows in the structure of the data. This was, for me, the main take-away from the spam detection challenge described in Chapter 2 – automated spam scripts not being fully capable of replicating the structure created by thinking humans. This mix between highly individual behaviour and consistent large-scale phenomena is for me one of the most fascinating features of this data. I hope the visualizations like the tag/document clouds of whole datasets or the community structure of particular subgraphs succeed in transporting the sense of beauty that emerges when dealing with this data.

Users are more than keyword providers. A plausible criticism of the methods described in this thesis might be: Why can't we simply aggregate over all users and use the sum of their tags? If we look, e.g., at the community structures in the previous chapter, we can see that the more interesting insights are typically generated by explicitly taking into account *who* tags *what* – given sufficiently sophisticated methods are in place to use this information.

There is little controversy. As noted earlier, the difference between the bipartite and the truly multi-partite methods did not turn out as visually significant as one might have (or at least I have) expected. However, a class of synthetic datasets could be produced that can only be resolved by these multi-partite measures. The conclusion seems to be that these particular cases do not play a huge role in the real data. Interpreting this example, the CONTRADICTIVE graph family, it turns out that it simulates a scenario of controversy: One community of users insists to tag one community of documents one way, another one insists to tag them differently; and there is another community of documents where the situation is reversed. If tags were ratings, such situations should be expected to occur. In the actual data however, such behaviour could not be found even when I explicitly looked for it (by examining the neighborhood of tags like "war" or "politics" more closely). The reason seems to be that even if two users disagree on politics, they are going to tag a political article with "politics" rather than with "good", "bad" or "propaganda". This is actually consistent with the fact that tags are primarily used for personal information management, and only secondarily, if at all, with the intention of signalling their content to other users.

6.2. Outlook

Finally, let me describe some possible extensions of the work presented here, as well as possible future developments it could provide insights for.

6.2.1. Hyperincident Connected Components

Numerous questions remain to be explored in the area of hyperincident connected components: Can we create a generative model that creates the component signatures of legitimate and spamming activity? Do the findings generalize to spam in other datasets?

On a more specific level, what are the reasons for the slightly different signature of the Delicious corpus? In comparison to the other datasets, it contains slightly larger next-largest components, and also the frequency of component sizes does not peak at 1, but at 6 edges, while no component smaller than 3 edges exists. One possible explanation for the bigger next-largest components might be the presence of moderate amounts of spam. A possible explanation for the shifted peak might be the presence of tag suggestions in the Delicious interface: Maybe users are less likely to remain completely unconnected if they are encouraged to repeat tags other users have assigned earlier. In any way, it has to be noted that this corpus, as opposed to the others, is only a slice from the middle of the usage period – any further investigation of these features would have to be performed on a more complete dataset.

Furthermore, what exactly is the process leading to the formation of the huge giant component in the search-log data, and what can it tell us about the difference between finding a document by a keyword and labelling it with a tag? Does this phenomenon generalize?

Finally, I have gone to some length to introduce m-incident and not simply 2-incident connected components – so another path of further investigations would be to examine hypergraphs with range >3, allowing the decomposition into 3- and higher-hyperincident connected components.

6.2.2. Community Detection

The current optimization of multi-partite modularity might be further improved. Although analytic gain updates as proposed by Clauset et al. (2004) for the non-partite case (not requiring further loops over edges after initialization) seem hard to obtain because of the non-linearity of potentially changing corresponding communities, further speed improvements should be possible by, e.g., allowing for incomplete updates where justifiable. Alternatively, the generalization of other optimization approaches such as the fast "Louvain" method (Blondel et al., 2008) appears promising. Many other community detection approaches are suitable for generalization as well.

Another big question is how to compare different community assignments for real-world data in a more quantitative way than what has been presented here. Either a dataset with ground truth communities along all three domains can be obtained, or justified unsupervised quality measures are developed. Such justification however should best be obtained relative to actual users, and this also brings up the probably most valid evaluation: Integrating tools

based on community detection into actual social bookmarking services and measuring the amount of human effort saved.

6.2.3. k-Partite, k-Uniform Hypergraphs

(3,3)-hypergraphs have been mostly discussed in relation to social bookmarking data, but in fact they are significant to a much wider range of datasets: Wherever terms are connected to entities not in a static, objective way as represented by the classic term/document matrices, but associated by users in noisy, subjective ways, the information about who associated the term becomes crucial. Therefore, the three-dimensional tensors gained by extending the term/document matrix by a user dimension, and the hypergraphs created by these tensors, appear as the basic model of social media: comments made by a particular user to a blog post/product/etc, tweets by a user involving a URL, changes made by a user to a wikipedia entry – all these cases involve tripartite structures, and I hope to have provided motivation for a dedicated treatment of these peculiar structures. Future research might even identify applications beyond the domain of social media.

6.2.4. Social Data

The technologies introduced in this thesis are prototypical, yet concrete applications could be built on top of them. Decomposing social bookmarking data in the ways described here could be shown to sometimes provide qualitatively new insights – e.g. by identifying communities of users that could be interesting for a user, and being able to label them based on the documents and tags they are associated with. The visualizations created by `mpce` certainly have an "expert mode" feel to them – still, they showcase how the rather abstract output of multi-partite community detection algorithms might be used to create new user interfaces, allowing a richer structuring of document collections based on socially annotated data.

Hendler et al. (2008) have coined the term "web science" for the interdisciplinary study of the internet "as an entity in its own right". Beyond the specific example of social bookmarking, the trend for users to generate ever more data and for services to mine this data for insights is unbroken – creating not only more, but also qualitatively new types of data. Time and location, for example, are two additional dimensions that play an important role in many new datasets. These dimensions are admittedly different than, say, tags, because there is already a strong relation between datapoints in terms of spatial or temporal proximity, so the need to establish such relations does not exist in the same sense as for the initially unrelated tags. Still, interpreting these coordinates as nodes in a hypergraph might open up new exploratory methods, and I would like to hope that some of the insights generated here can be helpful for their construction.

A. Appendix

A.1. Software Packages

Here, the software packages that have been created as part of this thesis are described. I will provide a short introduction for each and then append excerpts of the corresponding readme files to give an idea of the actual usage. All software packages can be found under `http://www.ni.tu-berlin.de/menue/software/`.

A.1.1. hcc (Hyperincident Connected Components)

`hcc` provides a database-backed implementation of the decomposition algorithm proposed in Section 2.2. While it requires the data to be inserted in to a database in a specific format, its advantage over the memory-based approach is that is suitable for decomposing networks with tens of millions of entries, furthermore persisting the results. It also contains auxialiary functions for charting the obtained results.

Description

hcc computes connected components and hyperincident connected components in social bookmarking datasets which are stored in a database.

Usage

```
python hcc.py [cc|hcc|hccplot|hccstats] db
```

`cc` computes the connected components.

`hcc` computes the 2-incident connected components.

`hccstats` outputs basic statistics about the found components.

`hccplot` creates .plt-files of the component size distributions to be processed by GnuPlot.

`db` is the name of a database reachable through `dbconnection()` and containing at least the tables and fields defined in `template.sql` . In particular, edges must be stored in the `entries` table, where `document_id`, `user_id` and `tag_id`point to the three elements connected by the edges. The corresponding tables `documents`,`users`, and `tags` can be used to store additional metadata.

A.1.2. mpcb (Multi-Partite Community Benchmarks)

The software package `mpcb` provides facilities for benchmarking future community detection algorithms on k, k-hypergraphs. It contains code to generate random hypergraphs according to the models introduced in Section 3.3, as well as a database-backed evaluation framework to integrate novel detection algorithms as well as additional hypergraph models. A database is available for download that contains performance evaluations of the non-partite, coupled bipartite and hybrid modularity, together with the hypergraphs on which these evaluations have been run. Like this, future algorithms can be evaluated on the same data, which – along with their included performance data – allows for a direct comparison with my approaches.

Description

`mpcb` runs benchmarking tests on multi-partite clustering algorithms.

Setup

`mpcb` creates synthetic hypergraphs and stores them for later reuse, along with any already obtained results. If you have already run experiments on 100 graphs with one method, you can add a second one and it will be applied to the same 100 graphs, making it comparable to the first method without need to recompute that first method's performance. In order to store this data, a database is required. Please create a database `mpcb` (the name can be changed along with database credentials in configuration.py) and execute the code in `mpcb.sql` or `mpcb_data.sql`, to get an empty database or, respectively, one containing reference data by my approaches to compare against.

