Structural Graph Theory Meets Algorithms:

Covering and Connectivity Problems in Graphs

Saeed Akhoondian Amiri

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. Rolf Niedermeier
Gutachter: Prof. Dr. Stephan Kreutzer
Gutachter: Prof. Dr. Marcin Pilipczuk
Gutachter: Prof. Dr. Dimitrios Thilikos

Tag der wissenschaftlichen Aussprache: 13. October 2017

Berlin 2017
This thesis is dedicated to my family, especially to my beautiful wife Atefe and my lovely son Shervin.
# Contents

Abstract iii

Acknowledgements v

I. Introduction and Preliminaries 1

1. Introduction 2
   1.0.1. General Techniques and Models . 3
   1.1. Covering Problems . 6
      1.1.1. Covering Problems in Distributed Models: Case of Dominating Sets . 6
      1.1.2. Covering Problems in Directed Graphs: Finding Similar Patterns, the Case of Erdős-Pósa property . 9
   1.2. Routing Problems in Directed Graphs . 11
      1.2.1. Routing Problems . 11
      1.2.2. Rerouting Problems . 12
   1.3. Structure of the Thesis and Declaration of Authorship . 14

2. Preliminaries and Notations 16
   2.1. Basic Notations and Definitions . 16
      2.1.1. Sets . 16
      2.1.2. Graphs . 16
   2.2. Complexity Classes . 18
      2.2.1. Approximation Algorithms . 18
      2.2.2. Parametrized Complexity . 19

II. Covering Problems 21

3. Distributed Approximation Algorithms for Dominating Set 22
   3.1. Introduction . 22
   3.2. Definitions and General Lemmas . 23
   3.3. The LOCAL MDS Approximation . 26
      3.3.1. LOCAL MDS Approximation: Ideas and Proof Sketch . 26
      3.3.2. MDS LOCAL Approximation Algorithm . 28
      3.3.3. Analysis . 29
   3.4. $O(g)$ Approximation Factor in $O(g)$ Communication Rounds . 33
   3.5. $(1 + \delta)$-Approximation for dominating set in $H$-Minor Free Graphs . 35
      3.5.1. Ideas and Proof Sketch . 35
      3.5.2. Proof Details . 36
### 3.6. On Connected $r$-Dominating Set Problem

3.7. Conclusion and Future Works

### 4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

4.1. Introduction

4.2. Directed Minors, Directed Grids and Directed Tree-Width

4.3. The Erdős-Pósa property for Strongly Connected Digraphs

4.4. The EP-Property for Vertex-Cyclic Digraphs

4.5. Positive Instances for the Erdős-Pósa Property on Weakly Connected Digraphs

4.5.1. Strong Star

4.5.2. Two Cycles Connected by a Single Edge

4.6. Conclusion and Future Work

### III. Routing and Connectivity Problems

#### 5. On Disjoint Paths Problem in Digraphs

5.1. Introduction

5.2. Preliminaries

5.3. NP-Completeness of $UpPlan-VDPP$

5.4. A Linear Time Algorithm for $UpPlan-VDPP$ for Fixed $k$

5.5. Disjoint Paths With Congestion on DAGs

5.6. Lower Bounds on Disjoint Paths with Congestion on DAGs

5.7. Induced Path Problem

5.8. Conclusion and Future Work

#### 6. Rerouting Problem

6.1. Introduction

6.2. Model and Definitions

6.3. Preliminaries

6.4. Congestion-Free Rerouting of Flows on DAGs

6.5. NP-Hardness of 2-Flow Update in General Graphs

6.6. Optimal Solution for $k = 2$ Flows

6.7. Updating $k$-Flows in DAGs is NP-complete

6.8. Linear Time Algorithm for Constant Number of Flows on DAGs

6.9. Conclusion and Future Work
Abstract

Structural graph theory proved itself a valuable tool for designing efficient algorithms for hard problems over recent decades. We exploit structural graph theory to provide novel techniques and algorithms for covering and connectivity problems.

First, we focus on the Local model of distributed computing. In the Local model, minimizing the number of communication rounds is the main goal. We exploit the local properties of bounded genus graphs. We provide the first constant factor approximation algorithm that solves the dominating set problem in constant rounds on bounded genus graphs. Then, we arbitrarily well approximate it in $O(\log^* |G|)$ rounds. We also introduce a simple technique for graphs of bounded expansion which turns any constant factor approximation of $r$-dominating sets to a constant factor approximation of connected $r$-dominating sets.

Finding similar patterns in graphs is one of the main challenges in graph theory. One such problem is either to find many disjoint instances of a particular pattern or to find a small set of vertices such that deleting them destroys all instances of that pattern. This question first raised by Erdős and Pósa. We provide an algorithmic classification for strongly connected digraphs analogous to the classical results of Robertson and Seymour on undirected graphs. Furthermore, a good characterization for vertex cyclic digraphs is provided. In the latter, we generalize Younger’s conjecture to weakly connected digraphs.

Next, we focus on routing and connectivity problems. The first routing problem we consider is the Vertex Disjoint Paths Problem (VDPP) and its descendant problems. We provide an efficient algorithm for solving $k$-VDPP on upward planar digraphs in linear time for a fixed $k$. Then we allow the vertices to have some congestion and we solve the disjoint paths problem with congestion in acyclic digraphs. On the complexity side, we show the hardness of those problems when the corresponding parameter (e.g., $k$) is part of the input. It follows that the time complexity of our algorithms are almost optimal. We also show that induced path problem is hard even on digraphs of bounded directed tree-width.

Finally, we consider the problem of rerouting. In a computer network, we may need to reroute packets from their old paths to new paths. The rerouting procedure should satisfy some consistency rules. E.g., the capacity of links should be respected, the flow of the packets cannot be interrupted, etc. Elements of the network are asynchronous. Thus, it is not possible to do the rerouting instantly. We show that it is NP-hard to find a feasible rerouting algorithm even on DAGs. In contrast, we provide a linear time rerouting algorithm on acyclic graphs for a fixed number of paths.
Abstract

Kurzfassung

Die strukturelle Graphentheorie hat in den letzten Jahrzehnten einen wichtigen Beitrag zur Theorie der effizienten Graph-Algorithmen geleistet. Wir entwickeln weitere strukturelle Ergebnisse und neue Techniken um Überdeckungs- und Zusammenhangsprobleme auf Graphen zu effizient zu lösen.


Schließlich betrachten wir ein re-routing Problem. In einem Netzwerk soll eine Route für Pakete auf eine neue Route umgestellt werden. Dabei müssen einige Konsistenzbedingungen beachtet werden. Zum Beispiel dürfen die Routen die Kapazitätsbeschränkungen der Kanten nicht überschreiten, der Fluss der Pakete darf nicht zerstört werden usw. Wir nehmen weiterhin an, dass die Klienten des Netzwerkes nicht synchron arbeiten, daher können wir nicht annehmen, dass die neue Route von allen Klienten gleichzeitig umgesetzt wird. Wir zeigen, dass das re-routing Problem sogar auf azyklichen gerichteten Graphen NP-hart ist. Wir zeigen, dass das Problem in Linearzeit gelöst werden kann, wenn die Zahl der neu zu routenden Pfade fest ist.
Acknowledgements

First and foremost I would like to thank my advisor Stephan Kreutzer for all of his unsparing support, the immense freedom he gave me in my research, and for dedicating his time to answering my many questions. Two of my most prominent research topics started out thanks to his suggestions: routing problems (in particular disjoint paths problem), and the directed Erdős-Pósa problem. His endless support and open mindedness allowed me to discover the wonderland of research and read into many interesting topics. On a personal level, he smoothened my move to Berlin. It was my pleasure to have such a supervisor.

Special thanks go to all of my colleagues, in particular to Christoph Dittmann, Roman Rabinovich, and Sebastian Siebertz. They are not just colleagues, they are awesome friends. I will never forget all those wonderful discussions we had on various research topics. They also helped me a lot with many of paperworks in Germany. Sebastian and Roman, thanks a lot for reading parts of this thesis multiple times and providing constructive feedbacks. Thank Jana Pilz, our kind secretary for solving all my bureaucratic problems. My gratitude also goes to Stefan Schmid for bringing new topics from the network and the distributed computing communities into my research, and for his nice feedbacks on different parts of this thesis and all of your contributions along my research. Thank Sebastian Wiederrecht who contributed to this thesis not only through our many excellent discussion, but by providing beautiful figures, many of which are included in this thesis. I also wish to thank my friends Ali Alavi, Ali Golshani, and Mahmood Parham for all of their discussions and feedbacks.

Thank all of my co-authors Szymon Dudycz, Klaus-Tycho Foerster, Riko Jacob, Lukasz Kaiser, Ken-ichi Kawarabayashi, Arne Ludwig, Jan Marcinkowski, Dániel Marx, Patrice Ossona de Mendez, and Paul Wollan who provided great insights into the research topics and contributed to shape of this thesis.

Last but not least, my special thanks go to my family and especially to my partens, without whom this thesis would not be even started.
Part I.

Introduction and Preliminaries
1. Introduction

Graphs provide abstract models for real life systems. We exploit methods from graph theory to solve computational problems which improve the functionalities of those systems. Informally speaking, a graph is a set of points (vertices) some of which are connected together by edges. We model a city by a graph, where important places are represented by vertices and roads between them by edges. Then we can solve the single source shortest path problem to find a quick route from our location to the destination. Of course, to provide such an abstraction, we ignored various influential factors such as the scheduling of the traffic lights, traffic load, public transportation delays, etc. However, this abstraction helps us to provide a near optimal solution. Besides the real world applications, graph theoretic problems are interesting from the theoretical point of view.

A graph $G$ is a pair of a set of vertices $V(G)$ and a relation $E(G)$ on pairs of vertices known as edges. Graphs are either undirected or directed, i.e. each edge of a graph is either a set of pairs of vertices or it is an ordered pair of vertices. We can model symmetric networks by undirected graphs and asymmetric networks by directed graphs. For instance, the communication on the asymmetric digital subscriber line (ADSL) is asymmetric: normally download speed is greater than the upload speed in ADSL. Another example for asymmetric networks is the one-way roads of a city. Sometimes edges in the graph are weighted. We can model the speed of ADSL connections or the length of roads in the city by edge weights in a graph. Sometimes edges are capacitated. Edge capacitated graphs very often appear in the routing problems such as maximum flow problems [63], multicommodity flows [43].

Utilizing existing knowledge about graphs, we can provide efficient algorithms for some computational graph theoretic problems. An algorithm is efficient if it runs in polynomial time w.r.t. the input size. There are different levels of hardness of problems and complexity classes w.r.t. their efficiency. Namely, $P$, $NP$, $PSPACE$, $EXPTIME$, etc. Although there are efficient algorithms for only a few computational graph problems, many others are hard for $NP$. Some of the most theoretical or practically important hard graph problems are the traveling salesman problem (TSP)[155], the disjoint paths problem, the dominating set problem, etc. [69].

A common approach to cope with a computational graph problem is to either try to provide an efficient algorithm or to show that the problem is inherently hard. In this thesis, we focus on hard graph problems. To provide efficient algorithms for hard problems, we either shrink the search area or we loosen our expectations of the candidate solution. We exploit structural graph theory and parametrized algorithms to narrow the search area. On the other hand, by approximations, we provide a sub optimal yet good solutions.

In our thesis we follow three main strategies to solve difficult problems: approximation, parametrization, and structural graph theory. Here, we give preliminaries on these techniques. In the next chapter, we will explain approximation and parametrization in more details. The benefits of using structural graph theory appear in chapters III, IV, V, VI.
1. Introduction

1.0.1. General Techniques and Models

As above explained, throughout the thesis, different techniques are used to provide efficient algorithms for hard problems. Moreover, we consider problems in different computational models.

Approximations

Sometimes we are not able to provide an efficient exact algorithm for a problem. Instead, maybe it is possible to provide an approximation of an optimal solution in reasonable time. There are two major parameters to measure the goodness of approximation algorithms: accuracy and running time. Based on these parameters different classes of approximation algorithms such as FPTAS, PTAS, constant factor approximations, etc., are defined.

We can efficiently, arbitrary well approximate some hard problems, namely they have fully polynomial time approximation scheme (FPTAS) or PTAS, e.g. the Euclidean TSP admits PTAS [155]. Quite a few problems such as the metric TSP or the vertex cover problem are APX-hard but they admit polynomial time constant factor approximation algorithms [40, 87]. There are problems that are even harder to approximate. The dominating set problem cannot be approximated in polynomial time within any multiplicative factor better than $\Omega(\log n)$ [130] unless some complexity assumptions fails. Even worse is the case of TSP. There is no polynomial algorithm, with polynomial approximation bound for the TSP on general graphs [155] unless $P = NP$.

Parametrized Algorithms

In practice, we rarely get difficult instances as input. For some of them several parameters are involved and as these parameters are small we can solve the problem efficiently. For instance, in the vertex cover problem, it is crucial to keep the size of the solution as low as possible. Hence, we can fix the size of the solution (as the fixed parameter) and search for a solution of that size. If this parameter is small we can devise an efficient algorithm for the vertex cover problem. In fact, the size of the instance is not the main issue here. Studying problems by fixing their crucial parameters can lead to efficient parametrized algorithms or can provide hardness results with respect to parametrized complexity classes [46, 62].

In the world of parametrized complexity there are problems, such as the $k$-vertex cover that can be solved efficiently, more specifically they belong to FPT. i.e. they are fixed parameter tractable which means that they are solvable in time $f(k) \cdot n^{O(1)}$ where $f$ is a computable function and $n$ is the size of input. However, many important problems do not belong to FPT, unless some broadly accepted complexity assumptions fail. In fact, if certain complexity assumptions hold, there are infinitely many levels of hardness in the parametrized world. These classes of parametrized problems form the $W$ hierarchy [62]. E.g. the $k$-clique problem is complete for the parametrized class of $W[1]$. The dominating set problem is even harder. The $k$-dominating set problem is $W[2]$-complete. On top of $W$-hierarchy, there are problems such as weighted $k$-SAT problem or the natural problem of resource allocation [10].
1. Introduction

**Structural Graph Theory**

Many problems are hard in general graphs even after we parametrized the problem. Another approach is to provide parametrized approximation or to include randomization or to use parametrized randomization, etc. However, in real world graph problems, we often encounter graphs that have specific structural properties. For example the class of forests (acyclic undirected graphs) has one of the most simple structures in graphs. That is, many hard problems are easy on this class.