Usage

`mpcb.py` provides a list of method groups as defined in configuration.py (by default, only nonvsbi) Example:

```
> python mpcb.py
```

`mpcb.py method_group graphtype nodes [start_id] [num_graphs]` runs the evaluation using the clusterers from the provided method group, the graph type, creating graphs with the given number of nodes. A start id and the highest id to be evaluated can be given. Graph types:

0 all

1 Simple

2 Overlapping

3 Contradictive

Example:

```
> python mpcb.py nonvsbi 1 10 0 10
```

`list.py` without parameters shows a list of available method groups.

`list.py method_group` shows a list of graph families for which results are available for the given method group.

Example:

```
> python list.py nonvsbi
```

might create an output like

```
 10 (   0) contradiction(5.0, 2.0)
 10 (   1) contradiction(5.0, 3.0)
 10 (   2) contradiction(5.0, 4.0)
 10 (   3) contradiction(5.0, 5.0)
 10 (   4) overlap(5.0, 1.0)
 10 (   5) overlap(5.0, 2.0)
 10 (   6) overlap(5.0, 3.0)
 10 (   7) simple(5.0, 0.5, 2.0)
 10 (   8) simple(5.0, 1.0, 2.0)
 10 (   9) simple(5.0, 2.0, 2.0)
```

This means, e.g., that 10 graphs have been evaluated on all methods in nonvsbi of the Contradictive family with 5 nodes per community and 2 edges per node. This graph family has the family id 0 (value in paratheses). Use these family ids to create a gridded result chart in the next step.

`Routput.py method_group familyid_familyid_...[#familyid_...[#...]]`

`rowtitle[_rowtitle[_rowtitle...]]` creates a set of data files that can be used with the provided R script `plotresults.r` to produce evaluation charts. Accepts a list of family ids which are separated by _, or by # in order to create a new row. Row titles are separated by _.

Example:

```
> python Routput.py nonvsbi 7_8_9#4_5_6#0_1_2
  Simple_Overlapping_Contradictive
```

`plotresults.r method_group [method_group]` plots the data created by Routput.py

Example:

```
> Rscript plotresults.r nonvsbi
```

creates a file results_nonvsbi_figure.pdf based on the data created by Routput.py.

`print.py method_group family_id` prints the numeric results of the methods in the provided method group for the given graph family.

Example:

```
> python print.py nonvsbi 0
```

shows the results for the graph family with id 0.

`scatter.py method_group family_id` creates a set of data files that can be used with the provided R script plotscatter.r to produce scatter plots showing the numeric results of the methods in method_group on all charts given by family_id.

Example:

```
> python scatter.py nonvsbi 7 1.0
```

if used on the dataset shown above would scatter the results of the non- and bipartite methods on graphs of the simple(5.0, 0.5, 2.0) family with p=1.0 (i.e. noiseless).

`plotscatter.r filename` creates a PDF from scatter data.

Example:

```
> Rscript plotscatter.r scatter_nonvsbi_7_1.000
```

creates a scatter plot from the file created by the previous call of scatter.py.

Adding own clusterers

In order to analyse your own clusterer, extend the `Clusterer` class in `clusterers.py`. `get_clusters` must accept in `edges` a dictionary of the format

```
{(d,u,t): w, ...}
```

and return a list of community assignment dictionaries, one per dimension:

```
[{d1: community_id, d2: ...},
 {u1: ..., ...},
 {t1: ..., ...}]
```

Overwrite `__str__` to control the name under which your clusterer's results will be stored. Integrate your clusterer by creating a method group in `configuration.py` , e.g. by adding

```
"my_method_group": [clusterers.FakeClusterer("Hybrid"),
                    clusterers.MyNewClusterer()]
```

to compare your algorithm to the performance data of the hybrid clusterer.

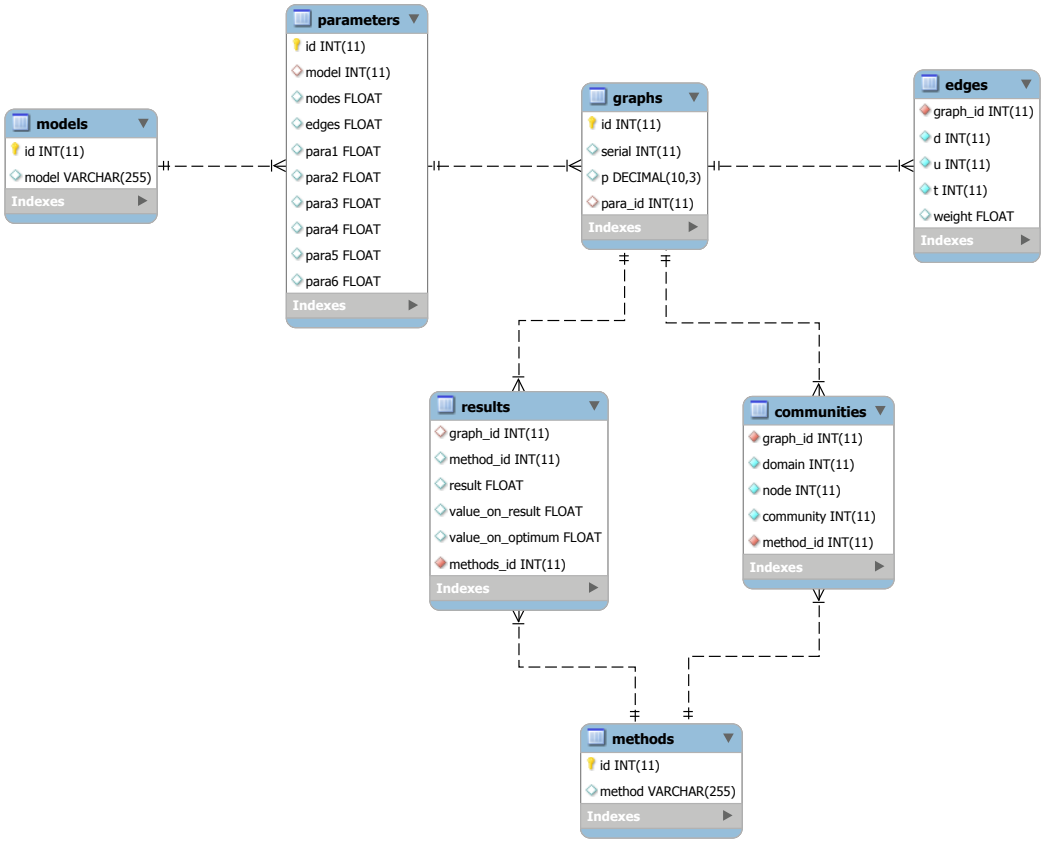


Figure A.1.: Entity Relationship Diagram for the database holding graphs and results created via `mpcb`.

A.1.3. mpcd (Multi-Partite Community Detection)

The software package `mpcd` provides a Java-based command-line implementation of the coupled bipartite modularity optimization introduced in Section 4.2.

Description

`mpcd` accepts a hypergraph in text format and performs a community detection based on coupled bipartite modularity optimization.

Input Format

`mpcd` expects as input a list of edges, given by three space-separated element ids and a weight per line, e.g.

```
2450 1656 1180 1.000000
...
```

Running `mpce`

```
mpcd [inputfile] -o [output file prefix] -om
```

`om` outputs the individual joins at each step, instead of just storing the final community assignment of optimal modularity

Output format

`[output file prefix].[0|1|2]` contain a list of

`node id: community id`

pairs.

A.1.4. mpce (Multi-Partite Community Exploration)

The software package `mpce` as described in Section 5.2 provides a Java-based frontend for visualizing and exploring community domains as created for example, but not necessarily, by the aforementioned `mpcd`.

Description

`mpce` accepts a hierarchical clustering in three dimensions and allows an interactively exploring it. The GUI starts showing the top-most state of the clustering, all elements of one dimension merged into a single community. By pressing space or clicking on individual communities, the next or the selected community is split into the two subcommunities it consists of. A PDF can be exported of the current interface state by pressing 'p'. By hovering over a community node and pressing 'a', annotations can be turned on or off.

Input format

`mpce` expects, as parameters, a directory and a merge file name. In the directory, a file `edges.txt` is expected which must contain the edges in the format

```
1   1     1  1.0
```

to describe a hyperedge between elements 1, 1, 1 of weight 1.0.

The `*.merges` files contain the main clustering information like

```
2       1937    1066    0.026370        0
```

which mean that in dimension 2, community 1937 is merged into 1066. The following two colums indicate step-wise clustering quality and clustering step.

Additionally, the directory can contain `names.0`, `names.1`, and `names.2`, which, for the corresponding dimension, contains labels like

```
121771: http://google.com
```

Running `mpce`

Either run `mcpe` via `mpce.bat [directory]`, or import `mpce` as an Eclipse project and run `MPCEViewer`, in both cases using a 32-bit JRE.

Example In the mpce directory, try

```
mpce example fc
mpce example tribi
```

to visualize the clustering of a sample graph using the non- or coupled bipartite approach.

A.2. Additional Figures

A.2.1. Full Set of Tag/Document Clouds

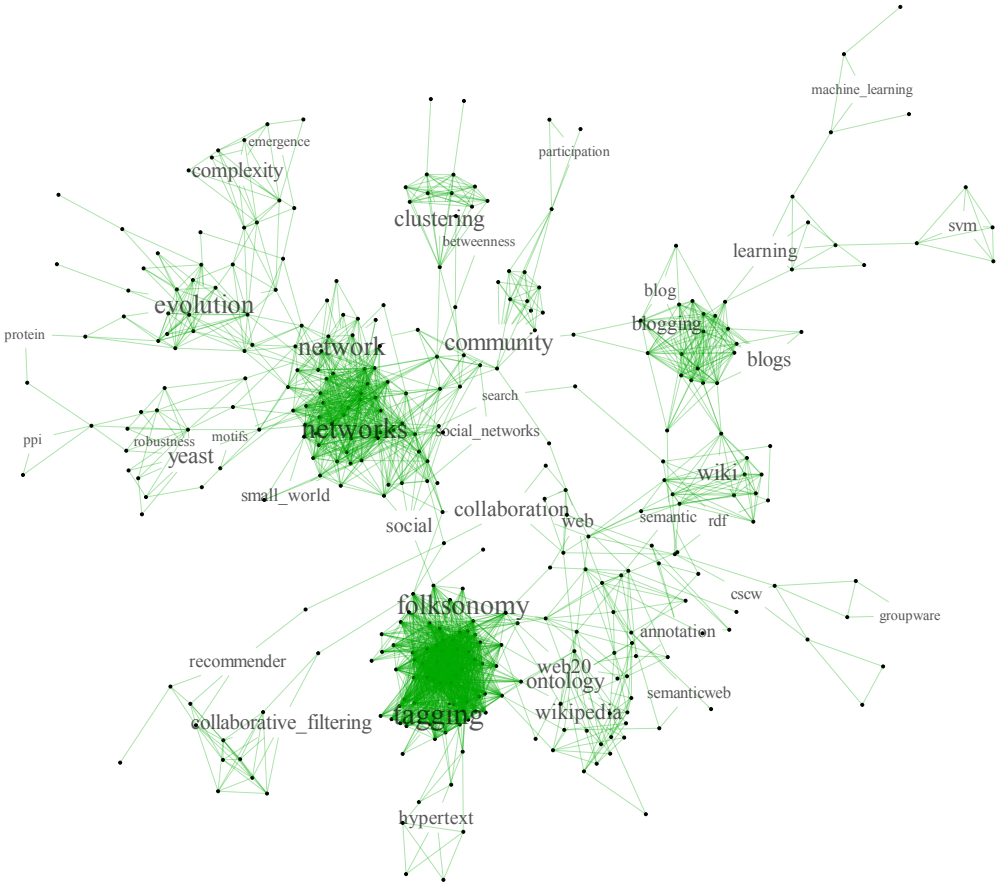


Figure A.2.: Tag/Document Cloud of the CiteULike dataset

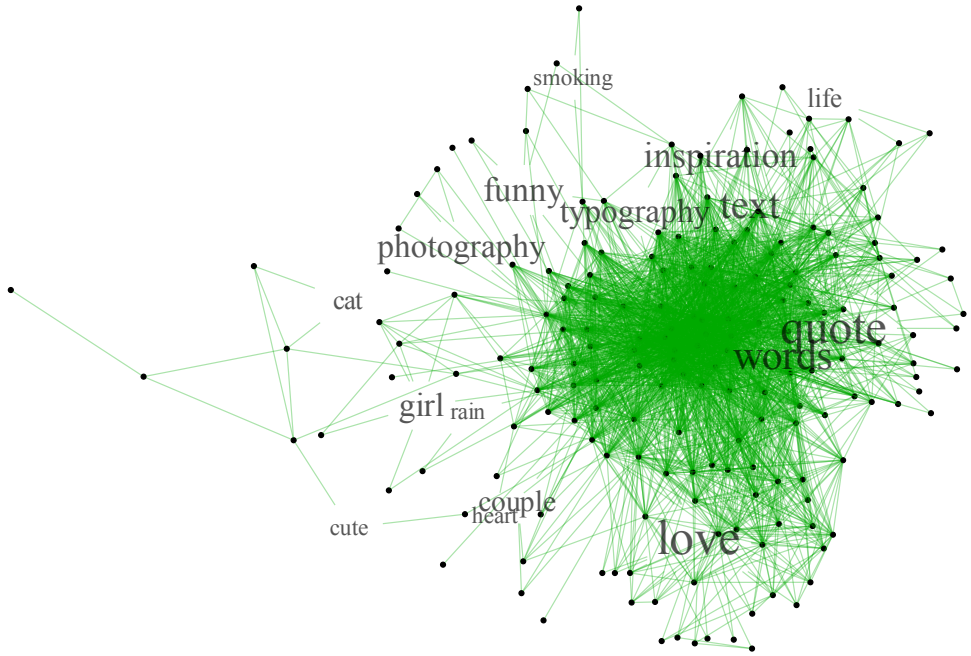


Figure A.3.: Tag/Document Cloud of the Visualizeus dataset

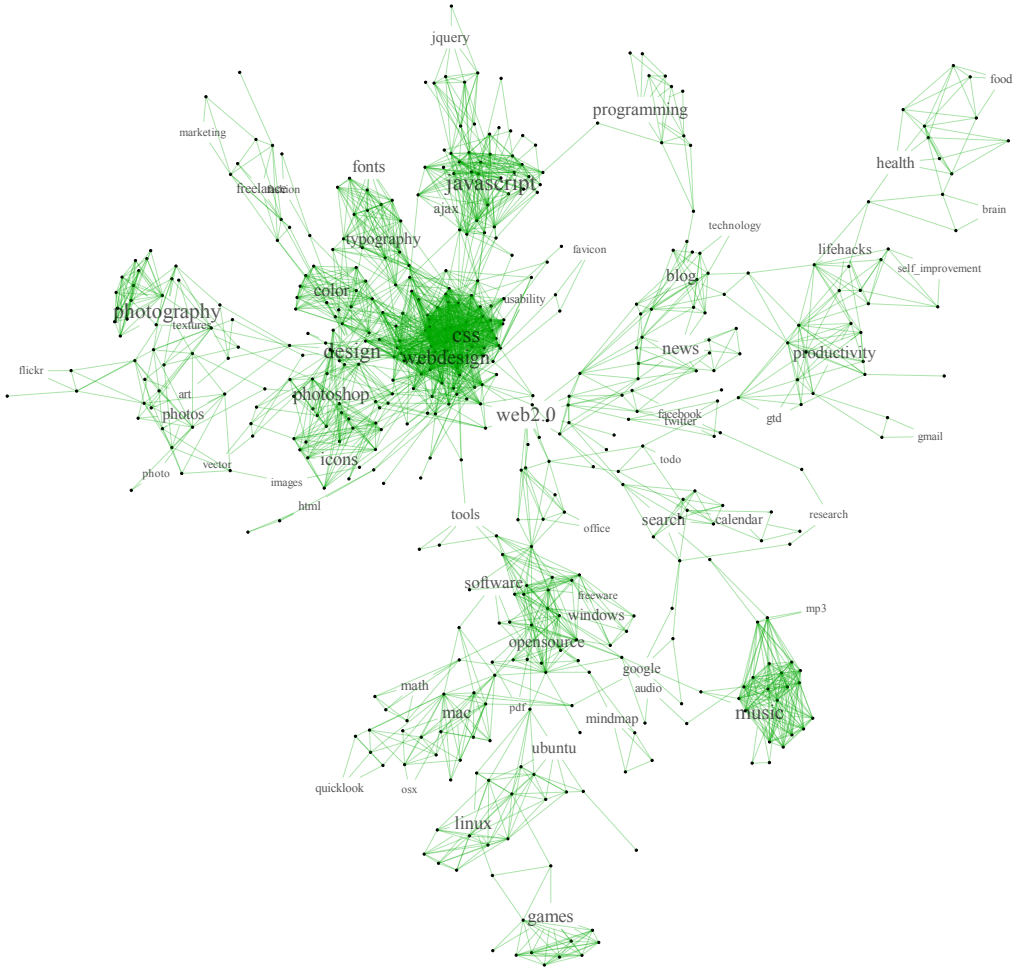


Figure A.4.: Tag/Document Cloud of the Delicious dataset

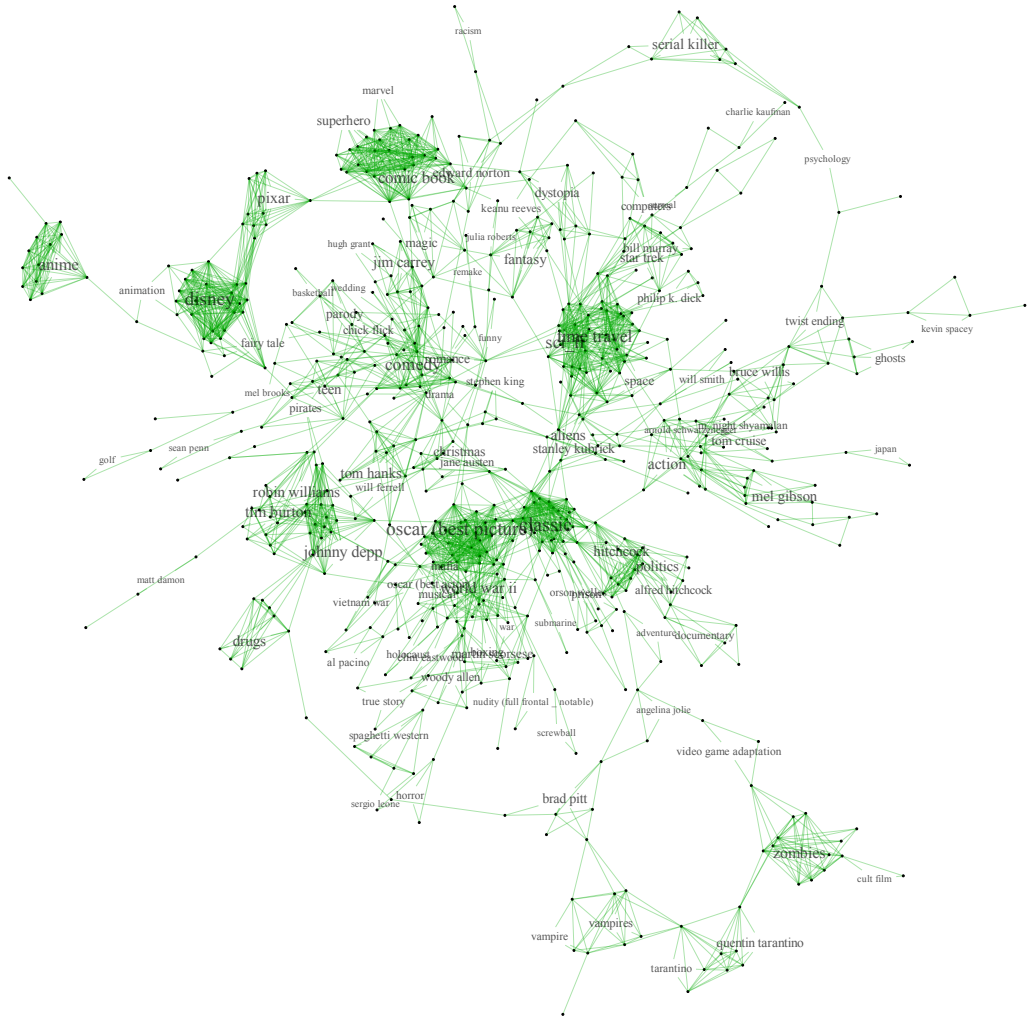


Figure A.5.: Tag/Document Cloud of the MovieLens dataset

A.2.2. Additional Charts

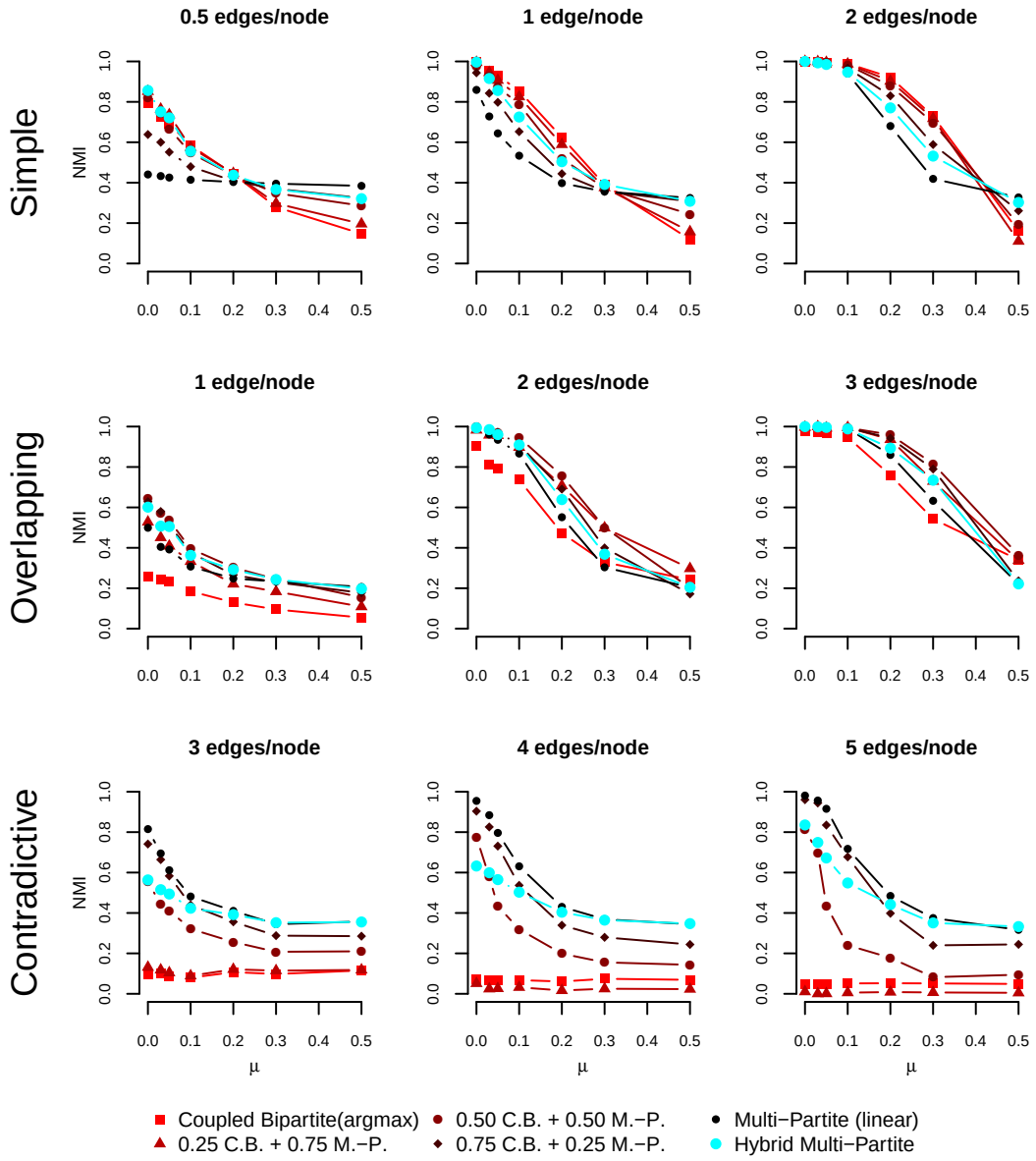
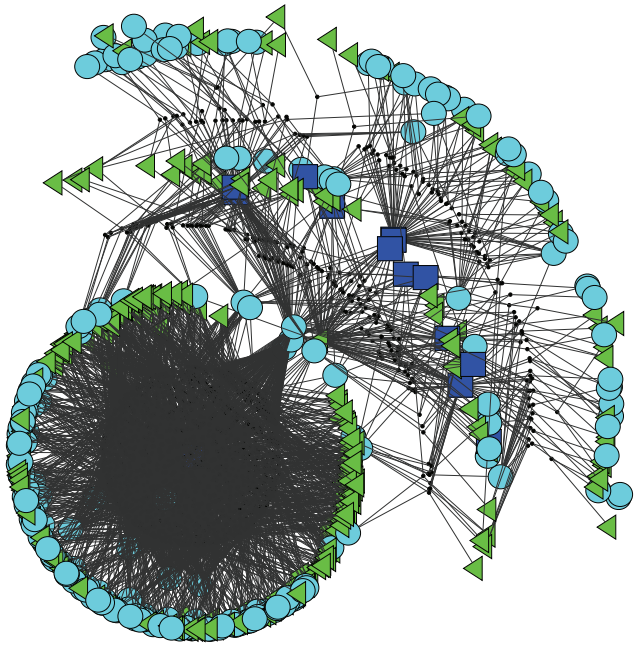
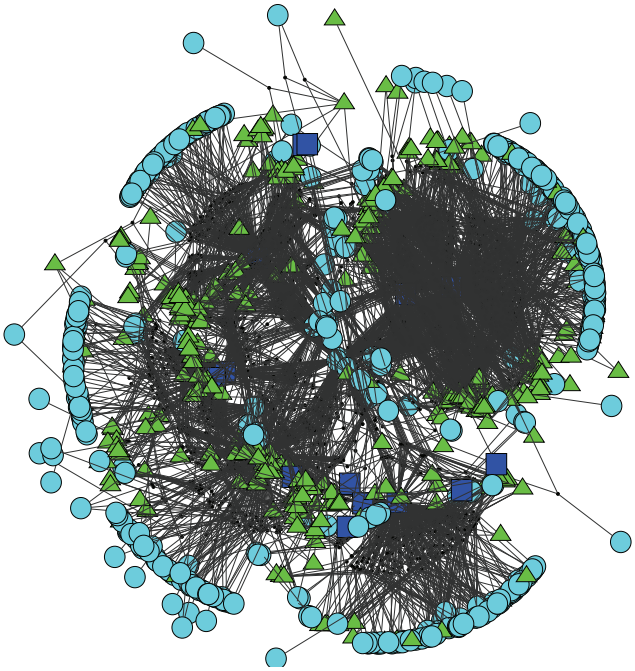