Forests form a proper subset of low edge density graphs. Some problems that are hard on general graphs become tractable on low edge density graphs. Among these graphs, there are well known subclasses such as planar graphs, bounded genus graphs, graphs of bounded tree-width, excluded minors, graphs of bounded expansion, and nowhere dense graph classes [125]. Each of the above classes supports additional structural properties. In particular, every cycle in a planar graph is either a face or a separator. Almost all of the abovementioned graph classes admit sublinear balanced separators. Such information could help us to provide efficient algorithms for hard problems. E.g. small separators are very helpful in devising divide and conquer algorithms. Interestingly this does not depend too much on the model of computation we choose [26, 55, 92, 138]. (see next section)

Many hard problems are easier on the restricted graph classes. E.g., the dominating set problem admits a PTAS on planar graphs [14], excluded minor graphs [75] and more generally even on graphs of polynomial expansion [77]. Such results motivate us to take advantage of structural properties of graphs in designing our algorithms. In this thesis we will deal with bounded genus graphs, acyclic graphs, graphs of bounded directed tree-width, etc.

**Local and Global Problems**

In this thesis, we will focus on two types of problems: the routing problems and the covering problems. They are from two different categories of problems. For some problems we need global information to solve them while for the others local information is sufficient. Typically routing and connectivity problems are global problems [48]. We cannot find a path between a pair of vertices without access to the whole graph. In contrast, many covering problems pose local attributes. In a local graph problem, it is possible to build a feasible solution by local neighborhood information. For many local problems, we can exploit dynamic programming to provide efficient exact or approximation algorithms. For instance, the dominating set problem is local. Every vertex of a graph can choose its own dominator.

**Computational Models**

There are different computational models, among others the centralized, the distributed, the parallel and the quantum model of computing. We focus on two types of these models: the Centralized Model and the Distributed Model of computing. In the Centralized Model, also known as the classical model, there is a single abstract machine which has the computational power of Turing machines [150]. In the Centralized Model we suppose the whole information of the graph is given to the computing machine in advance. Many centralized graph problems already have been well studied in classical graph theory.
1. Introduction

Important problems are shortest path algorithms [17], minimum cuts [117], maximum flows [63], classical domination problems [78], and etc.

We face the Distributed Model of computation in computer networks, mobile networks, etc. Unlike the Centralized Model, in the Distributed Model, we have multiple computing machines. In many cases, every element in the Distributed Model can be seen as a complete Turing machine. In the next section, we will discuss the Distributed Model of computation and then later in Chapter III we focus on the dominating sets problem in the Distributed Model of computation.

Structural Graph Theory and Models of Computation

Regardless of the model of computation we choose, we take advantage of structural properties of graphs. For undirected graphs, there are very helpful known structural properties and structural tools. One of the computationally powerful structural graph property known up to now is tree-width [19, 26, 134]. In the Centralized Model of computation, many hard problems are easy for the class of bounded tree width graphs. However, in this thesis, we are not dealing with undirected graphs in the Centralized Model. In the Centralized Model we consider directed graphs. In the Distributed Model we work on undirected graphs. Therefore, we cannot [68] use such an inspiring notion of tree-width and the related notions such as tree-depth [125], tree-cut-width [158], etc.

For the Distributed Model, we take advantage of topological properties of graphs. In Chapter III, first we study some local structural properties of graphs of bounded genus. Then, based on this understanding, we provide approximation algorithms.

Unlike undirected graphs, there is no agreed notion of a good width measure in directed graphs [68]. Hence, we do not expect to find a general successful structural meta-theorem on those graphs. Thus, in the case of directed graphs we either use the deep structural result of directed grid theorem [90] as a tool, or we focus on the acyclic graphs. In both cases, we introduce several new ideas and techniques for different problems.

In the following sections, we give an introduction to the problems and the models considered in this thesis.
1. Introduction

1.1. Covering Problems

In the covering problems, the task is to provide a subset of vertices or edges or other objects of the graph that covers all the edges, vertices, cycles or any other objects of the graph. The dominating set, the vertex cover, the feedback vertex set, the matching and the cycle cover problem are among basic covering problems. Many advanced covering problems are extensions of the above primitive problems. We tackle some fundamental covering problems on undirected and directed graphs.

For undirected graphs we introduce novel combinatorial structural properties and attributes of bounded genus graphs. We use those structural properties to provide algorithms for the dominating set problem. We then develop those techniques to apply them to the connected dominating set problem.

For the case of directed graphs we use the directed grid theorem. There we study certain structure of different digraphs. Based on these insights, we design algorithms to classify some classes of directed graphs based on their Erdős-Pósa property. In the following we provide an introduction on those problems. We will discuss them in depth in the corresponding chapters.

1.1.1. Covering Problems in Distributed Models: Case of Dominating Sets

The advent of new technologies and the growth of computer networks raised a considerable attention on distributed computing and parallel computing over the last decades. They are not just designated for supercomputers but also for everyday applications. For instance consider the following example: there is a large file that stores billions of numbers. We want to know the $k$th largest number in that file and suppose $k$ is small. This can be done with selection algorithm in linear time [25, 93]. The main issue here is that reading from the file is already a time-consuming task. We can provide a simpler algorithm as follows. Divide the input file into some smaller parts, find largest $k$ elements of each part in parallel and store them in some list in memory. Next, find the $k$th largest element among these small lists. At the first glance, it seems that this algorithm is very naive and inefficient. Amazingly, it is much faster than running a single threaded selection algorithm on the whole file. Even in nowadays personal computers and laptops, IO supports parallelization.

Distributed computing is essential in huge networks. There are many routers in the network and each of them can provide services to the clients. Every client should choose a good router as its service provider. However, usually we cannot afford a lot of servers and every client should reach its corresponding server quickly. We can model the clients and the servers as nodes of a graph, we add an edge between a pair of nodes if their corresponding network elements can reach each other quickly. The task boils down to finding a small dominating set. The issue here is that there are billions of nodes in the network. The communication cost is not negligible. Thus not all of them can communicate together in order to figure out the whole structure of the graph. On the other hand, every node is a computational machine and can communicate with its neighborhood. Every node in the graph should determine its own server i.e. dominator. Here we can benefit the distributed computing and distributed algorithms as we will see later. Distributed computing plays an important role in some other scenarios such as distributed routing algorithms [45].
Distributed Models of Computation: Local Model

There are two major types of distributed models based on their communication attributes: Synchronous and Asynchronous models of computation. We explain the former in the following, we discuss the asynchronous model later. In the synchronous model of computation, we assume that there is a central clock, every processor is synchronized with a central clock and moreover all processors have the same computational power. This model of computation is very abstract as the real world problems are not synchronized that way. However, we can pay additional costs to make the real world model closer to synchronized model. For instance, we can consider a big interval of time as one unit of time (a round) and pay for the extra time we waste. Hence we can somehow assume that the clocks are synchronized and moreover we assume that all machines have the same power. For more details on synchronous model we refer the reader to [111].

In the synchronous model of distributed computing, there are different kinds of computational models. Clearly, ideally we would like to know the structure of the whole network (graph) in advance. However, this is beyond our power as we cannot spend too much time on the communications. Instead, as we explained in the previous section, we can solve the problem locally. We use the local information and we rely on the local subsolutions. In fact, we solve the local subproblems and at the end aggregate them into a single general solution. If we aim to an optimal solution, most often we have to consider the whole input and hence we have to spend time on communication rounds. If a suboptimal solution suffices, sometimes we can save some communication rounds. A good local solution should be cheap with respect to specific parameters. One crucial parameter is the number of communication rounds. The other parameter is the gap between the local solution and an optimum solution. In fact, a suitable local solution is required to give a good approximation of an optimal solution. The local algorithm is a powerful approach to deal with big instances. In fact, a constant-time local algorithm can be executed on a big problem instance such that its running time is independent of the instance size.

The message passing model is one of the most interesting practical models in distributed computing. In the message passing model, we have a network of computing machines denoted by a graph $G = (V, E)$. Every vertex (node) in the graph has infinite computing power and they can communicate with their neighbors via their communication links (edges). Furthermore, as we already explained nodes are synchronous, it means that at each communication round they will send (receive) messages to (from) their neighbors at the same time. Every link can carry a message of size $B$, $B$ is the bandwidth of the network. If we assume that the bandwidth of the network is unbounded, then we are in the LOCAL model [56, 105, 127]. Given a distributed graph problem, we aim to provide a solution that uses few number of communication rounds.

A local distributed graph algorithm $A$ runs on every node of the graph in $r$ rounds. In the LOCAL model, usually $r$ is a constant independent of the input size. $r$ also called as the local horizon of a local algorithm [147]. If the bandwidth is restricted to $O(\log n)$ for the input of size $n$, we are in the CONGEST model. Each of these two models has its own advantages. The CONGEST model is more suited for dense networks, where the amount of data flow a communication link incurs is a big issue. Whereas the LOCAL model is helpful for sparse networks. In sparse networks, not every node has many neighbors so congestion of links does not cause big delays. In addition to that, the LOCAL model provides an abstraction that
helps us to understand the CONGEST model better.

Various types of problems in the LOCAL and the CONGEST model of computation have been considered. These problems include routing and connectivity problems such as the shortest path problem [57, 79] or covering problems such as the spanning tree, the dominating set, the independent set and the matching problems [22, 100].

**Dominating Sets vs Independent Sets: Local vs Global**

We consider the dominating set problem in the LOCAL model. A set of vertices $D$ is a dominating set of a graph $G = (V, E)$ if every vertex in $V − D$ is adjacent to a vertex of $D$. The dominating set problem is to find a dominating set of a graph. For a given instance, among all such dominating sets, a set of the least size is an optimum solution. In the first step, to get familiar with the distributed settings, we do not aim at optimal or even a near optimal solution. Instead just a feasible solution suffices. Consider the following simple algorithm: every vertex chooses one of its neighbors arbitrarily as a dominator. This already provides a valid solution for the dominating set problem. However, it can be very far from the optimum solution. However, such a simple algorithm provides a constant factor approximation for the dominating set problem in certain graph classes such as the class of 3-regular graphs. In the next step, we improve this naive algorithm with a simple greedy choice: every node arbitrarily chooses a neighbor with the highest degree as its dominator. Although this does not give an optimum or an approximate solution in general graphs, it performs in constant rounds and for some instances, like a wheel graph, it may provide a better solution than the naive algorithm.

In contrast, consider the case of the independent set problem. A subset of vertices of a graph forms an independent set if there is no edge between any pair of them. In the independent set problem, we have to find an independent set. Here the goal is to maximize the size of independent set.

Contrary to the dominating set problem, in the independent set problem, neither each node can choose itself to be in the independent set, nor it can choose any of its neighbors to be so. Such choices can conflict each other: a pair of nodes are in the independent set but also there is an edge between them. One can observe that there is no way to resolve this conflict even in a simple ring (simple cycle). To overcome this issue, the technique of symmetry breaking has been introduced [84]. We can equip every node with a unique identifier. Hence, one node can put itself into the independent set if it has the lowest identifier among its neighbors.

This already shows that finding a small dominating set in the LOCAL model is in a sense easier than finding a big independent set. Our simple algorithm can lead to a very bad solution. For instance in a long path at every round, we may choose only one vertex to be in the independent set: e.g. if every vertex, except exactly one of the endpoints of the path, has a neighbor with a smaller identifier. It is not straightforward to significantly improve this naive algorithm already in simple structures like rings [42, 105].

The situation gets worse in the case of connectivity problems. E.g. checking whether a graph is connected is not first-order definable. Performing DFS in a graph has a trivial lower bound of $\Omega(d)$, here $d$ is the diameter of the graph. So we cannot provide a valid solution in a constant number of communication rounds already for straightforward classical connectivity problems such as DFS. Connectivity problems are very interesting and it is
good to see whether we can improve the existing distributed algorithms, however, we do not consider them in the distributed part of this thesis.

**What are we looking for?**

In this thesis, we investigate how to use the power of local computation for the problems such as the dominating set problem. Is it possible to provide exact or approximation solutions to the problem? We discuss this in the Chapter III. In the LOCAL model of computation we provide the first constant factor approximation algorithm for a minimum dominating set in a constant number of rounds on bounded genus graphs. We also well approximate it with $O(\log^* n)$ extra rounds. On the other hand, we study the connected dominating set problem and will give the first constant factor approximation on planar graphs. To achieve these results we provide novel analysis and algorithmic tools for bounded genus graphs which might be helpful for other problems in the LOCAL model.

1.1.2. Covering Problems in Directed Graphs: Finding Similar Patterns, the Case of Erdős-Pósa property

Often it is important to find common patterns in graphs. For instance in various clustering problems in social networks, one of the most prominent parameters is the clustering coefficient \[156\] of nodes. This parameter helps to determine the likelihood of nodes and eventually cluster them accordingly. In some of these problems, the clustering coefficient of a vertex $v$ is defined to have a direct relation to the number of edges with both ends in the (open) neighborhood of $v$ and an inverse relation with the degree of $v$ squared. So if there are many triangles, all of them intersecting at a single vertex $v$, then the clustering coefficient of $v$ is also large. Thus this parameter is closely related to the number of triangles in the graph \[59, 140\].

Underneath such practical motivations of studying common substructures in graphs, there are rich theoretical materials. For instance, a more general version of the triangle finding problem is to find cycles of particular length in the graph, either edge disjoint or vertex disjoint. This problem is known as the cycle packing problem \[27, 33, 142\]. To get a better understanding of the cycle packing problem we can take a look at its dual problem. A set of vertices that hits every cycle in the graph is called a feedback vertex set. The associated problem, to check whether a feedback vertex set of a certain size exists, is the feedback vertex set problem \[39, 69\].

In 1965 Erdős and Pósa showed that every graph either has many pairwise (vertex) disjoint cycles or it has a small feedback vertex set. Nowadays this is known as the Erdős-Pósa property. We can generalize the Erdős-Pósa property in two steps. First, we replace cycles with subgraphs then we replace subgraphs with minors. I.e. the corresponding problem formulates as follows: let $H$ be a fixed graph, is there any relation between the number of disjoint (minor) models of $H$ in any graph $G$ and the size of a minimum set that hits all models of $H$ in $G$? The problem of natural generalization of the Erdős-Pósa property was resolved by Robertson and Seymour who showed that the answer to above question is positive if and only if the graph $H$ is planar.

Several years after Erdős and Pósa’s great result, Younger raised his famous conjecture for directed graphs (1973):
1. Introduction

“There is a function \( f : \mathbb{N} \rightarrow \mathbb{N} \) such that for all \( k \) and all digraphs \( G = (V, E) \), \( G \) contains \( k \) pairwise disjoint cycles or there is a set of vertices \( T \subseteq V \) such that \( G - T \) is acyclic and the size of \( T \) is bounded from above by \( f(k) \).”

Unlike undirected graphs, dealing with directed cycles was not easy and it took about a quarter of a century for researchers to answer Younger’s conjecture positively.

As for undirected graphs one can consider the more general question for digraphs:

“Is there a function \( f : \mathbb{N} \rightarrow \mathbb{N} \) such that for every digraph \( H \) and for all \( k \) and every digraphs \( G = (V, E) \), \( G \) either has \( k \) disjoint butterfly models of \( H \) or there is a set of vertices \( T \) of size at most \( f(k + |H|) \) such that \( G - T \) has no butterfly model of \( H \)?”

In other words, the question is whether we can either find many disjoint models of a directed graph or a small set of vertices which they hit all such models.

We answer the above question when the graph \( H \) is strongly connected by providing a full classification of the class of graphs that have this generalized property. On the other hand, we provide a classification of the vertex cyclic graphs (the class of graphs where every vertex lies on some cycle) with respect to Erdős-Pósa property. We show that many natural graphs in that class do not have the Erdős-Pósa property. We also provide a affirmative instance of Erdős-Pósa property in vertex cyclic graphs. This positive instance extends the results for Younger’s conjecture to a more general subset of weakly connected digraphs. This covered in Chapter VI. There we develop new tools to analyze directed graphs.
1. Introduction

1.2. Routing Problems in Directed Graphs

Routing is the process of selecting a path for traffic in a network. In the early 1960s, researchers established the foundations of routing algorithms in graphs. Dijkstra [54] provided an elegant single source shortest path algorithm and Bellman [18] used newly established techniques of dynamic programming to provide an algorithm for the all pair shortest paths problem. Later, Ford and Fulkerson [63] considered the more general problem of routing flows in networks.

The length or cost of a path which is chosen for routing can be measured with respect to different parameters: these include the number of hubs a packet needs to travel to reach the destination, physical distance between the source and the destination, the traffic of the network and security constraints, etc [5, 7, 123, 149]. Many algorithms and protocols have been designed to address these issues, e.g. see [16, 52, 85]. We considerably simplify the problems by modeling them as a simple graph problem.

We consider two major types of graph routing problems:

1. **Finding routes from sources to destinations**, e.g. finding shortest paths between a pair of vertices, finding disjoint paths between pairs of vertices, finding flows, etc.

2. **Rerouting or reconfiguration problems**, e.g. when routes from a source to a destination are already known and the task is to reroute them while maintaining some feasibility constraints.

1.2.1. Routing Problems

To measure the connectivity of a network one can rely on the vertex connectivity of the underlying graph, that is, the number of vertex disjoint paths between every pair of vertices. In 1927, Menger [117] proved that the number of vertex disjoint paths between two vertices is equal to the size of a minimum vertex cut between them. To find the corresponding disjoint paths, it is possible to turn Menger’s proof into an algorithmic proof. This is basically similar to standard flow algorithms. A similar theorem was later proved for edge connectivity. It is possible to extend the Menger’s theorem to sets of vertices: Let $G$ be a graph and let $A, B \subseteq V(G)$ be two sets of vertices of $G$. Then the number of vertex disjoint paths with one end in $A$ and the other end in $B$ is equal to the size of a minimum vertex cut which disconnects $A$ and $B$. Again there are several polynomial time algorithms to either find those disjoint paths or a small vertex cut.

In many applications, set-wise connectivity alone is not sufficient. E.g. in communication networks, different agents communicate with each other, so we seek pairwise disjoint paths between pairs of vertices and not just between any vertices of the given sets. A similar situation arises in traffic engineering, where it may be important to have disjoint paths between them [15, 149], as otherwise, some packets have to wait for others, but often a link or node can tolerate some congestion. The fewer a node is used, the faster the resulting communication. Another classical application is VLSI design [34, 95], where it is necessary to wire different points together and the wires must not intersect each other.

We consider the disjoint paths problem as one of our main routing problems. Formally, the problem is to find $k$ disjoint paths between $k$ pairs of source and terminal vertices $(s_1, t_1), \ldots, (s_k, t_k)$. This problem is known as the *k-vertex disjoint paths problem*. 

11
The disjoint paths problem, especially in directed graphs, is a challenging and a hard problem. We will give further technical details on its hardness in chapter V. Similar to the case of the dominating sets problem, instead of considering the problem in general graphs, we try to solve it on restricted graph classes. We consider acyclic digraphs and a weaker notion, namely upward planar digraphs.

Acyclic digraphs are not only theoretically interesting but also practically relevant. E.g., once we want to send a flow along some nodes in a network, we usually want to avoid a circulation of the flow: it is not just time-consuming but also it can cause packet loss due to high congestion or it may lead to inappropriate order of packets.

We consider the disjoint paths problem in the special case of acyclic graphs and provide a new efficient algorithmic technique. We also consider the problem of disjoint paths with congestion and give a new simple but elegant algorithm that solves the problem in acyclic digraphs. We try to solve a related routing problem in graphs of bounded directed tree-width (a generalization to undirected tree-width). We will see that the very simple looking routing problem of induced path remains hard on these graphs.

1.2.2. Rerouting Problems

Reconfiguration problems have received considerable attention in theoretical computer science [29, 31, 74, 76, 120–122, 154]. In such problems we are interested in whether we can transform a source configuration to a target configuration when only certain transformations are allowed [154]. We are especially interested in rerouting paths [30, 120].

Rerouting of paths appears very often in computer networks. However, in a real network, there are plenty of paths each of which routes a flow of data. Therefore, our objective is to generalize the path reroutings and reroute multi-commodity flow.

Rerouting requests can happen due to change of traffic demands, security reasons or because of failure on some nodes or links in the network, etc. There are several constraints that a rerouting procedure must respect. Of foremost importance are safety constrains, see e.g. [64, 72, 108, 133] and we will shortly explain them later in this section.

We consider the rerouting problem in asynchronous networks. A Software Defined Network (SDN) [153], provides an abstract model of a network. This helps us to focus on the high level issues such as routing and rerouting and avoids problems which can arise in the lower levels such as packet loss, etc [3]. In SDN’s, we assume that there is a controller who sends the rerouting commands to the nodes. Controller commands have the following format: activate/deactivate an outgoing link $e$ of a node $v$ for some flow $i$. So, for now, we can consider its commands as a tuple $(node, link, path/flow, activation code)$.

Nodes will update their routing tables according to commands of the controller. Every link (or edge) in the network is capacitated. If multiple flows route packets along a single link, they have to respect the capacity constraints of the link. A formulation for such constraints can be found in [128]. In short, the network should be congestion free.

By the controller command, any node might update its packet forwarding rules. If the new flows are congestion free, clearly by the end of the rerouting procedure, every link is congestion free. So ideally we would like to update all nodes instantaneously. Unfortunately, in reality, we cannot update all nodes instantly. The main issue here is that nodes in the network are not completely synchronous. We can just suppose they are synchronous at the end of each so called round. For example rerouting outgoing links of two nodes at the
1. **Introduction**

same time may cause a packet loss, due to the transient blocking of network traffic. We will explain this in more details by examples in Chapter VI. Common issues in multi-commodity flow rerouting are as follows:

1. There can be a moment such that there is no flow of data between a source and destination pair. This interrupts the flow.

2. It is possible that one packet falls in an unwanted loop. This is bad, beyond delay and extra costs it can also destroy the order of packets.

3. Some nodes are slower. They may deactivate their forwarding old edges a bit too late. In contrast, some others are faster, they activate their new flow forwarding links earlier. Now there can be a situation in which one link carries extra flow due to the fact it routes some old flows and some new flows. This violates the congestion of the link and can again cause a data loss. Notice that this scenario can occur if there are at least two different commodities.

Therefore we cannot simply update all forwarding rules in a single round. We have to update the old routing plan to the new one in a sequence of rounds \( \rho_1, \ldots, \rho_\ell \) in a consistent way. The rerouting should obey the *consistency* rules. These consistency rules (should) prevent the above problems. Now we can informally define our problem.

The problem of multi-commodity flow rerouting consists of a set of \( k \) pairs of flow: old and update pairs \((f_1^o, f_1^u), \ldots, (f_k^o, f_k^u)\). Each old flow \( f_i^o \) (or update flow \( f_i^u \)) is a single path with demand \( d_i \). At the start, all edges in \( f_i^o \) for flow \( i \) are active and edges of \( f_i^u - f_i^o \) are inactive. Note that every edge can have up to \( 2^k \) states: \( f_i^o \) (or \( f_i^u \)) is active/inactive for \( i = 1, \ldots, k \). The task is to reroute flows from \( f_i^o \) to \( f_i^u \) in a consistent way in a sequence of rounds. The following is the informal definition of *consistency* rules, later in the last chapter we will define it formally.

1. At any moment, the amount of flow a link incurs should be upper bounded by the capacity of that link.

2. At any moment (e.g. during a round), every commodity should be satisfied.

3. Having a loop during updating a specific flow must be avoided. So the subgraph which carries a specific flow should be acyclic. In our case it is required that for every commodity there is exactly a single path routing a flow.

The above consistency rules are discussed in previous works [12, 64, 108].

If there is only a single flow, it is easy to see that a simple update schedule always exists: we can update switches one by one, proceeding from the destination toward the source of a route. In practice, however, it is desirable that updates are fast and new routes become available quickly: In order to be able to use as many new links as possible, one aims to maximize the number of concurrently updated switches [64]. This approach is known as the *greedy approach*.

We will discuss the multi-commodity flow rerouting problem in acyclic graphs. Interestingly the problem is NP-hard in acyclic graphs but we show that it is fixed parameter tractable when the number of commodities is the parameter.
1. Introduction

1.3. Structure of the Thesis and Declaration of Authorship

This thesis consists of four technical chapters. Each chapter focuses on one of the above introduced problems. Before starting with technical chapters, in Chapter II we fix our notation and provide general definitions used throughout the thesis. In the beginning of each chapter, there are chapter specific definitions and notations. Hence, in addition to Chapter II, every chapter has its own preliminaries and definitions. Every technical chapter ends with its own conclusion. In each one, first, we give a summary of what we did and then we propose some interesting open questions.

Chapter III: The Dominating Set Problem

This chapter is dedicated to the dominating set problem in the Local model. It is built on the following papers:


All the mentioned papers were results of numerous hours of collaborative work among the authors. In this chapter, only the parts to which I had major contributions are presented.

Chapter IV: The Erdős-Pósa property in Digraphs

In Chapter IV we consider the classical problem associated to Erdős-Pósa property. The skeleton of Chapter IV is based on the following paper and some unpublished works.


In Chapter IV my major contribution was in developing algorithms and proving affirmative instances. Many of the ideas in regard to counter examples were developed by my co-authors and we present them here for the sake of completeness.

Chapter V: On Disjoint Paths Problem in Digraphs

Chapter V considers some hard routing problems, with a focus on the disjoint paths problem. It is based on the following papers and some unpublished works.
1. Introduction


Although the main idea of the hardness in the first paper does not belong to me, I collaborated in its proof and for the sake of completeness it is included in the thesis.

From the second paper I included only the parts that I was the major contributor.

Chapter VI: The Flow Rerouting Problem

Chapter VI is devoted to the problem of rerouting multicommodity singleton flows in the asynchronous model. It is based on the following papers.


This chapter is mainly based on the second paper, as it presents a more generic, affirmative solution of the problem address in the first paper. Hence, I only included the parts of the second paper that I was the major contributor to.
2. Preliminaries and Notations

In this chapter, we provide the notations and definitions which are used in the whole thesis.

2.1. Basic Notations and Definitions

2.1.1. Sets

We denote the set of integers with \( \mathbb{Z} \) and the set of non-negative integers known as natural numbers with \( \mathbb{N} \), we write \( \mathbb{R} \) for the set of real numbers. We write \( [n] \) for the set \{1, \ldots, n\}. We denote a powerset of a set \( S \) by \( 2^S \). A partition \( P \) of a set \( S \) is a set of non-empty and pairwise disjoint subsets \( S_1, \ldots, S_k \) of \( S \).

2.1.2. Graphs

A graph (digraph) is a pair \( G = (V, E) \) where \( V \) (\( V(G) \)) is a set of vertices and \( E(G) \) is a multiset of pairs (ordered pairs) of vertices. For a graph (digraph) \( e \) with pair of vertices \( \{u, v\} \) (ordered pair of vertices \( (u, v) \)) we write \( e = \{u, v\} \) (resp. \( (u, v) \) for digraphs). For undirected graphs we call \( u \) and \( v \) the endpoints of \( e \), for digraphs \( u \) is the tail of \( e \) (\( tail(e) \)) and \( v \) is a head of \( v \) (\( head(e) \)). If both endpoints of an edge \( e \) are identical then we say \( e \) is a loop.

In a graph (digraph) \( G \) two edges \( e, e' \) are parallel if they have the same endpoints (\( h(e) = h(e') \) and \( t(e) = t(e') \)). A graph (digraph) is simple if it does not have parallel edges. Whenever we write graph (digraph) we mean a simple graph except when we explicitly clarify it. We write \( G := G_1 \cup G_2 \) to denote the union of two disjoint graphs \( G_1, G_2 \).

Walks, Cycles, Induced Graphs, Strongly Connected

A (directed) walk \( W \) in a graph \( G \) is an alternating sequence of vertices and edges \( v_0, e_1, \ldots, e_m, v_m \) where endpoints (tail and head) of \( e_i \in [m] \) are \( v_{i-1} \) and \( v_i \), we say \( v_0 \) is the start of the walk \( W \) and \( v_m \) is the end of the walk \( W \), all other vertices are internal vertices of the walk, in other words \( W \) is a \( (v_0, v_m) \)-walk. A walk \( W \) is a closed walk if its start and end vertices are identical. A (directed) path \( P \) in a graph \( G \) is a (directed) walk such that every vertex appears only once in \( P \). A (directed) cycle \( C \) in a graph \( G \) is a closed walk where every vertex except the first (and the last one) appears only once. A (digraph) graph \( G \) is (strongly) connected if for every pair of vertices \( u, v \in V(G) \) there is a (directed) path with start vertex \( u \) and end vertex \( v \). The underlying undirected graph \( G' \) of a directed graph \( G \), is an undirected graph obtained from \( G \) by ignoring the edge directions. A directed graph is weakly connected if its underlying undirected graph is connected. A graph \( S \) is a subgraph of \( G \) if \( E(S) \subseteq E(G) \) and \( V(S) \subseteq V(G) \) such that \( V(S) \) contains all endpoints (head and tails) in \( E(S) \) and we write \( S \subseteq G \). We may consider a path \( P \) and a cycle \( C \) as subgraphs of the graph \( G \). A subgraph \( S \subseteq G \) is an induced subgraph of \( G \) if
2. Preliminaries and Notations

\[ E(S) \] contains all edges of \( G \) with endpoints (head and tail) in \( V(S) \), we write \( G[S] \) for the induced subgraph \( S \) of \( G \). For \( S \subseteq V(G) \) we write \( G - S \) for the graph \( G[V(G) \setminus S] \).