Figure A.6.: This shows basically the same information as Figure 4.11 on page 97, except an argmax-based correspondence function is used for the bipartite modularity. See Section 4.1.2 for a detailed explanation of this chart.

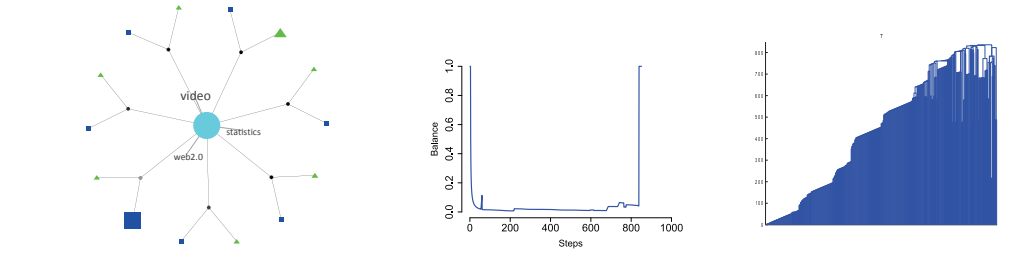


(a) A sample hypergraph H_{low}

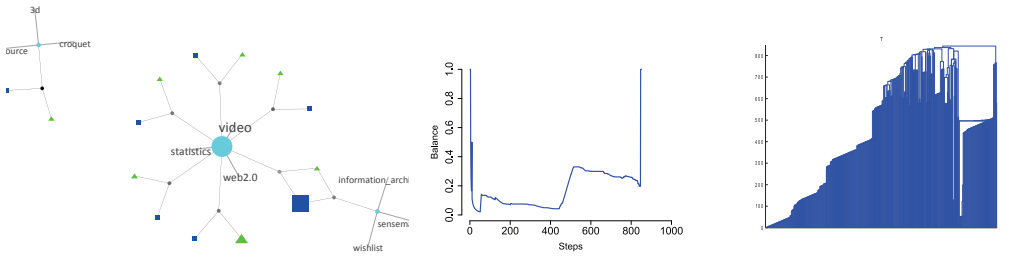


(b) A sample hypergraph H_{high}

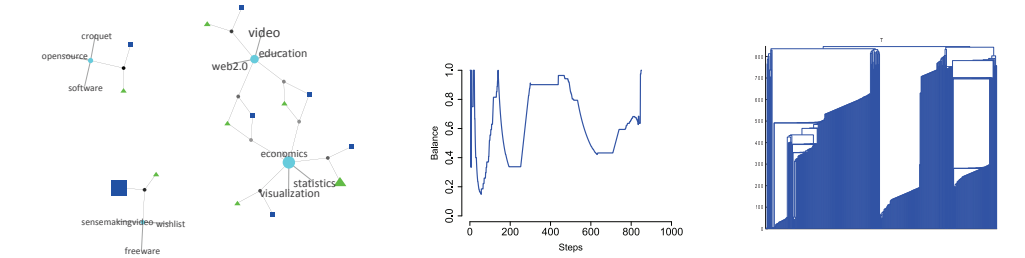
Figure A.7.: The two hypergraphs examined in the following two charts



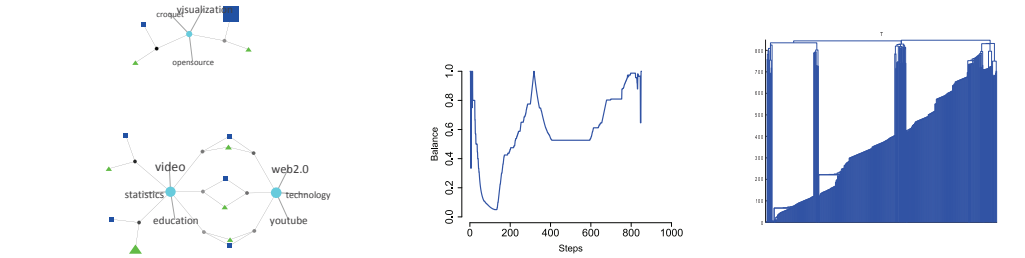
(a) Visualization for $\alpha = 1$ (b) Balance chart for $\alpha = 1$ (c) Tag dendrogram for $\alpha = 1$



(d) Visualization for $\alpha = \frac{1}{2}$ (e) Balance chart for $\alpha = \frac{1}{2}$ (f) Tag dendrogram for $\alpha = \frac{1}{2}$

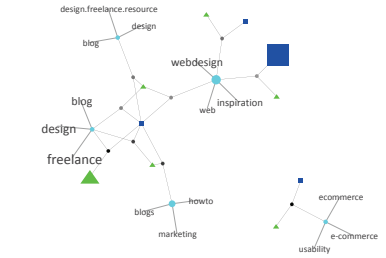


(g) Visualization for $\alpha = 0$ (h) Balance chart for $\alpha = 0$ (i) Tag dendrogram for $\alpha = 0$

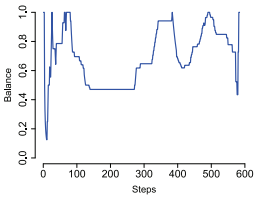


(j) Visualization – balanced (k) Balance chart – balanced (l) Tag dendrogram – balanced

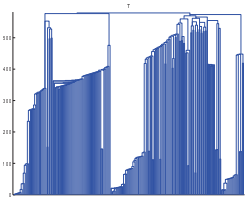
Figure A.8.: H_{low} – results for the dampened modularity with various values of α and the balanced modularity



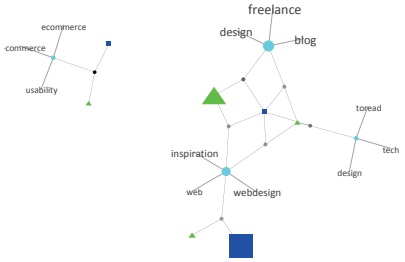
(a) Visualization for $\alpha = 1$



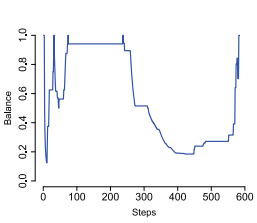
(b) Balance chart for $\alpha = 1$



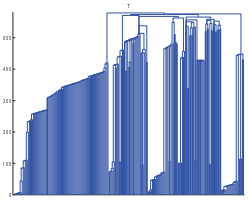
(c) Tag dendrogram for $\alpha = 1$



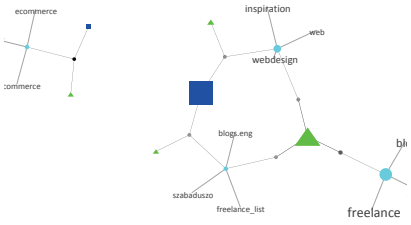
(d) Visualization for $\alpha = \frac{1}{2}$



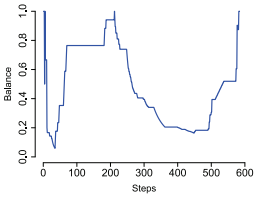
(e) Balance chart for $\alpha = \frac{1}{2}$



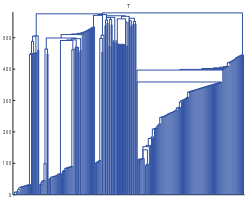
(f) Tag dendrogram for $\alpha = \frac{1}{2}$



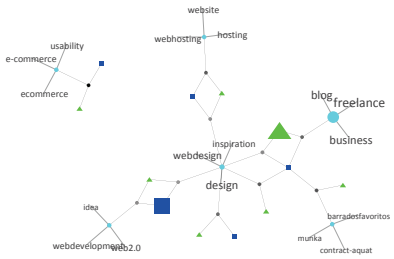
(g) Visualization for $\alpha = 0$



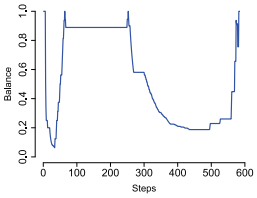
(h) Balance chart for $\alpha = 0$



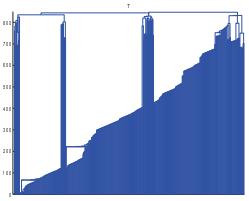
(i) Tag dendrogram for $\alpha = 0$



(j) Visualization – balanced



(k) Balance chart – balanced



(l) Tag dendrogram – balanced

Figure A.9.: H_{high} – results for the dampened modularity with various values of α and the balanced modularity

Bibliography

Acar, E., Camtepe, S. A., Krishnamoorthy, M. S., and Yener, B. (2005). *Intelligence and Security Informatics*, chapter Modeling and Multiway Analysis of Chatroom Tensors, pages 256–268. Springer Berlin / Heidelberg.

Ahern, S., Naaman, M., Nair, R., and Yang, J. H.-I. (2007). World Explorer: visualizing aggregate data from unstructured text in geo-referenced collections. In *JCDL '07: Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 1–10, New York, NY, USA. ACM.

Al-Khalifa, H. S. and Davis, H. C. (2006). Folksonomies versus automatic keyword extraction: An empirical study. *IADIS International Journal on Computer Science and Information Systems (IJCSIS)*, Vol. 1:132–143.

Alvarez-Hamelin, J. I., Dall'Asta, L., Barrat, A., and Vespignani, A. (2005). k-core decomposition: a tool for the analysis of large scale internet graphs. Technical Report cs.NI/0511007.

Arenas, A., Duch, J., Fernandez, A., and Gomez, S. (2007). Size reduction of complex networks preserving modularity. *New Journal of Physics*, 9(6):176.

Bao, S., Xue, G., Wu, X., Yu, Y., Fei, B., and Su, Z. (2007). Optimizing web search using social annotations. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 501–510, New York, NY, USA. ACM.

Barber, M. J. (2007). Modularity and community detection in bipartite networks. *Physical Review E*, 76(6):066102.

Begelman, G., Keller, P., and Smadja, F. (2006). Automated tag clustering: Improving search and exploration in the tag space. In *Collaborative Web Tagging Workshop at WWW2006, Edinburgh, Scotland*.

Bekkerman, R., El-Yaniv, R., and McCallum, A. (2005). Multi-way distributional clustering via pairwise interactions. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 41–48, New York, NY, USA. ACM.

Berge, C. (1970). *Graphes et hypergraphes*, volume 37 of *Monographies Universitaires de Mathématiques*. Dunod, Paris.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008.

Boesch, F. (1986). Synthesis of reliable networks, a survey. *IEEE Trans. Reliab*, 35:240–246.

Bradde, S. and Bianconi, G. (2009). The percolation transition in correlated hypergraphs. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(07):P07028.

Bruns, A. (2008). *Blogs, Wikipedia, Second Life, and Beyond - From Production to Produsage*, volume Vol. 45 of *Digital Formations*. Peter Lang Publishing Group.

Carrasco, J., Fain, D., Lang, K., and Zhukov, L. (2003). Clustering of bipartite advertiser-keyword graph. In *Proc. International Conference on Data Mining (ICDM'03)*, Melbourne, Florida.

Cattuto, C., Benz, D., Hotho, A., and Stumme, G. (2008). Semantic grounding of tag relatedness in social bookmarking systems. In Sheth, A. P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T. W., and Thirunarayan, K., editors, *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 615–631. Springer.

Cattuto, C., Loreto, V., and Pietronero, L. (2007a). Semiotic dynamics and collaborative tagging. *PNAS*, 104(5):1461–1464.

Cattuto, C., Schmitz, C., Baldassarri, A., Servedio, V. D. P., Loreto, V., Hotho, A., Grahl, M., and Stumme, G. (2007b). Network properties of folksonomies. *AI Communications Journal, Special Issue on "Network Analysis in Natural Sciences and Engineering"*, 20(4):245–262.

Chakrabarti, D., Papadimitriou, S., Modha, D. S., and Faloutsos, C. (2004). Fully automatic cross-associations. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 79–88, New York, NY, USA. ACM.

Chvátal, V. (2003). Claude Berge: 5.6.1926-30.6.2002. *Graphs and Combinatorics*, 19(1):1–6.

Cichocki, A., Zdunek, R., Phan, A. H., and Amari, S.-i. (2009). *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Wiley Publishing.

Clauset, A., Newman, M. E. J., and Moore, C. (2004). Finding community structure in very large networks. *Physical Review E*, 70(6):066111.

Danon, L., Guilera, A. D., Duch, J., and Arenas, A. (2005). Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(9).

delicious.com (2010). *Detail page for Hubble Space Telescope Advent Calendar 2008*. `http://www.delicious.com/url/f8092c0a95f029d2b9bd04aff9d3f169`, retrieved December 2010.

Dhillon, I. S., Mallela, S., and Modha, D. S. (2003). Information-theoretic co-clustering. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 89–98, New York, NY, USA. ACM.

Dourish, P. and Chalmers, M. (1994). Running out of space: Models of information navigation. Glasgow, UK. Short paper presented at HCI'94.

Erdos, P. and Renyi, A. (1960). On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61.

Estrada, E. and Rodriguez-Velazquez, J. A. (2005). Complex networks as hypergraphs. *Systems Research*, page 16. arXiv:physics/0505137v1 [physics.soc-ph].

Evans, T. S. and Lambiotte, R. (2009). Line graphs, link partitions, and overlapping communities. *Physical Review E*, 80(1):016105.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27(8):861–874.

Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486:75–174.

Fortunato, S. and Barthélemy (2007). Resolution limit in community detection. *Proc. Natl. Acad. Sci.*, 104:36–41.

Franz, T., Schultz, A., Sizov, S., and Staab, S. (2009). TripleRank: Ranking semantic web data by tensor decomposition. In *8th International Semantic Web Conference (ISWC2009)*.

Fu, W.-T. (2008). The microstructures of social tagging: a rational model. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, CSCW '08, pages 229–238, New York, NY, USA. ACM.

Gansner, E. R. and North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software — Practice and Experience*, 30(11):1203–1233.

Ghosh, R. and Lerman, K. (2009). Structure of heterogeneous networks. In *Proc. of the 1st IEEE Social Computing Conference*, Vancouver, Canada.