**Length of a Path, Endpoints of Paths, Lexicographically Shortest Paths**

\( \ell(p) \) denotes the length of a path \( p \) (number of edges in \( p \)) and we write \( \text{distance}(u,v) \) for the distance (number of edges in the shortest path) between a pair of vertices \( u,v \). We say a path \( p = v_1,\ldots,v_k \) is a \((u,v)\) - path if \( v_1 = u, v_k = v \), \( u \) and \( v \) are endpoints of \( p \). If a graph \( G \) is a vertex labeled graph such that its labels (\( Label \)) are comparable, for a path \( p = v_1,\ldots,v_k \) the lexicographical order of vertices of \( p \) is \( Label(v'_1),\ldots,Label(v'_k) \) where \( \{v_1,\ldots,v_k\} = \{v'_1,\ldots,v'_k\} \) such that \( Label(v'_1) < \ldots < Label(v'_k) \). We say a path \( p \) is the lexicographically shortest \((u,v)\) - path if \( p \) is a \((u,v)\) - path and \( \ell(p) \) is minimum among all \((u,v)\) - path’s and additionally, among all such paths, it is the lexicographically smallest one. We say a path \( p \) is an \( \ell \)-path if \( \ell(p) \geq \ell \). We denote the start vertex of the path \( p \) by \( \mathcal{S}(p) \) and the end vertex of \( p \) by \( \mathcal{E}(p) \).

**Components and Strong Components**

In a (digraph) graph \( G \) a (strong) component is a maximal (strongly) connected subgraph of \( G \). In a directed graph \( G \), weak components are maximal weakly connected subgraphs of \( G \). A component is trivial if it has only one vertex.

**Tree, directed tree, arborescence**

A tree is an undirected connected graph without a cycle. A rooted tree is a tree with a root vertex \( r \), the height of a vertex \( v \in V(T) \) is the length of a \((v,r)\)-path and the height of a \( T \) is the length of a longest path in \( T \) where ends at \( r \). A directed tree is a directed graph without loops such that its underlying undirected graph is a tree. An arborescence is a directed tree with a root vertex \( r \) such that for every vertex \( u \in V(T) \) there is a directed path which starts at \( r \) and ends at \( u \). If the directed tree \( T \) is an arborescence we denote it by \( \hat{T} \). A \( k \)-connected component of a graph \( G \) is a subgraph \( S \) of \( G \) such that for every pair of distinct vertices \( u,v \in V(S) \) there exists at least \( k \) internally vertex disjoint paths between them.

**Neighborhood of Vertices**

For a (digraph) graph \( G \) and a vertex \( v \in V(G) \), neighbors of \( v \) are all vertices \( u \) such that \( \{u,v\} \in E(G) \) (\( (u,v) \in E(G) \) or \( (v,u) \in E(G) \)). We write \( N[v] \) for the closed neighborhood of \( v \). The \( r \)-neighborhood of a vertex \( v \), \( N^r(v) \), is defined in a similar way. In an undirected graph, a vertex \( u \) belongs to \( N^r(v) \) if there exists a \((u,v)\) - path of length at most \( r \). For a set \( A \subseteq V(G) \), we denote by \( N[A] \) the set \( \bigcup_{v \in A} N[v] \).

**Edge Contraction and Minor**

In an undirected graph \( G = (V,E) \) we contract an edge \( \{u,v\} \) and obtain a graph \( G/e \) by adding a new vertex \( uw \) to \( G \) and connecting all neighbors of \( u \) and \( v \) to the vertex \( uw \) then removing the vertices \( u \) and \( v \). We say a graph \( H \) is minor of \( G \) if \( H \) can be obtained from \( G \).
by taking a subgraph of $G$ and applying some edge contractions recursively, we write $H \preceq G$. In other words the graph $H$ is a minor of a graph $G$ if there are set of pairwise vertex disjoint connected subgraphs $G_1, \ldots, G_{|H|}$ of $G$ such that for every edge $e = \{v_i, v_j\} \in E(H)$, there is an edge between a vertex of $G_i$ and a vertex of $G_j$. We introduce analogous definitions and notations for digraphs in Chapter IV.

### 2.2. Complexity Classes

We assume basic familiarity with decision problems and complexity classes $P, NP$ and refer the reader to the books [13, 69] for more background. In the following we present more information on approximation algorithms and parameterized complexity.

#### 2.2.1. Approximation Algorithms

An approximation algorithm is an efficient algorithm for a maximization/minimization optimization problem which produces an approximate solution instead of an exact solution. We refer the reader to the book [155] for detailed background on approximation algorithms. In the following, $OPT$ denotes the size of an optimum solution for a particular instance $I$, $OPT^*$ denotes the size of an approximated solution, $c$ refers to some constant. We also suppose all problems are maximization problems, the analogous case of minimization is easy to obtain.

**Approximation Factor:** Given a function $f : \mathbb{N} \to \mathbb{R}$, an algorithm $A$ provides an $f$-approximation to a maximization problem $P$, if for every instance $I$ of $P$ we have $OPT \leq f(|I|)OPT^*$.

1. **Approximation with additive constant error:** Approximation algorithm $A$ is an approximation with additive constant error if for all inputs to $A$ we have that $|OPT^* - OPT| \leq c$. Having an algorithm with such precise approximation guarantee is ideal in many cases. This sort of approximation/inapproximability, most often shows up when we analyze strategical games e.g. see [11, 50, 139].

2. **Fully Polynomial Time Approximation Scheme (FPTAS):** The approximation algorithm $A$ is an FPTAS if for every $0 < \epsilon < 1$ and input of size $n$ the algorithm runs in time $Poly(1/\epsilon + n)$ and provides a $(1 - \epsilon)$-approximation to the optimal solution. E.g. the Knapsack problem admits an FPTAS [155].

3. **Polynomial Time Approximation Scheme (PTAS):** The approximation algorithm $A$ is a PTAS if for every $0 < \epsilon < 1$ and input of size $n$ the algorithm runs in time $f(n, \epsilon)$ and provides a $(1 - \epsilon)$-approximation to the optimal solution. E.g. the Bin Packing problem admits a PTAS [155].

4. **APX:** A class of problems with a polynomial time constant factor approximation algorithm belongs to APX. This includes e.g. the metric TSP [40]. On the other hand metric TSP at a same time is hard for APX [87].

5. **Beyond Constant Factor Approximations:** There is no sublogarithmic factor approximation algorithm for some problems, (e.g. the dominating set problem [130]).
2. Preliminaries and Notations

There are problems such that for every function \( f : \mathbb{N} \rightarrow \mathbb{R} \) there is no polynomial time \( f \)-approximation algorithm for them. For instance the TSP cannot be approximated by any computable function in polynomial time.

Sometimes we cannot provide a good approximation algorithm but it is possible to provide a good approximation with high probability. This introduces another concept of randomized approximation algorithms such as fully polynomial time randomized approximation scheme (FPRAS) and PRAS [155]. We do not deal with randomized algorithms hence we omit their definitions.

2.2.2. Parametrized Complexity

Sometimes finding a solution for a hard problem on practical instances is easy. This can be due to the fact that the hard instances do not appear in practice. The difficulty can occur when one of the parameters of the problem is big. E.g. as we will discuss later, finding 2-vertex disjoint paths can be done quickly in undirected graphs but finding \( k \)-disjoint paths is NP-hard for general \( k \). Here the bottleneck is the number of paths. In fact the \( k \)-disjoint paths problem can be solved in \( f(k)n^{O(1)} \), hence for a fixed \( k \) it is polynomial time solvable in undirected graphs [135]. There are other problems where if we fix a parameter, then there is a polynomial time solution for them. To get a better feeling about the role of parameterization, consider the dominating set problem. We are looking for a set of at most \( k \) vertices whose neighborhood is the whole graph. We can simply guess those at most \( k \) vertices and then check whether their closed neighborhood is the whole graph. This leads to an \( n^{O(k)} \) algorithm for the dominating set problem. If the set is small (let say 3), this gives us an exact algorithm in the reasonable time without backtracking.

Those observations motivate us to study the problems from the viewpoint of parameterized complexity. A parameterized problem (language) \( L \) is a subset of \( \Sigma_L^* \times \mathbb{N} \) for some alphabet \( \Sigma_L \). We parameterize an instance \( I \) of the problem \( L \) with a parameter \( k \) and denote it by \( (I,k) \). The question is to decide whether the instance \( (I,k) \in L \).

Like polynomial time reductions, we have the concept of reductions in parameterized complexity. Polynomial reductions are not necessarily helpful in the parameterized context. E.g. consider the instance \((G,k)\) of the vertex cover problem. Here the question is to find a set of vertices \( S \) of order at most \( k \) such that every edge \( e \in E(G) \) is adjacent to a vertex of \( S \). We know the trivial duality relation between the vertex cover and the independent set. I.e. the maximum independent set of \( G \) has size \( |V(G)| - k \) if and only if the size of minimum vertex cover is \( k \). So the reduction is a direct reduction from a vertex cover instance \((G,k)\) to an independent set instance \((G,|V(G)| - k)\). This reduction does not preserve the parameter \( k \). In fact, \( V(G) \) is not bounded by any function of \( k \). Hence it is not useful for the context of parameterized complexity.

FPT reductions: A parameterized problem \((L_1,k_1)\) is reducible to a parameterized problem \((L_2,k_2)\) if there is a mapping \( \mu : \Sigma_{L_1} \rightarrow \Sigma_{L_2} \) and computable functions \( f,g : \mathbb{N} \rightarrow \mathbb{N} \) such that the following conditions hold:

1. For every \( x \in \Sigma_{L_1} \) we have \( x \in L_1 \iff \mu(x) \in L_2 \).
2. For every instance \((I_1,k_1)\) of \( L_1 \) the mapping \( \mu \) can be computed in time \( f(k_1)poly(|I_1|) \).
2. Preliminaries and Notations

3. For every instance $x$ in $L_1$ with parameter $k_1$ (i.e. $k_1 = k_1(x)$) we have that $k_2(\mu(x)) \leq g(k_1(\mu(x)))$.

The above definition is an intuitive form of the standard definition from the book [62]. For more precise definitions and more background we refer the reader to the above mentioned book or the recent algorithmic book [46].

These reductions introduced the parameterized complexity classes. These classes are $\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \ldots \subseteq \text{XP}$. We give informal definitions and examples of these classes in the following. The actual definition of the classes is by the weft of their corresponding circuits, but as it is not relevant to this thesis we are not going into such details. Instead, we provide definitions which might give a better understanding about the behavior of parameterized problems. In the following, $n$ denotes the size of the input.

1. **Fixed Parameter Tractable or FPT**: A parameterized problem $(L, k)$ is fixed parameter tractable if there is a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ and an algorithm $A$ that answers $x \in (L, k)$ in time $f(k) \cdot \text{Poly}(n)$. Clearly every polynomial time solvable problem is fixed parameter tractable. Problems such as vertex cover problem and chordal deletion [114] admit FPT algorithm.

2. **W[1]-complete problems**: A problem is W[1]-complete if and only if, it admits an FPT reduction to/from the $k$-clique problem. It is widely accepted that a W[1]-complete problem is not fixed parameter tractable. In other words, we cannot expect anything better than $n^{f(k)}$ algorithm for them.

3. **W[2]-complete problems**: One of the canonical problems in this class is the dominating set problem.

4. **XP-complete problems**: The class of XP-complete problems sits on top of the W hierarchy. These problems are those who have an algorithm running in time $n^{f(k)}$.

There is another concept that we did not define it yet. It is related to parameterized complexity and efficiency of algorithms. This is the Exponential Time Hypothesis (ETH). In short, ETH states that the 3-SAT problem cannot be solved in subexponential time [81]. To see the relation to parameterized complexity, we just note that the $k$-clique problem is W[1]-hard and it cannot be solved in $n^{o(k)}$ unless the ETH fails [38, 46].
Part II.

Covering Problems
3. Distributed Approximation Algorithms for Dominating Set

3.1. Introduction

A set of vertices \( D \subseteq V(G) \) is a dominating set of a graph \( G \) if their closed neighborhood is the whole graph. In the dominating set problem, the input is an undirected graph \( G = (V, E) \) and an integer \( k \). The objective is to find a dominating set of size at most \( k \). The optimization version of the problem can be defined in a natural way: minimize the size of the dominating set, we call this optimization version the Minimum Dominating Set problem or in short MDS.

Dominating set problem is NP-complete even on planar graphs [69]. But it admits polynomial time approximation scheme (PTAS) in planar graphs [14]. Grohe [75] showed that even on excluded minor graphs, the problem admits a PTAS. On the other hand from parametrized complexity side, the problem is known to be W[2]-complete. We consider the dominating set problem in the Local model (see 1.1.1). It is known that we cannot approximate a minimum dominating set locally [98]. Kuhn et al. [99] showed that the dominating set problem cannot be approximated within factor \( O(|G|^{c/r^2}) \) in \( r \) communication rounds, for any constant \( c \). Therefore there is no local algorithm for the dominating set problem in general. On the other hand for general graphs, there are approximations if we allow a non-constant number of communication rounds [101, 118].

On classes of sparse graphs, there are better results. Lenzen et al. [102] provide the first constant factor approximation local algorithm in planar graphs for the dominating set problem. Later this result was improved by Wawrzyniak [157]. Czygrinow et al. [49], based on the previous results and using coloring techniques and special clustering provide a \((1 + \epsilon)\)-approximation algorithm in \( O(\log^* |G|) \) rounds for planar graphs. We will discuss all aforementioned algorithms in more details later. On the negative side Hilke et al. [80] showed that there is no local approximation algorithm for the dominating set problem with approximation factor \( 7 - \epsilon \) for any positive \( \epsilon \) in planar graphs.

Also, Lenzen and Wattenhofer [103] provide a constant factor approximation for graphs of bounded arboricity in randomized time \( O(\log |G|) \) and a deterministic \( O(\log \Delta) \)-approximation algorithm in time \( O(\log \Delta) \) where \( \Delta \) is the maximum degree of the graph.

We extend some of the previous results in this chapter. First, we show that the very first algorithm of Lenzen et al [102] can be generalized to graphs of bounded genus. The main difficulty in this part was the new analysis. In the classical model for extending results on planar graphs, often, it suffices to find a good non-contractible cycles. But this is not the case for local algorithm. Because these non-contractible cycles can be costly and to identify them, we may need a non-constant number of communication rounds. Hence we use a novel technique, namely using properties of locally embeddable graphs to overcome
that problem. We also improve the approximation factor of our algorithm in the cost of increasing the number of communication rounds proportional to the genus. Furthermore in the same paper [4] we observed that it is possible to make the problem first order definable. In particular, we prove the following theorems.

**Theorem 3.3.1.** Let $\mathcal{C}$ be a class of graphs of genus at most $g$ (non-orientable genus or orientable genus). There is a local algorithm that computes an $O(g^2)$-approximation for the dominating set in $O(g)$ communication rounds.

We improved the approximation factor by preprocessing and we show:

**Theorem 3.4.1.** There is a local algorithm that provides a $O(g)$ MDS approximation for graphs of genus at most $g$, and requires $12g + O(1)$ communication rounds.

For every positive real number $\delta$, we provide a $(1 + \delta)$-approximation algorithm for the dominating set problem that runs in roughly speaking $O(\log^* |G| + t)$ rounds in the class of graphs excluding $K_t$ as a minor. More precisely, we assume there is an algorithm that provides a constant factor approximation to the dominating set problem on these graphs in $t$ rounds, e.g. the randomized algorithm provided in [102]. We use that algorithm as a black box and show that it is possible to improve the approximation ratio to $1 + \delta$ for every $\delta > 0$ by using only $O(\log^* |G|)$ extra rounds. This improvement in particular works on graphs of bounded genus and we provide an algorithm with almost tight approximation factor. The general machinery of above mentioned approach is based on techniques used in [49]. In particular, we prove the following theorem.

**Theorem 3.5.1.** Let $H$ be a fixed graph and $0 < \delta < 1$ and $c \in \mathbb{R}$. Let $\mathcal{C}$ be a class of $H$-minor free graphs. Let $A$ be an algorithm that provides $c$-approximation local algorithm for the dominating set problem on the class $\mathcal{C}$ in $t$ rounds. There is a $(1 + \delta)$-approximation local algorithm for the dominating set problem in the class $\mathcal{C}$ in $O(t + \log^* |G| \cdot |H| \log |H| \cdot \log \frac{|H|}{\delta})$ rounds.

At the end of this chapter, we provide a first distributed constant factor approximation algorithm for the connected dominating set problem in the class of graphs of forbidden minors. This algorithm, to the best of our knowledge, is the first constant factor approximation for the connected dominating set problem in the class of planar and bounded genus graphs in a constant number of rounds. There were folklore constant factor local approximations for these particular graphs but not with a constant number of rounds. We prove the following theorem in that section.

**Theorem 3.6.1.** Let $t$ be an integer and $\mathcal{C}$ be a class of $H$-minor free graphs. Let $A$ be a distributed algorithm that for every graph $G \in \mathcal{C}$ provides an $r$-dominating set $D$ of $G$ in $t$ rounds. Suppose $D$ is $c$-approximation for the minimum $r$-dominating set of $G$. There is a linear function $f : \mathbb{N} \rightarrow \mathbb{N}$ and a distributed $O(crf(|H|))$-approximation local algorithm $A$ for the connected $r$-dominating set that runs in $O(|H| + r + t)$ rounds.

### 3.2. Definitions and General Lemmas

A *star* is a connected graph with at most one vertex of degree greater than one, such a vertex is called the *center* of the star. We choose an arbitrary vertex as a the center of a single
3. Distributed Approximation Algorithms for Dominating Set

A graph $H$ is a depth-$1$ minor of $G$ if it is possible to obtain a $H$ from a subgraph of $G$ by only applying star contractions, that is, contracting disjoint stars to a single vertex.

We write $K_{3,t}$ for the complete bipartite graph with partitions of size $3$ and $t$, respectively. The edge density of a graph $G$ is defined as $\epsilon(G) = |E(G)|/|V(G)|$.

**Definition 3.2.1.** A graph $G$ is a locally embeddable graph if it excludes $K_{3,t}$ as a depth-$1$ minor for some $t \geq 3$ and if the edge density of every depth-$1$ minor $H$ of $G$ is at most $c$ for some constant $c$.

**Dominating Set.**

Let $G$ be a graph. A set $M \subseteq V(G)$ dominates $G$ if all vertices of $G$ lie either in $M$ or are adjacent to a vertex of $D$, that is, if $N[M] = V(G)$. A minimum dominating set $M$ is a dominating set of minimum cardinality (among all dominating sets). The size of a minimum dominating set of $G$ is denoted by $\gamma(G)$.

For a weighted graph $G$, we define an edge weight function as $w : E(G) \to \mathbb{N}$. For a subgraph $S \subseteq G$, we write $W(S)$ for $\sum_{e \in E(S)} w(e)$, and call it the total edge weight of $S$. In a weighted graph we contract an edge $\{u,v\}$ by identifying its two ends, creating a new vertex $w$, but keeping all edges (except for parallel edges and loops). Additionally if $\{u,x\}, \{v,y\} \in E(G)$, we set the edge weight of $\{uv,xy\}$ to $w(uv,xy) := w(u,v) + w(x,y)$. The degeneracy of a graph $G$ is the least number $d$ for which every induced subgraph of $G$ has a vertex of degree at most $d$.

**Bounded Genus Graphs.**

The (orientable, respectively non-orientable) genus of a graph is the minimal number $\ell$ such that the graph can be embedded on an (orientable, respectively non-orientable) surface of genus $\ell$. We write $g(G)$ for the orientable genus of $G$ and $\tilde{g}(G)$ for the non-orientable genus of $G$. Every connected planar graph has orientable genus $0$ and non-orientable genus $1$. In general, for connected graph $G$, we have $\tilde{g}(G) \leq 2g(G) + 1$. On the other hand, there is no bound for $g(G)$ in terms of $\tilde{g}(G)$. As all our results apply to both variants, for ease of presentation, and as usual in the literature, we will not mention them explicitly in the following. We do not make explicit use of any topological arguments and hence refer to [119] for more background on graphs on surfaces. In this chapter will use the following facts and lemmas about bounded genus graphs.

Graphs of genus $g$ are closed under taking subgraphs and edge contraction.

**Lemma 3.2.1.** If $H \preceq G$, then $g(H) \leq g(G)$ and $\tilde{g}(H) \leq \tilde{g}(G)$.

**Proof.** Suppose $G$ is embedded on the minimum surface $S$. Contracting any edge of $G$ does not essentially change the embedding of $G$ on $S$ therefore, the genus of $G$ can only decrease and the rest of lemma follows.

One of the arguments we will use is based on the fact that bounded genus graphs exclude large bipartite graphs as minors (and in particular as depth-1 minors). The lemma follows immediately from Lemma 3.2.1 and from the fact that $g(K_{m,n}) = \left\lceil \frac{(m-2)(n-2)}{4} \right\rceil$ and $\tilde{g}(K_{m,n}) = \left\lfloor \frac{(m-2)(n-2)}{2} \right\rfloor$ (see e.g. Theorem 4.4.7 in [119]).
3. Distributed Approximation Algorithms for Dominating Set

Lemma 3.2.2. If \( g(G) = g \), then \( G \) excludes \( K_{4g+3,3} \) as a minor and if \( \bar{g}(G) = \bar{g} \), then \( G \) excludes \( K_{2\bar{g}+3,3} \) as a minor.

Graphs of bounded genus do not contain many disjoint copies of minor models of \( K_{3,3} \): this is a simple consequence of the fact that the orientable genus of a connected graph is equal to the sum of the genera of its blocks (maximal connected subgraphs without a cut-vertex) and a similar statement holds for the non-orientable genus, see Theorem 4.4.2 and Theorem 4.4.3 in [119].

Lemma 3.2.3. \( G \) contains at most \( \max\{g(G), 2\bar{g}(G)\} \) disjoint copies of minor models of \( K_{3,3} \).

Lemma 3.2.4. We have \( |E(G)| \leq 3 \cdot |V(G)| + 6g(G) - 6 \) and \( |E(G)| \leq 3 \cdot |V(G)| + 3\bar{g}(G) - 3 \)

Proof. We only prove the first inequality; the second can be proved analogously. Suppose \( G \) is embedded in an orientable surface of genus \( g = g(\Pi) = g(G) \), with embedding \( \Pi \). Denote the number of faces of \( G \) with respect to \( \Pi \) by \( f \), its number of edges by \( e \) and its number of vertices by \( v \). Every face in \( \Pi \) has at least 3 edges and each edge appears in at most 2 faces, so \( 3f \leq 2e \), and hence \( f \leq 2/3e \). By the Euler formula (see [119]) we have:

\[
g = 1 - 1/2\chi(\Pi) = 1 - 1/2(v - e + f).
\]

Hence

\[
2g = -v + e - f + 2 \geq e/3 - v + 2,
\]

which implies \( e \leq 3v + 6g - 6 \).

The degeneracy of graphs of bounded genus is small.

Lemma 3.2.5. Let \( G \) be a class of graphs of genus at most \( g \). Then the degeneracy of every graph \( G \in G \) is in \( O(\sqrt{g}) \).

Proof. We prove the lemma for graphs with orientable genus \( g \); an analogous argument works for graphs of non-orientable genus \( g \). Let \( G \in G \) with genus at most \( g \), and suppose the degeneracy of \( G \) is \( c \). We prove that \( c \in O(\sqrt{g}) \). Let us denote by \( v, e \) the number of vertices and edges of \( G \), respectively. By Lemma 3.2.4 we have \( e \leq 3 \cdot v + 6g - 6 \). On the other hand, by definition of the degeneracy, every vertex in \( G \) has degree at least \( c \), so \( \frac{v}{c} \leq 3v + 6g - 6 \Rightarrow c \leq \frac{12g-12}{v} + 6 \) (1). To find the maximum value of \( c \) for a fixed genus, we must minimise \( v \). A complete graph on \( v \) vertices has genus at most \( v^2/12 \) [119], therefore by plugging it into (1), we obtain that \( c \leq \sqrt{12g} + 6 \).

We extend some known results to \( H \)-minor free graphs. To do so we need some definitions.

Let \( G \) be a graph and let \( F \) be a family of forests of \( G \). \( F \) is a forest cover of \( G \), if for every edge \( e \in E(G) \) there is a forest \( F \in F \) such that \( e \in E(F) \). The forest cover of \( G \) which has the minimum size is the minimum forest cover of \( G \) and its size is the arboricity of \( G \) and we denote it with \( \text{arbo}(G) \). For a graph \( G \) we define minor arboricity of \( G \) to be the \( \max_{H \subset G} \text{arbo}(H) \).
Excluded Minors

Let $H$ be a fixed graph. A graph class $C$ is called $H$-minor free graph class, if for every $G \in C$ there is no minor of $H$ in $G$. Mader in [112] proved that every graph class which excludes a fixed graph $H$ as minor, has edge density bounded above by $O(|H| \log |H|)$. Later almost tight lower bound of $\Omega(|H| \log |H|)$ was obtained by Kostochka [96] and Thomasson [152] showed that Mader’s upper bound is essentially tight, and he also showed that there are some graphs which they do not have $H$ as a minor but they do have edge density $O(|H| \log |H|)$. From these informations we have the following lemma.

Lemma 3.2.6. Let $H$ be a fixed graph and let $C$ be a class of $H$-minor free graphs. Then there is a constant integer $a_H$ such that minor arboricity of every graph $G \in C$ is bounded above by $a_H$.

Proof. By Mader’s theorem [112] there is a constant $c$ such that every graph $G \in C$ has an edge density bounded above by $d := c |H| \log |H|$. Every graph $G \in C$ has edge density at most $c$ but as the class of $H$-minor free graphs is closed under taking minor (by definition), the edge density of every minor $G'$ of $G$ is also bounded by $c$, but then by Nash Williams theorem [124] the arboricity of $G$ is at most $[c]$. We set $a_H := [c]$.  

3.3. The Local MDS Approximation

What we prove? Ideas and Results:

In this section, we provide a constant factor approximation local algorithm for dominating set problem in graphs of bounded genus. More specifically, we prove the following theorem.

Theorem 3.3.1. Let $C$ be a class of graphs of genus at most $g$ (non-orientable genus or orientable genus). There is a local algorithm that computes an $O(g)$-approximation for the dominating set in $O(g)$ communication rounds.

Before providing a formal proof of the above theorem we give the outline of the proof.

3.3.1. Local MDS Approximation: Ideas and Proof Sketch

We use a very similar algorithm to the one provided in [102]. Although the algorithm looks very similar the heart of analysis is our major contribution in this part. The algorithm is straightforward and in a nutshell, it is as follows.

1. Find vertices of the graph that their neighborhood cannot be covered by few vertices. Put them in the dominating set $D$.

2. Every vertex which is not dominated yet chooses a neighbor with a largest residual degree as its dominator.

The algorithm clearly provides a dominating set. With a simple counting argument we can show that the set produced in the first part, is not much bigger than the size of the minimum dominating set. The main and the most difficult part is to show that in the second part we do not pick many vertices.
We know that our graph is *locally embeddable*. It means that its depth-1 minor edge density is small and it does not contain a depth-1 model of $K_{3,t}$ for some $t \in O(g)$.

Having this knowledge in hand, we can decompose the residual graph into vertex disjoint parts. Each part is a star that its center belongs to the minimum dominating set. Let $\{u,v\}$ be an edge and suppose $u$ dominates $v$ by choice of our algorithm. We say the edge $\{u,v\}$ is a domination edge. We show that the number of domination edges is not too big.

Using the locally embeddable graphs, it is possible to show that there are not many edges (in particular domination edges) between two stars as otherwise, we can show that the graph has a $K_{3,t}$ model. To show the total number of (domination) edges between stars is relatively small, we construct an auxiliary graph that we call it *star contraction graph* of $G$. If we contract each star to a single vertex then star contraction graph has a bounded edge density and as the center of each star was in the optimum dominating set, then the total number of edges in the star contraction graph is small, in fact, it is in $O(|M|)$. These two together proves that the total number of edges between stars is small. Consequently, the number of domination edges between different stars is small. Thus there are few vertices whose dominators are not in their own stars.

It remains to show that there are not many dominator vertices in each star. A vertex chosen in the algorithm had a high degree. A vertex of high degree cannot have many neighbors within one star. Otherwise, we can find a $K_{3,t}$. More generally we can prove there are few vertices of high degree ($O(|M|)$). In fact, the degree of every other vertex in any star is bounded by $O(t)$. In particular for a vertex $v$ in star $S_i$, the degree of its dominator ($u$) is small. But the degree of $u$ is at least as big as the center vertex of $S_i$ because the algorithm chooses a vertex of maximum degree. Hence, as the number of vertices in the star $S_i$ is small, even if the algorithm choose all vertices in the star $S_i$, it cannot increase the approximation factor a lot. Above arguments will complete the proof of correctness of the algorithm. Detailed technical proofs will appear in the next sections.
3. Distributed Approximation Algorithms for Dominating Set

3.3.2. MDS Local Approximation Algorithm

First, we recall the algorithm provided in [102]. Here we used dom function to denote dominator vertices.

Algorithm 1.