Ghoshal, G., Zlatic, V., Caldarelli, G., and Newman, M. E. J. (2009). Random hypergraphs and their applications. *Physical Reviewe E*, 79:066118.

Girvan, M. and Newman, M. E. J. (2002). Community structure in social and biological networks. In *Proc. Natl. Acad. Sci. USA*, volume 99, pages 7821–7826.

Gkanogiannis, A. and Kalamboukis, T. (2008). A novel supervised learning algorithm and its use for spam detection in social bookmarking systems. In *ECML PKDD Discovery Challenge 2008 (RSDC'08)*.

Glushko, R. J., Maglio, P. P., Matlock, T., and Barsalou, L. W. (2008). Categorization in the wild. *Trends in Cognitive Science*, 12(4):129–135.

Golder, S. and Huberman, B. A. (2006). Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208.

Goldschmidt, C. (2005). Critical random hypergraphs: the emergence of a giant set of identifiable vertices,. *Annals of Probability*, 33(4):1573–1600.

Good, B. H., de Montjoye, Y. A., and Clauset, A. (2010). Performance of modularity maximization in practical contexts. *Physical Review E*, 81(4):046106+.

Guimerà, R., Sales-Pardo, M., and Amaral, L. A. N. (2007). Module identification in bipartite and directed networks. *Physical Review E*, 76(3):036102.

Halpin, H., Robu, V., and Shepherd, H. (2007). The complex dynamics of collaborative tagging. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 211–220, New York, NY, USA. ACM.

Hartsperger, M., Blochl, F., Stumpflen, V., and Theis, F. (2010). Structuring heterogeneous biological information using fuzzy clustering of k-partite graphs. *BMC Bioinformatics*, 11(1):522.

Hassan-Montero, Y. and Herrero-Solana, V. (2006). Improving tag-clouds as visual information retrieval interfaces. In *InScit2006: International Conference on Multidisciplinary Information Sciences and Technologies*.

Hendler, J., Shadbolt, N., Hall, W., Berners-Lee, T., and Weitzner, D. (2008). Web science: an interdisciplinary approach to understanding the web. *Commun. ACM*, 51:60–69.

Heymann, P. and Garcia-Molina, H. (2009). Contrasting controlled vocabulary and tagging: Do experts choose the right names to label the wrong things? In *Second ACM International Conference on Web Search and Data Mining (WSDM 2009), Late Breaking Results Session*, pages 1–4. Stanford InfoLab.

Heymann, P., Koutrika, G., and Garcia-Molina, H. (2007). Fighting spam on social web sites: A survey of approaches and future challenges. *IEEE Internet Computing*, 11(6):36–45.

Heymann, P., Koutrika, G., and Garcia-Molina, H. (2008). Can social bookmarking improve web search? In *WSDM '08: Proceedings of the international conference on Web search and web data mining*, pages 195–206, New York, NY, USA. ACM.

Höltschi, P., Aschoff, F.-R., and Schwabe, G. (2008). wehmütig, witzig, wunderbar - subjektive tags als wegweiser im web 2.0. In *Tagungsband Multikonferenz Wirtschaftsinformatik (MKWI 2008)*.

Hotho, A., Benz, D., Jäschke, R., and Krause, B., editors (2008). *ECML PKDD Discovery Challenge 2008 (RSDC'08)*. Workshop at 18th Europ. Conf. on Machine Learning (ECML'08) / 11th Europ. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'08).

Hotho, A., Jäschke, R., Schmitz, C., and Stumme, G. (2006a). Emergent semantics in bibsonomy. In Hochberger, C. and Liskowsky, R., editors, *Informatik 2006 - Informatik für Menschen. Band 2*, volume P-94 of *Lecture Notes in Informatics*, Bonn. Proc. Workshop on Applications of Semantic Technologies, Informatik 2006.

Hotho, A., Jäschke, R., Schmitz, C., and Stumme, G. (2006b). Information retrieval in folksonomies: Search and ranking. In *Proceedings of the 3rd European Semantic Web Conference*, volume 4011 of *LNCS*, pages 411–426, Budva, Montenegro. Springer.

Jäschke, R., Eisterlehner, F., Hotho, A., and Stumme, G. (2009). Testing and evaluating tag recommenders in a live system. In Benz, D. and Janssen, F., editors, *Workshop on Knowledge Discovery, Data Mining, and Machine Learning*, pages 44 –51.

Jäschke, R., Hotho, A., Schmitz, C., Ganter, B., and Stumme, G. (2008). Discovering shared conceptualizations in folksonomies. *Journal of Web Semantics*.

Jäschke, R., Marinho, L., Hotho, A., Schmidt-Thieme, L., and Stumme, G. (2007). Tag recommendations in folksonomies. pages 506–514.

Joachims, T. (1999). Making large-scale support vector machine learning practical. In Schölkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA.

Kipp, M. E. I. (2007). Tagging practices on research oriented social bookmarking sites. http://dlist.sir.arizona.edu/2027/01/kipp_2007.pdf.

Knowledge & Data Engineering Group, University of Kassel (2008). Benchmark folksonomy data from bibsonomy, version of june 30th, 2008.

Köcher, T. (2009). *"Kanzlerin" jetzt in der SPD*. `http://www.vorwaerts.de/artikel/ae-kanzlerin-ae-jetzt-in-der-spd-0`, retrieved November 2011.

Kolda, T. G. and Sun, J. (2008). Scalable tensor decompositions for multi-aspect data mining. In *ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 363–372. IEEE Computer Society, Washington, DC, USA.

Koutrika, G., Effendi, F. A., Gyöngyi, Z., Heymann, P., and Garcia-Molina, H. (2007). Combating spam in tagging systems. In *AIRWeb '07: Proceedings of the 3rd international workshop on Adversarial information retrieval on the web*, pages 57–64, New York, NY, USA. ACM.

Krause, B., Hotho, A., and Stumme, G. (2008). The anti-social tagger - detecting spam in social bookmarking systems. In *Proc. of the Fourth International Workshop on Adversarial Information Retrieval on the Web*.

Krestel, R., Fankhauser, P., and Nejdl, W. (2009). Latent dirichlet allocation for tag recommendation. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pages 61–68, New York, NY, USA. ACM.

Kumar, R., Tomkins, A., and Vee, E. (2008). Connectivity structure of bipartite graphs via the knc-plot. In *Proceedings of the international conference on Web search and web data mining*, WSDM '08, pages 129–138, New York, NY, USA. ACM.

Lambiotte, R. and Ausloos, M. (2006). Collaborative tagging as a tripartite network. In *Computational Science – ICCS 2006*, volume 3993 of *Lecture Notes in Computer Science*, pages 1114–1117. Springer Berlin / Heidelberg.

Lancichinetti, A. and Fortunato, S. (2009). Community detection algorithms: A comparative analysis. *Phys. Rev. E*, 80(5):056117.

Lancichinetti, A., Fortunato, S., and Kertész, J. (2009). Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015.

Lancichinetti, A., Fortunato, S., and Radicchi, F. (2008). Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110.

Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791.

Lehmann, S., Schwartz, M., and Hansen, L. K. (2008). Biclique communities. *Physical Review E*, 78(1):016108.

Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. (2008). Statistical properties of community structure in large social and information networks. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 695–704, New York, NY, USA. ACM.

Li, X., Guo, L., and Zhao, Y. E. (2008). Tag-based social interest discovery. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 675–684, New York, NY, USA. ACM.

Ligges, U. and Mächler, M. (2003). Scatterplot3d - an r package for visualizing multivariate data. *Journal of Statistical Software*, 8(11):1–20.

Lin, Y.-R., Sun, J., Castro, P., Konuru, R., Sundaram, H., and Kelliher, A. (2009). Metafac: community discovery via relational hypergraph factorization. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 527–536, New York, NY, USA. ACM.

Liu, X. and Murata, T. (2009). Community detection in large-scale bipartite networks. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '09, pages 50–57, Washington, DC, USA. IEEE Computer Society.

Liu, X. and Murata, T. (2010). Detecting communities in tripartite hypergraphs. *CoRR*, abs/1011.1043.

Liu, Y., Niculescu-Mizil, A., and Gryc, W. (2009). Topic-link lda: joint models of topic and author community. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 665–672, New York, NY, USA. ACM.

Lu, C., Chen, X., and Park, E. K. (2009). Exploit the tripartite network of social tagging for web clustering. In *Proceeding of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 1545–1548, New York, NY, USA. ACM.

Marlow, C., Naaman, M., Boyd, D., and Davis, M. (2006). Ht06, tagging paper, taxonomy, flickr, academic article, to read. In Wiil, U. K., Nürnberg, P. J., and Rubart, J., editors, *Hypertext*, pages 31–40. ACM.

McGlohon, M., Akoglu, L., and Faloutsos, C. (2008). Weighted graphs and disconnected components: patterns and a generator. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 524–532, New York, NY, USA. ACM.

Meila, M. (2007). Comparing clusterings–an information based distance. *Journal of Multivariate Analysis*, 98(5):873 – 895.

Melenhorst, M., Grootveld, M., van Setten, M., and Veenstra, M. (2008). Tag-based information retrieval of video content. In *Proceeding of the 1st international conference on Designing interactive user experiences for TV and video*, UXTV '08, pages 31–40, New York, NY, USA. ACM.

Murata, T. (2009). Modularities for bipartite networks. In *HT '09: Proceedings of the 20th ACM conference on Hypertext and hypermedia*, pages 245–250, New York, NY, USA. ACM.

Murata, T. (2010a). Detecting communities from tripartite networks. In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 1159–1160, New York, NY, USA. ACM.

Murata, T. (2010b). Modularity for heterogeneous networks. In *Proceedings of the 21st ACM conference on Hypertext and hypermedia*, HT '10, pages 129–134, New York, NY, USA. ACM.

Naaman, M. (2009). Spatio-tempo-social: Learning from and about humans with social media. In *Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases*, SSTD '09, pages 1–2, Berlin, Heidelberg. Springer-Verlag.

Neubauer, N. and Obermayer, K. (2008). Predicting tag spam examining cooccurrences, network structures and url components. In *ECML PKDD Discovery Challenge 2008 (RSDC'08)*.

Neubauer, N. and Obermayer, K. (2009a). Hyperincident connected components of tagging networks. In Cattuto, C., Ruffo, G., and Menczer, F., editors, *HT '09: Proceedings of the 20th ACM conference on Hypertext and hypermedia*, pages 229–238, New York, NY, USA. ACM.

Neubauer, N. and Obermayer, K. (2009b). Towards community detection in k-partite k-uniform hypergraphs. In *Workshop on Analyzing Networks and Learning with Graphs at NIPS 2009*.

Neubauer, N. and Obermayer, K. (2010). Community detection in tagging-induced hypergraphs. In *Workshop on Information in Networks. NYU*.

Neubauer, N. and Obermayer, K. (2011). Tripartite community structure in social bookmarking data. *The New Review of Hypermedia and Multimedia, Special Issue: Social Linking and Hypermedia*, 17(3):267–294.

Neubauer, N., Wetzker, R., and Obermayer, K. (2009). Tag spam creates large non-giant connected components. In *AIRWeb '09: Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web*, pages 49–52, New York, NY, USA. ACM.

Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69:066133.

Newman, M. E. J. (2006). Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3):036104.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project.

Pass, G., Chowdhury, A., and Torgeson, C. (2006). A picture of search. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 1, New York, NY, USA. ACM Press.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Ramage, D., Heymann, P., Manning, C. D., and Garcia-Molina, H. (2009). Clustering the tagged web. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 54–63, New York, NY, USA. ACM.

Rendle, S., Balby Marinho, L., Nanopoulos, A., and Schmidt-Thieme, L. (2009). Learning optimal ranking with tensor factorization for tag recommendation. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 727–736. ACM, New York, NY, USA.

Rosvall, M. and Bergstrom, C. T. (2007). An information-theoretic framework for resolving community structure in complex networks. *Proceedings of the National Academy of Sciences*, 104(18):7327–7331.

Schaeffer, S. E. (2007). Graph clustering. *Computer Science Review*, 1(1):27–64.

Schifanella, R., Barrat, A., Cattuto, C., Markines, B., and Menczer, F. (2010). Folks in folksonomies: social link prediction from shared metadata. In Davison, B. D., Suel, T., Craswell, N., and Liu, B., editors, *WSDM'10: Proceedings of the Third ACM International Conference on Web Search and Data Mining*, pages 271–280. ACM.

Schmidt-Pruzan, J. and Shamir, E. (1984). Component structure in the evolution of random hypergraphs. *Combinatorica*, 5(1):81–94.

Seidman, S. B. (1983). Network structure and minimum degree. *Social Networks*, 5:269–287.

Selvakkumaran, N. and Karypis, G. (2006). Multi-objective hypergraph partitioning algorithms for cut and maximum subdomain degree minimization. *IEEE Transactions on CAD*, 25(3):504–517.

Serra, J. (1998). Connectivity on complete lattices. *Journal of Mathematical Imaging and Vision*, 9:231–251.

Shannon, C. E. (1948). A mathematical theory of communiation. *Bell System Technical Journal*, 27:379–423 and 623–656.

Shepitsen, A., Gemmell, J., Mobasher, B., and Burke, R. (2008). Personalized recommendation in social tagging systems using hierarchical clustering. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 259–266, New York, NY, USA. ACM.

Smith, G. (2007). *Tagging: People-powered Metadata for the Social Web (Voices That Matter)*. New Riders Press.

Stefaner, M. (2007). Visual tools for the socio-semantic web. Master's thesis, University of Applied Sciences Potsdam, Potsdam, Germany.

Sun, J.-T., Zeng, H.-J., Liu, H., Lu, Y., and Chen, Z. (2005). CubeSVD: a novel approach to personalized web search. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 382–390, New York, NY, USA. ACM.

Suzuki, K. and Wakita, K. (2009). Extracting multi-facet community structure from bipartite networks. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 04*, pages 312–319, Washington, DC, USA. IEEE Computer Society.

Symeonidis, P., Nanopoulos, A., and Manolopoulos, Y. (2008). Tag recommendations based on tensor dimensionality reduction. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 43–50, New York, NY, USA. ACM.

Tang, L. and Liu, H. (2010). *Community Detection and Mining in Social Media*. Morgan and Claypool Publishers.

Vander Wal, T. (2005). Explaining and showing broad and narrow folksonomies. http://www.personalinfocloud.com/2005/02/explaining_and_.html.

VZ.net (2009). Ich kann Kanzler!: Bei studivz/meinvz siegt Antje Krug mit 4250 Anhängern. http://blog.studivz.net/2009/06/19/ich-kann-kanzler-bei-studivzmeinvz-siegt-antje-krug-mit-4250-anhangern-groses-finale-heute-im-zdf-um-2115-uhr/, retrieved November 2011.

Wakita, K. and Tsurumi, T. (2007). Finding community structure in mega-scale social networks: [extended abstract]. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1275–1276. ACM, New York, NY, USA.

Wartena, C. and Brussee, R. (2008). Topic detection by clustering keywords. In *Proceedings of the 2008 19th International Conference on Database and Expert Systems Application*, pages 54–58, Washington, DC, USA. IEEE Computer Society.

Watts, D. J. (1999). *Small worlds: The dynamics of networks between order and randomness*. Princeton University Press, Princeton, NJ.

Wetzker, R., Umbrath, W., and Said, A. (2009). A hybrid approach to item recommendation in folksonomies. In *Proceedings of the WSDM '09 Workshop on Exploiting Semantic Annotations in Information Retrieval*, ESAIR '09, pages 25–29, New York, NY, USA. ACM.

Wetzker, R., Zimmermann, C., and Bauckhage, C. (2008). Analyzing social bookmarking systems: A del.icio.us cookbook. In *Mining Social Data (MSoDa) Workshop Proceedings, ECAI 2008*, pages 26–30.

Wetzker, R., Zimmermann, C., Bauckhage, C., and Albayrak, S. (2010). I tag, you tag: translating tags for advanced user models. In *WSDM '10: Proceedings of the third ACM international conference on Web search and data mining*, pages 71–80, New York, NY, USA. ACM.

Yanbe, Y., Jatowt, A., Nakamura, S., and Tanaka, K. (2007). Can social bookmarking enhance search in the web? In *JCDL '07: Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 107–116, New York, NY, USA. ACM.

Zhou, D., Huang, J., and Schölkopf, B. (2006). Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in Neural Information Processing Systems (NIPS) 19*, page 2006. MIT Press.

Zlatić, V., Ghoshal, G., and Caldarelli, G. (2009). Hypergraph topological quantities for tagged social networks. *Physical Review E*, 80(3 Pt 2).