Minimum Dominating Set Approximation Algorithm for Planar Graphs

Input: Planar graph $G$

(* Phase 1 *)

$D \leftarrow \emptyset$

for $v \in V$ (in parallel) do

if there does not exist a set $A \subseteq V(G) \setminus \{v\}$ such that $N(v) \subseteq N[A]$ and $|A| \leq 6$ then

$D \leftarrow D \cup \{v\}$

end if

end for

(* Phase 2 *)

$D' \leftarrow \emptyset$

for $v \in V$ (in parallel) do

$d_{G-D}(v) \leftarrow |N[v] \setminus N[D]|$

if $v \in V \setminus N[D]$ then

$\Delta_{G-D}(v) \leftarrow \max_{w \in N[v]} d_{G-D}(w)$

choose any $\text{dom}(v)$ from $N[v]$ with $d_{G-D}(\text{dom}(v)) = \Delta_{G-D}(v)$

$D' \leftarrow D' \cup \{\text{dom}(v)\}$

end if

end for

return $D \cup D'$

We make a little change in the above algorithm. We ask for an input $c$, the edge density of the graph and we replace the number 6 with $2c$. We will show that the modified algorithm computes a constant factor approximation of the dominating set problem in graphs of bounded genus. Number of communication rounds is clearly in $O(1)$. Later we determine the constant $c$ exactly.
3. Distributed Approximation Algorithms for Dominating Set

3.3.3. Analysis

First we show that the set $D$ computed in the Phase 1 is small. This can be obtained by straightforward generalization of Lemma 6.3 in [102].

**Lemma 3.3.1.** Let $G$ be a graph and let $M$ be a minimum dominating set of $G$. Assume that for some constant $c$ all depth-1 minors $H$ of $G$ satisfy $\epsilon(H) \leq c$. Let

$$D := \{ v \in V(G) : \text{there is no set } A \subseteq V(G) \setminus \{v\} \text{ such that } N(v) \subseteq N[A] \text{ and } |A| \leq 2c \}.$$  

Then $|D| \leq (c + 1) \cdot |M|$.

**Proof.** Let $H$ be the graph with $V(H) = M \cup N[D \setminus M]$ and where $E(H)$ is a minimal subset of $E(G[V(H)])$ such that all edges with at least one endpoint in $D \setminus M$ are contained in $E(H)$ and such that $M$ is a dominating set in $H$. Therefore every vertex $v \in V(H) \setminus (D \cup M)$ has exactly one neighbour $m \in M$, no two vertices of $V(H) \setminus (M \cup D)$ are adjacent, and no two vertices of $M$ are adjacent.

We construct a depth-1 minor $\tilde{H}$ of $H$ by contracting the star subgraphs $G_m$ induced by $N_H[m] \setminus D$ for $m \in M \setminus D$ to a single vertex $v_m$. Let $w \in D \setminus M$. As $N_G(w)$ cannot be covered by less than $(2c + 1)$ elements from $V(G) \setminus \{w\}$ (by definition of $D$), $w$ also has at least $(2c + 1)$ neighbours in $\tilde{H}$. On the other hand, $\tilde{H}$ has at most $c \cdot |V(\tilde{H})|$ edges, and also the subgraph $H[D \setminus M]$ has at most $c \cdot |D \setminus M|$ edges (by assumption on $\epsilon(H)$).

Hence

$$(2c + 1) \cdot |D \setminus M| - c \cdot |D \setminus M| \leq \sum_{w \in D \setminus M} d_{\tilde{H}}(w) - |E(\tilde{H}[D \setminus M])| \leq |E(\tilde{H})| \leq c \cdot |V(\tilde{H})| = c \cdot (|D \setminus M| + |M|),$$

and hence $|D \setminus M| \leq c \cdot |M|$, which implies the claim. $\square$

In the rest of the proof we assume that depth-1 minor edge density of the input graph is bounded by a constant $c$. We also assume that for some $t \geq 3$, $G$ excludes $K_{t,3}$ as depth-1 minor. These assumptions are actually direct result of the fact that the input graph is locally embeddable.

We use $M$ for a minimum dominating set and $D$ as the set provided in Lemma 3.3.1.

Let us write $R$ for the set $V(G) \setminus N[D]$ of the remaining vertices which are not dominated by $D$. The algorithm defines a dominator function $\text{dom} : R \rightarrow N[R] \subseteq V(G) \setminus D$. The set $D'$ computed by the algorithm is the image $\text{dom}(R)$, which is a dominating set of vertices in $R$. As $R$ contains the vertices which are not dominated by $D$, $D' \cup D$ is a dominating set of $G$. So the algorithm computes a dominating set of $G$. Our aim is to bound $|\text{dom}(R)|$. 

29
We define an ordering $\subseteq$ be an ordering $m_1, \ldots, m_{|M|}$ of vertices in $M$ such that vertices of $M \cap D$ appearing first in $\subseteq$. We construct an auxiliary graph that we call it star graph as follows.

Let $E' = \emptyset$. We add edges to $E'$ to obtain a star graph. For each $m_i$ with respect to $\subseteq$ add all edges $\{m_i, v\}$ where $v \in V(G) - (M \cup N[\bigcup_{j<i} m_j]$). The star graph $S = (V(G), E')$ is obtained from a set of vertex disjoint stars $S_1, \ldots, S_{|M|}$ where center of $S_i$ is $m_i$.

Next, in $G$, we contract each subgraph $S_i \in S$ to obtain the star contraction graph $H$. The Figure 3.1 illustrates this construction. In the figure, solid lines represent edges from $E$,
lines from $E(G) \setminus E'$ are dashed. We want to count the endpoints of directed edges, which represent the dominator function $\text{dom}$.

![Figure 3.1.: The graphs $G_m$. Solid lines represent edges from $E'$, directed edges represent the dominator function $\text{dom}$.](image)

In the following, we call a directed edge which represents the function $\text{dom}$ a dom-edge. We did not draw $\text{dom}$-edges that either start or end in $M$. When counting $|\text{dom}(R)|$, we may simply add a term $2|M|$ to estimate the number of endpoints of those edges. We also did not draw a dom-edge starting in $G_{m_1}$. In the figure, we assume that the vertex $m_1$ belongs to $M \cap D$. Hence every vertex $v$ from $N[m_1]$ is dominated by a vertex from $D$ and the function is thus not defined on $v$.

$H$ has $|M|$ vertices and by our assumption on the density of depth-1 minors of $G$, it has at most $c|M|$ edges.

We analyze as follows. We distinguish between two types of dom-edges, namely those which go from one star to another star and those which start and end in the same star. By the star contraction, all edges which go from one star to another star are represented by a single edge in $H$. We show in Lemma 3.3.2 that each edge in $H$ does not represent many such dom-edges with distinct endpoints. As $H$ has at most $c|M|$ edges, we will end up with a number of such edges that is linear in $|M|$. On the other hand, all edges which start and end in the same star completely disappear in $H$. In Lemma 3.3.5 we show that these star contractions “absorb” only few such edges with distinct endpoints.

We first show that an edge in $H$ represents only few dom-edges with distinct endpoints. For each $m \in M \setminus D$, we fix a set $C_m \subseteq V(G) \setminus \{m\}$ of size at most $2c$ which dominates $N_{E'}(m)$, note that existence of $C_m$ follows from the definition of the set $D$. Recall that we assume that $G$ excludes $K_{t,3}$ as depth-1 minor.

**Lemma 3.3.2.** Let $1 \leq i < j \leq |M|$. Let $N_i := N_{E'}(m_i)$ and $N_j := N_{E'}(m_j)$.

1. If $m_j \notin D$, then $|\{u \in N_j : \text{there is } v \in N_i \text{ with } \{u, v\} \in E(G)\}| \leq 2ct$.
2. If $m_i \notin D$ (and hence $m_j \notin D$), then $|\{u \in N_i : \text{there is } v \in N_j \text{ with } \{u, v\} \in E(G)\}| \leq 4ct$. 

30
3. Distributed Approximation Algorithms for Dominating Set

Proof. By definition of $E'$, it holds that $m_i \notin C_{m_j}$. Let $c \in C_{m_j}$ be arbitrary. Then there are at most $t-1$ distinct vertices $u_1, \ldots, u_{t-1} \in (N_j \cap N(c))$ such that there are $v_1, \ldots, v_{t-1} \in N_i$ (possibly not distinct) with $\{u_k, v_k\} \in E(G)$ for all $k$, $1 \leq k \leq t-1$. Otherwise, we can contract the star with center $m_i$ and branch vertices $N(m_i) \setminus \{c\}$ and thereby find $K_{t,3}$ as depth-1 minor, a contradiction. See Figure 3.2 for an illustration in the case of an excluded $K_{3,3}$. Possibly, $c \in N_j$ and it is connected to a vertex of $N_i$, hence we have at most $t$ vertices in $N_j \cap N[c]$ with a connection to $N_i$. As $|C_{m_j}| \leq 2c$, we conclude the first item.

Regarding the second item, let $c \in C_{m_i}$ be arbitrary. If $c \neq m_j$, we conclude just as above, that there are at most $t-1$ distinct vertices $u_1, \ldots, u_{t-1} \in (N_i \cap N(c))$ such that there are $v_1, \ldots, v_{t-1} \in N_j$ (possibly not distinct) with $\{u_k, v_k\} \in E(G)$ for all $k$, $1 \leq k \leq t-1$ and hence at most $t$ vertices in $N_i \cap N[c]$ with a connection to $N_j$. Now assume $c = m_j$. Let $c' \in C_{m_j}$. There are at most $t-1$ distinct vertices $u_1, \ldots, u_{t-1} \in (N_i \cap N_E(m_j))$ such that there are vertices $v_1, \ldots, v_{t-1} \in N_j \cap N(c)$ (possibly not distinct) with $\{u_k, v_k\} \in E(G)$ for all $k$, $1 \leq k \leq t-1$. Again, considering the possibility that $c' \in N_i$, there are at most $t$ vertices in $N_i \cap N_E(m_j)$ with a connection to $N_j \cap N(c)$. As $|C_{m_j}| \leq 2c$, we conclude that in total there are at most $2ct$ vertices in $N_i \cap N_E(m_j)$ with a connection to $N_j$. In total, there are hence at most $(2c-1)t + 2ct \leq 4ct$ vertices of the described form. □

![Figure 3.2:](image)

Figure 3.2: Visualisation of the proof of Lemma 3.3.2 in the case of excluded $K_{3,3}$

We write $Y$ for the set of all vertices $\{u \in N_{E'}(m_i) : m_i \notin D$ and there is $v \in N_{E'}(m_j), j \neq i \text{ and } \{u, v\} \in E(G)\}$.

**Corollary 3.3.1.** $|Y| \leq 6c^2t |M|$.

**Proof.** Each of the $c|M|$ many edges in $H$ represents edges between $N_i$ and $N_j$, where $N_i$ and $N_j$ are defined as above. By the previous lemma, if $i < j$, there are at most $2ct$ vertices in $N_i \cap Y$ and at most $4ct$ vertices in $N_j \cap Y$, hence in total, each edge accounts for at most $6ct$ vertices in $Y$. □

We continue to count the edges which are inside the stars. First, we show that every vertex has small degree inside its own star.

**Lemma 3.3.3.** Let $m \in M \setminus D$ and let $v \in N_{E'}(m) \setminus C_m$. Then

$$|\{u \in N_{E'}(m) : \{u, v\} \in E(G)\}| \leq 2c(t-1).$$

**Proof.** Let $c \in C_m$. By the same argument as in Lemma 3.3.2, there are at most $t-1$ distinct vertices $u_1, \ldots, u_{t-1} \in (N_{E'}(m) \cap N(c))$ such that $\{u_k, v\} \in E(G)$ for all $k$, $1 \leq k \leq t-1$. □
3. Distributed Approximation Algorithms for Dominating Set

Let \( C := \bigcup_{m \in M \setminus D} C_m \). There are only few vertices which are highly connected to \( M \cup C \).
Let \( Z := \{ u \in N_{E'}(M \setminus D) : |N(u) \cap (M \cup C)| > 4c \} \).

**Lemma 3.3.4.**
\[
|Z| < |M \cup C|.
\]

**Proof.** Assume that \( |Z| > |M \cup C| \). Then the subgraph induced by \( Z \cup M \cup C \) has more than \( \frac{1}{4}4c|Z| \) edges and \( |Z \cup M \cup C| \) vertices. Hence its edge density is larger than \( 2c|Z|/(|Z \cup M \cup C|) > 2c|Z|/(2|Z|) = c \), contradicting our assumption on the edge density of depth-1 minors of \( G \) (which includes its subgraphs).

Finally, we consider the image of the \( \text{dom} \)-function inside the stars.

**Lemma 3.3.5.**
\[
\left| \bigcup_{m \in M \setminus D} \{ u \in N_{E'}(m) : \text{dom}(u) \in (N_{E'}(m) \setminus (Y \cup Z)) \} \right| 
\leq (2(t - 1) + 4)c|M|.
\]

**Proof.** Fix some \( m \in M \setminus D \) and some \( u \in N_{E'}(m) \) with \( \text{dom}(u) \in N_{E'}(m) \setminus (Y \cup Z) \). Because \( \text{dom}(u) \notin Y \), \( \text{dom}(u) \) is not connected to a vertex of a different star, except possibly for vertices from \( M \). Because \( \text{dom}(u) \notin Z \), it is however connected to at most \( 4c \) vertices from \( M \cup C \). Hence it is connected to at most \( 4c \) vertices from different stars. By Lemma 3.3.3, \( \text{dom}(u) \) is connected to at most \( 2c(t - 1) \) vertices from the same star. Hence the degree of \( \text{dom}(u) \) is at most \( 4c + 2c(t - 1) \). Because \( u \) prefers to choose \( \text{dom}(u) \in N_{E'}(m) \) over \( m \) as its dominator, we conclude that \( m \) has at most \( 4c + 2c(t - 1) \) \( E' \)-neighbours. Hence, in total there can be at most \( (2(t - 1) + 4)c|M| \) such vertices.

We are now ready to put together the numbers.

**Lemma 3.3.6.** If all depth-1 minors \( H \) of \( G \) have edge density at most \( c \) and \( G \) excludes \( K_{1,3} \) as depth-1 minor, then the modified algorithm computes a \( 6c^2t + (2t + 5)c + 4 \) approximation for the minimum dominating set problem on \( G \).

**Proof.** The set \( D \) has size at most \( (c + 1)|M| \) according to Lemma 3.3.1. Since \( M \) is a dominating set also with respect to the edges \( E' \), it suffices to determine:
\[
|\{ \text{dom}(u) : u \in (N_{E'}(M \setminus D) \setminus N[D]) \}|.
\]

According to Corollary Corollary 3.3.1, the set \( Y = \{ u \in N_{E'}(m_i) : \text{there is } v \in N_{E'}(m_j), i \neq j \text{ and } \{ u, v \} \in E(G) \} \) has size at most \( 6c^2t|M| \). In particular, there are at most so many vertices \( \text{dom}(u) \in N_{E'}(m_i) \) with \( u \in N_{E'}(m_j) \) for \( i \neq j \). Clearly, \( |\text{dom}(R) \cap M| \leq |M| \) and \( |\text{dom}(M)| \leq |M| \). Together, this bounds the number of endpoints of \( \text{dom} \)-edges that go from one star to another star. According to Lemma 3.3.4, there are only few vertices which are highly connected to \( M \cup C \), that is, the set \( Z = \{ u \in N_{E'}(M \setminus D) : |N(u) \cap (M \cup C)| > 4c \} \).
3. Distributed Approximation Algorithms for Dominating Set

satisfies $|Z| < |M \cup C|$. We have $|C| \leq 2c|M|$, as each $C_m$ has size at most $2c$. It remains to count the image of $\text{dom}$ inside the stars which do not point to $Y$ or $Z$. According to Lemma 3.3.5, this image has size at most $(2(t - 1) + 4)c|M|$. In total, we hence find a set of size

$$(c + 1)|M| + 6c^2t|M| + 2|M| + (2c + 1)|M| + (2(t - 1) + 4)c|M| \leq (6c^2t + (2t + 5)c + 4)|M|.$$  

\square

Our theorem for bounded genus graphs is now a corollary of Lemma 3.2.2 and Lemma 3.3.6.

**Theorem 3.3.1.** Let $C$ be a class of graphs of genus at most $g$ (non-orientable genus or orientable genus). The modified algorithm computes an $O(g^2)$-approximation for the dominating set in $O(g)$ communication rounds.

**Proof.** Number of communication rounds in the modified version of the Algorithm 1 is in $O(1)$. Also by Lemma 3.2.2 and Lemma 3.3.6 the algorithm computes $O(c^2g)$ approximation. On the other hand by Lemma 3.2.2, the asymptotic value of $c$ is 3, therefore the asymptotic factor of the approximation algorithm is in $O(g)$. Also we know that the genus of complete graph on $v$ vertices is in $O(v^2)$, therefore the value of $c$ is in $O(\sqrt{g})$ in all cases, so the algorithm provides $O(g^2)$-approximation for MDS. \square

For the special case of planar graphs, our analysis shows that the algorithm computes a 199-approximation. This is not much worse than Lenzen et al.’s original analysis (130), however, off by a factor of almost 4 from Wawrzyniak’s [157] improved analysis (52).

### 3.4. $O(g)$ Approximation Factor in $O(g)$ Communication Rounds

We now show that the approximation algorithm provided in the previous section can be improved such that approximation factor has linear dependency to the genus. This can be done in the cost of $O(g)$ communication rounds.

Given a graph $G$ and a vertex $v \in V(G)$. Let $K = \{K_1, \ldots, K_j\}$ denote the set of minimal subgraphs of $G$ such that for all $1 \leq i \leq j$, $K_{3,3}$ is a depth-1 minor of $K_i$. Let $K_h \in K$ be the one with lexicographically smallest identifiers in $K$, we say $K_h$ is the $v$-canonical subgraph of $G$ and we denote it by $K_v$. If $K = \emptyset$ we set $K_v := \emptyset$.

**Lemma 3.4.1.** Given a graph $G$ and a vertex $v \in V(G)$. The $v$-canonical subgraph of $G$ ($K_v$) can be computed locally in at most 6 communication rounds. Furthermore, $K_v$ has at most 24 vertices.

**Proof.** The proof is constructive. As $K_{3,3}$ has diameter 2, every minimal subgraph of $G$ containing $K_{3,3}$ as a depth-1 minor has diameter at most 6 (every edge may have to be replaced by a path of length 3). Therefore, it suffices to consider the subgraph $H = G[N^6(v)]$ and find the lexicographically minimal subgraph which contains $K_{3,3}$ as depth-1 minor in $H$ which includes $v$ as a vertex. If this is the case, we output it as $K_v$; otherwise we output the empty set. Furthermore, $K_{3,3}$ has 9 edges and hence a minimal subgraph containing it as depth-1 minor has at most 24 vertices (again, every edge is subdivided at most twice and $2 \cdot 9 + 6 = 24$).  

\square
To improve the approximation factor, we propose the following modified algorithm, see Algorithm 2.

**Algorithm 2.**

**Dominating Set Approximation Algorithm for Graphs of Genus \( \leq g \)**

Input: Graph \( G \) of genus at most \( g \)

**Run Phase 1 of Algorithm 1**

(* Preprocessing Phase *)

for \( v \in V - D \) (in parallel) do

compute \( K_v \) in \( G - D \) (see Lemma 3.4.1)

for \( i = 1 .. g \) do

for \( v \in V - D \) (in parallel) do

if \( K_v \neq \emptyset \) then

chosen := true

for all \( u \in N^{12}(v) \)

if \( K_u \cap K_v \neq \emptyset \) and \( u < v \) then chosen := false

if (chosen = true) then \( D := D \cup V(K_v) \)

**Run Phase 2 of Algorithm 1**

**Theorem 3.4.1.** Algorithm 2 provides a \( O(g) \) MDS approximation for graphs of genus at most \( g \), and requires \( 12g + O(1) \) communication rounds.

**Proof.** The resulting vertex set is clearly a legal dominating set. Moreover, as Phase 1 is unchanged, we do not have to recalculate \( D \).

In the preprocessing phase, if for two vertices \( u \neq v \) we choose both \( K_u, K_v \), then they must be disjoint. Since the diameter of any depth-1 minor of \( K_{3,3} \) is at most 6, if two such canonical subgraphs intersect, the distance between \( u, v \) can be at most 12. On the other hand, by Lemma 3.2.3, there are at most \( g \) disjoint such models. So in the preprocessing phase, we can remove at most \( g \) disjoint subgraphs \( K_v \) (and add their vertices to the dominating set) and thereby select at most \( 24g \) extra vertices for the dominating set. Once the preprocessing phase is finished, the remaining graph is locally embeddable. In order to compute the size of the set in the third phase, we can use the analysis of Lemma 3.3.6 for \( t = 3 \). This together with the first phase and preprocessing phase, results in a \( O(g) \)-approximation guarantee.

To count the number of communication rounds, note that the only change happens in the second phase. In that phase, in each iteration, we need 12 communication rounds to compute the 12-neighbourhood. Therefore, the number of communication rounds is \( 12g + O(1) \). \( \square \)

This significantly improves the approximation upper bound of Theorem 3.3.1 to linear dependency to the genus at the price of \( 12g \) extra communication rounds.
3. Distributed Approximation Algorithms for Dominating Set

3.5. \((1 + \delta)\)-Approximation for dominating set in \(H\)-Minor Free Graphs

In this section we prove the following meta theorem.

**Theorem 3.5.1.** Let \(H\) be a fixed graph and let \(0 < \delta < 1\) and let \(c \in \mathbb{R}\). Let \(C\) be a class of \(H\)-minor free graphs. Let \(A\) be an algorithm which provides \(c\)-approximation for dominating set in the class \(C\) in \(t\) rounds. There is an algorithm which provides \((1 + \delta)\)-approximation for dominating set in the class \(C\) with \(O(t + \log^*|G| \cdot |H| \cdot \log |H| \cdot \log \frac{|H| \log |H|}{\delta})\) rounds.

The above meta-theorem together with our previous results gives us two main follow-ups.

1. A deterministic distributed local \((1 + \delta)\)-approximation algorithm for dominating set in constant rounds on graphs of bounded genus.

2. \((1 + \epsilon)\)-approximation algorithm for dominating set on \(H\)-minor free graphs.

The algorithm in fact relies on a slight modification of the clustering algorithm for planar graphs presented by Czygrinow et al. \[49]\.

**3.5.1. Ideas and Proof Sketch**

The algorithm works as follows.

(i) Input: a graph \(G\) with depth-1 minor arboricity at most \(a\) and a set \(D\). \(D\) should be a \(c\)-approximation for optimal dominating set of \(G\).

(ii) Take disjoint spanning stars as start of iteration. Centers of stars are vertices of the dominating set.

(iii) Contract each star and in the remaining graph find a good clustering. A clustering is good if the total number of edges between clusters is small.

(iv) Un-contract things and find a dominating set within each cluster. This we can do as clusters are of small radius. Update the dominating set and repeat Item ii.

We follow the lines of proof provided in \[49\]. For the start setting, we take the dominating set provided by an Algorithm \(A\). First, we show that in \(O(1)\) communication rounds we can find a heavy forest \(F\) in every graph of bounded arboricity. \(F\) has a big total edge weight. We color the vertices of \(F\). Then by slight modifications on known techniques, we decompose the graph into loosely coupled clusters.

We find the minimum dominating set in each cluster. As the clusters are loosely coupled they form a good approximation for the original dominating set problem. The latter is basically a technique where its origin is back to the classical model (Baker’s technique) for planar graphs. The number of iterations depends to arboricity, approximation factor \(c\) of the helper algorithm \(A\) and the \(\delta\).
3. Distributed Approximation Algorithms for Dominating Set

3.5.2. Proof Details

We need some definitions and we present them here.

Definition 3.5.1 (Pseudo-Forest [49]). A pseudo-forest is a directed graph in which every vertex has an out-degree at most 1.

For a directed graph $G$, if we ignore the edge directions, we write $\bar{G}$.

The following lemma is the generalization of the Fact 1 in [49].

Lemma 3.5.1. There is a constant $c_1$ such that for an edge weighted graph $G$ of arboricity $a$, we can find, in two communication rounds, a pseudo-forest $F$ such that $\bar{F}$ is a spanning subgraph of $G$ and $W(\bar{F}) \geq \frac{W(G)}{2a}$.

Proof. As $G$ has arboricity at most $a$, there is a forest cover $F$ such that $|F| \leq a$. So it has a forest $F_1 \in F$ such that $W(F_1) \geq W(G)/a$. We run the following algorithm. For a vertex $v$, we choose an edge $\{v,u\}$ of largest weight, and direct it from $v$ to $u$. If we happen to choose an edge $\{v,u\}$ for both vertices $u$ and $v$, we direct it from $v$ to $u$, using the larger identifier as a tie breaker. This algorithm creates a pseudo-forest $F$. $\bar{F}$ is a spanning sub-graph of $G$ and it has a total edge weight of at least half of $W(F_1)$, so $W(\bar{F}) \geq W(G)/(2a)$. Note that we found $F$ in two rounds. $\square$

We use the HeavyStar algorithm provided in [49] as a subroutine. Instead of providing the algorithm we summarize it with the following lemma.

Lemma 3.5.2 (HeavyStar). There is a local algorithm which takes a graph $G$ of minor arboricity at most $a$ as input and outputs stars of total weight at least $\Omega(\frac{W(G)}{a})$.

Proof. Let $t := 8a$. By applying the HeavyStars algorithm from [49] on the pseudo forest provided in the proof of Lemma 3.5.1, we obtain stars of weight at least $\frac{|E(G)|}{t}$ as claimed. $\square$

We customize the Clustering Algorithm in [49] in the following. In the following algorithm, note that we supposed the arboricity of every minor of the input graph is at most $a$ and $r$ is the number of iterations.

Algorithm 3.

Clustering Algorithm

$H := G, t := 8a, r := \log(\frac{1}{\epsilon})/\log(\frac{t}{t-1})$

Iterate $r$ times

(a) Call HeavyStars Lemma 3.5.2 to find vertex disjoint stars in $H$.

(b) Modify $H$ by contracting each star to a vertex and computing the weights accordingly.

Let $W$ denote the set of vertices contracted to $w$. Return $\{W | w \in V(H)\}$.

Lemma 3.5.3 (Clustering). The Clustering Algorithm takes an $0 < \epsilon < 1$ and an edge weighted graph $G$ of minor arboricity at most $a$ as input, runs in $O(\log^* |G| \cdot \log(\frac{1}{\epsilon}))a$ rounds and returns a set of clusters $C_1, \ldots, C_l$ partitioning $G$, such that, each cluster has a diameter
bounded above by $O(10 \cdot 2^r)$ (which $r$ is the number of iterations in the algorithm). Moreover, if we contract each $C_i$ to a single vertex to obtain a graph $H$, then $W(H) \leq \epsilon \cdot W(G)$.

**Proof.** Let $t := a$. Let denote by $W_i$ the weight of the graph after $i$th iteration of the algorithm. By Lemma 3.5.2 in each iteration, the algorithm contracts stars which their total edge weight is at least $W_{i-1}/t$. Hence at the end of step $i$ the weight of the graph is at most $(t-1)W_{i-1}/t$ and consequently after $i$th iteration the weight of the graph $H$ is at most $(\frac{t-1}{t})^i W(G)$. We have to find the $i$ such that $(\frac{t-1}{t})^i \leq \epsilon$ but then by reversing both side (noting that both of them are smaller than 1) and then taking logarithm of both sides we get $i \geq \log(\frac{1}{\epsilon})/\log(\frac{t-1}{t})$, but then assuming that $t > 1$ (it means supposing there are some edges in $G$), we have $\log(\frac{t-1}{t}) \geq 1/t$. So $i \leq \log(\frac{1}{\epsilon}) \cdot t \in O(\log(\frac{1}{\epsilon})a)$, hence number of iterations $r$, chosen in the algorithm, is exactly equal to minimum value of $i$ so the resulting graph has total edge weight at most $\epsilon W(G)$.

On the other hand in each iteration we use $O(\log^* |G|)$ rounds, so total number of communication rounds in the clustering algorithm is in $O(r \log^* |G|)$ or it is in $O(\log^* |G| \cdot \log(\frac{1}{\epsilon})a)$ as claimed.

Each cluster at each iteration by heavy star algorithm has diameter 10 and we may contract those clusters recursively at most $r$ times so the total diameter of each cluster is at most $10 \cdot 2^r$ as claimed. \hfill $\square$

Now we can prove the main theorem of this section. The following proof is similar to Theorem 3.4 in [49].

**Theorem 3.5.1.** Let $H$ be a fixed graph, $0 < \delta < 1$, and $c \in \mathbb{R}$. Let $C$ be a class of $H$-minor free graphs. Let $A$ be an algorithm which provides $c$-approximation for the minimum dominating set in the class $C$ in $t$ rounds. There is an algorithm which provides $(1 + \delta)$-approximation for dominating set in the class $C$ with $O(t + \log^* |G| \cdot |H| \log |H| \cdot \log |H| \cdot \log (\frac{|H|}{\log |H|}))$ rounds.

**Proof.** As $C$ is the class of $H$-minor free graphs, by Lemma 3.2.6 there is an integer $a \in O(|H| \log |H|)$ such that minor arboricity of every graph $G \in C$ is bounded above by $a$. Let $G \in C$ and suppose $OPT$ is the optimal dominating set of $G$. By Algorithm $A$, in $t$ rounds, we find an dominating set $D$ of $G$ such that $|D| \leq c \cdot |OPT|$

Let us order the vertices of $G$ arbitrarily, and suppose $d_1, \ldots, d_{|D|}$ is such an ordering. Create a $D$-partition $P_D := (V_1, \ldots, V_{|D|})$ of $V(G)$. We next contract each $V_i$ to a single vertex $v_i$ to obtain a graph $H$.

We assign edge weights to $H$, i.e., for all $e \in E(H)$, we set $w(e) := 1$. It is clear that $W(H) = |E(H)|$. Also $H$ has at most $a |D|$ edges.

Set $\epsilon := \frac{\delta}{2\alpha}$. We apply the algorithm in Lemma 3.5.3, to find clusters $C_1, \ldots, C_l$ such that the total edge weights between clusters is at most $\epsilon \cdot |E(H)|$. As $\epsilon \in \Omega(\delta/a)$, the clustering algorithm uses $O(t + \log^* |G| \cdot \log \frac{1}{\epsilon}a) = O(\log \frac{2}{\epsilon} \cdot \log \frac{2}{\epsilon} \log^* |G| + t)$ rounds.

For a cluster $C_j$, suppose $V(C_j) = \{v_{j1}, \ldots, v_{jk}\}$, and let $U_j$ be an induced subgraph of $G$ on vertices of a subgraph $X := \bigcup_{j=1, \ldots, k} V_{j}$, i.e $U_j := G[X]$.

We know that each $C_i$ had a constant diameter $r' \leq 10r$ which $r$ is the number obtained from the clustering algorithm and only depends to $\delta$, so each $U_j$ will have a diameter at most $O(r)$, hence finding an optimum dominating set $S_i$ of each $U_j$ can be done in $O(r)$ rounds.

Now take a dominating set $S := \bigcup S_i$. We claim $S$ is a $(1 + \delta)$-approximation to the $OPT$.\hfill 37
First of all, it is clear that $S$ is a dominating set of $G$. To prove the upper bound, let $S^*$ be the union of vertices of $OPT$ and a set of vertices of $D$ which for every $d \in S^*$ there is a vertex $w$ such that $\text{distance}_{G}(d, w) \leq 2$ and $w$ is not in the same cluster as $d$.

By the CLUSTERING algorithm and the above counting, we have:

$$|S^*| \leq 2 \epsilon |E(H)| + |OPT|$$

$$\leq 2 \frac{\delta}{2ac} |D| + |OPT|$$

$$\leq (1 + \delta) |OPT|.$$ 

But for each cluster $U_i$ we have that $S^* \cap U_i$ is a dominating set of $U_i$: if there is a vertex $u \in OPT$ which dominates some vertex $v \in U_i$ but $u \notin U_i$ we have a vertex $d \in D \cap S^*$ which dominates $v$.

On the other hand, we know that for each $U_i$ we have $|U_i \cap S| \leq |S^* \cap U_i|$ so $|S| \leq |S^*| \leq (1 + \delta) |OPT|$. Replacing the value of $a$ in the calculation of the running time proves the claim of the lemma.

**Corollary 3.5.1.** Let $0 < \epsilon < 1$ and $C_g$ be a class of graphs of genus at most $g$. Then there is a distributed algorithm which uses $O(g + \log^{*} |G| \cdot \sqrt{g} \cdot \log g)$ rounds and provides a $(1 + \epsilon)$-approximation for minimum dominating set problem for graphs in $C_g$.

**Proof.** This is a consequence of Theorem 3.5.1 and Theorem 3.4.1 and Lemma 3.2.5. 

### 3.6. On Connected $r$-Dominating Set Problem

Both theoretically and practically relevant problem to the dominating set problem is the connected $r$-dominating set problem. In fact, a dominating set which is a connected subgraph of the original graph. It is clear that connected dominating set exists if, and only if the graph is already connected. The problem is defined as follows.

The $r$-dominating set problem asks for a set of vertices $D$ of a graph $G$ such that for every vertex $v \in G$ there is a vertex $u \in D$ such that $\text{distance}_{G}(u, v) \leq r$.

**Definition 3.6.1** (Connected Dominating Set Problem). The input to the problem is a graph $G$ and the task is to find a $r$-dominating set $D$ such that $G[D]$ is connected. We denote the problem with connected $r$-dominating set.

In this section, we consider the problem on sparse graphs, in particular, $H$-minor free graphs. We show that one can provide a fast distributed approximation algorithm for the problem in those graphs. We also provide a constant factor approximation in a constant number of rounds for graphs of bounded genus.

Let $G$ be a graph and $D := \{d_1, \ldots, d_{|D|}\}$ a $r$-dominating set of $G$. A $D$-partition of $V(G)$ is a partition $P = (V_1, \ldots, V_{|D|})$ such that for all $i \in [|D|]$ and for every $v \in V_i$ the lexicographically shortest $(v, d_i) - \text{path}$ is lexicographically smaller than the lexicographically shortest path from $v$ to $d_j$ for all $j \in [|D|] - \{i\}$. We say $d_1, \ldots, d_{|D|}$ are centers of partition. We have the following main observation.

**Lemma 3.6.1.** Let $G$ be a graph and $D = \{d_1, \ldots, d_{|D|}\}$ an $r$-dominating set of $G$ and $P = (V_1, \ldots, V_{|D|})$ be a $D$-partition of $V(G)$. Then for all $i \in [|D|]$ and every $v \in V_i$ we have $\text{distance}_{G[V_i]}(v, d_i) \leq r$. 

38
3. Distributed Approximation Algorithms for Dominating Set

**Proof.** It is clear that if \( v \in V_i \) then there is a lexicographically shortest \((v, d_i)\) – path \( p \) in \( G \) such that \( \ell(p) \leq r \). Suppose distance\( G[V_i] (v, d_i) > r \). Then there is a vertex \( w \in p \) such that \( w \in V_j \) for some \( j \neq i \). But as \( w \in V_j \) then the shortest lexicographical \((w, d_j)\) – path \( p' \) is lexicographically smaller than \((w, d_i)\) – path but then the lexicographically smallest \((v, w)\) – path together with \( p' \) creates a lexicographically smaller walk than \( p \), so \( v \notin V_i \), a contradiction.

If we have an \( r \)-dominating set, we can find the \( D \)-Partition in \( O(r) \) rounds. This we show in the following lemma.

**Lemma 3.6.2.** Let \( r \in \mathbb{N} \) and \( G = (V, E) \) be a graph and \( D \subseteq V \) be an \( r \)-dominating set of \( G \). There is a distributed algorithm which uses \( O(r) \) rounds and finds \( D \)-Partition of \( G \).

**Proof.** Every vertex \( v \in G \) finds its \( r \) neighbourhood \( N^r[v] \) in \( r \) rounds. Then finds the shortest lexicographical paths to each of vertices in \( D \cap N^r[v] \). Among those paths, takes the lexicographically shortest one, let say it is to a vertex \( d \in D \cap N^r[v] \). Then \( v \) assigns itself to the partition associated to the vertex \( d \).

We propose the following algorithm Algorithm 4.

**Algorithm 4.** 1. Input: an \( H \)-minor free graph \( G \), an integer \( r \) and a \( r \)-dominating set \( D \) of \( G \) which is a \( c \)-approximation for optimal \( r \)-dominating set.

2. Set \( r \)-ConDomSet := \( D \).

3. Find \( D \)-Partition \( P = (V_1, \ldots, V_{|D|}) \) of \( G \).

4. For every pair of partitions \( V_i, V_j \) find the edge \( e = \{u, v\} \) such that it has one end in \( V_i \) and the other in \( V_j \) and it is lexicographically shortest such edge among all edges between \( V_i, V_j \) (for \( 1 \leq i < j \leq |D| \)). Put endpoints of \( e \) in the set \( \mathcal{E}(G) \).

5. For \( i \in [|D|] \) and for every vertex \( v \in \mathcal{E}(G) \cap V_i \) find the lexicographically shortest \((v, d_i)\) – path and put all of its vertices in \( r \)-ConDomSet.

6. Output \( r \)-ConDomSet.

We first claim the set \( r \)-ConDomSet provided in the algorithm Algorithm 4 is a connected \( r \)-dominating set of \( G \).

**Lemma 3.6.3.** If \( G \) is a connected graph then the set \( r \)-ConDomSet is a connected \( r \)-dominating set of \( G \).

**Proof.** As every vertex in \( D \) is also in \( r \)-ConDomSet it is a dominating set. We prove that the set \( r \)-ConDomSet is also connected. Suppose there are two vertices \( u, v \) such that there is no path between \( u \) and \( v \) in \( G[r \text{-ConDomSet}] \) and such that \( u, v \) have the shortest distance in \( G \) among all such pairs, suppose there is a shortest path \( p \) which connects \( u, v \) in \( G \). Suppose \( u \in V_i, v \in V_j \) for some \( i, j \in |D| \). \( V(p) = V(V_i \cup V_j) \) then \( u, v \) are connected by some path in \( G[r \text{-ConDomSet}] \). Otherwise there is a partition \( V_y \) such that \( i \neq j \neq y \) and a vertex \( w \in p \cap V_y \). Then we have distance\( G(u, w) < \) distance\( G(u, v) \) and distance\( G(v, w) < \) distance\( G(u, v) \). However, by our choice of \( u, v \) there is a path \( p_1 \) between \( v, w \) in \( G[r \text{-ConDomSet}] \) and there is a path \( p_2 \) between \( u, w \) in \( G[r \text{-ConDomSet}] \), so there is a walk with vertices in \( p_1, p_2 \) from \( u \) to \( v \) in \( G[r \text{-ConDomSet}] \) which is a contradiction to our assumption.
3. Distributed Approximation Algorithms for Dominating Set

We also show that $r$-ConDomSet is small in $H$-minor free graphs. In the following, $a_H$ is the edge density function obtained from Lemma 3.2.6.

**Lemma 3.6.4.** Let $G$ be an $H$-minor free graph. Let $OPT$ be the size of minimum $r$-dominating set of $G$. Then we have $|r$-ConDomSet$| \leq a_H \cdot c |OPT|$.

**Proof.** In Algorithm 4 every partition $V_i$ is connected by Lemma 3.6.1 and additionally it has radius at most $r$. So by Lemma 3.2.6 the number of edges in $\mathcal{E}(G)$ is at most $a_H |D|$. Also every path which we put its vertices in $r$-ConDomSet has length at most $r$ and for each edge we take at most two such paths. Hence the number of vertices in the $r$-ConDomSet is bounded above by $2(r + 1) |\mathcal{E}(G)||D| = 2(r + 1)a_H |D|$.

**Theorem 3.6.1.** Let $t$ be an integer and $\mathcal{C}$ be a class of $H$-minor free graphs. Let $A$ be a distributed algorithm that for every graph $G \in \mathcal{C}$ provides an $r$-dominating set $D$ of $G$ in $t$ rounds. Suppose $D$ is $c$-approximation for the minimum $r$-dominating set of $G$. There is a linear function $f: \mathbb{N} \to \mathbb{N}$ and a distributed $O(crf(|H|))$-approximation local algorithm $A$ for the connected $r$-dominating set that runs in $O(|H| + r + t)$ rounds.

**Proof.** By Lemmas 3.6.3 and 3.6.4 the Algorithm 4 provides $O(cra_H)$-approximation for connected $r$-dominating set in $G$. The above algorithm runs in $O(r)$ rounds. Adding $t$ rounds of the algorithm $A$ gives the claimed running time.

**Corollary 3.6.1.** There is a local algorithm which computes a constant factor approximation for connected dominating set in the graphs of bounded genus with a constant number of rounds.

**Proof.** To the Algorithm 4 we give a graph of bounded genus $G$ and an Algorithm 2 as input, and by Lemma 3.2.4 the edge density of every graph of bounded genus is bounded above by a constant so by the theorem Theorem 3.6.1, the claim follows.
3.7. Conclusion and Future Works

We provide the first constant factor approximation local algorithm of dominating set problem in graphs of bounded genus. This new result is interesting beyond the scope of dominating set problem. In fact, to the best of the author’s knowledge, this is the first local algorithm which uses properties of bounded genus graphs. In fact any other known distributed algorithm on graphs and networks, either is non-local or it is not tailored for those graphs. The main difficulty to approach this class of graphs was that unlike the classical model, we cannot find a non-contractible cycle and cut the graph through that cycle to reduce the genus. In fact, non-contractible cycles can be arbitrarily large and they do not admit the LOCAL model requirements. So obtaining an algorithm with a constant number of rounds was our main novelty.

We also improved the dependency of approximation factor to have a linear dependency to the genus of the graph. This has been done with a simple topological argument and additional preprocessing phase.

Then we showed that, in the class of excluded minor graphs, we can arbitrary well approximate the dominating set problem. In particular this yields a \((1 + \epsilon)\)-approximation in \(O(\log g \cdot \sqrt{g} \log^* |G|)\) rounds, in graphs of bounded genus.

We then provide the first constant factor approximation for connected dominating set in a constant number of rounds in graphs of bounded genus. In fact, we provide an extension of that result which gives a constant factor approximation for connected \(r\)-dominating set in the class of bounded expansion graphs. This obtained mainly by usage of lexicographically shortest path technique, which we think can be used in other problems.

There is one big open question left open. Is there a local algorithm that provides a constant factor approximation for the dominating set problem in the class of forbidden minor graphs in constant rounds? By Kuratowski’s theorem, we know that there is no \(K_{3,3}\) minor in a planar graph. In our analysis, we use the fact that a graph of genus \(g\) does not contain a minor of \(K_{3,t}\). So we extend one of the 3’s from planar graph to \(t\). Now the question is whether we can extend the other 3 to \(t\). It is known that in a logarithmic number of rounds one can provide a constant factor approximation in graph classes which exclude a minor[103].

But the following question remains open. Is it possible to provide a constant approximation in constant number of rounds for the dominating set problem in the LOCAL model when we exclude a \(K_t\) (or \(K_{t,t}\))?
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

4.1. Introduction

Any graph either has a big matching or a small vertex cover. A more general version of this duality can be seen in Menger’s theorem: take any two subsets of vertices of a graph, either there are many disjoint paths between them or there is a small set of vertices which hits all such paths. More general duality theorem is Dirac’s theorem: take a subset of vertices of the graph as principal vertices, either there are many internally vertex disjoint paths between them (internally also disjoint from principal vertices) or there is a small hitting set which hits all such paths.

In [1] authors considered similar problem for trees: Let $F$ be a fixed set of trees. Then for a tree $T$, and any integer $k$, either there are $k$ disjoint subtrees of $T$ each of them isomorphic to an element of $F$ or there are at most $k - 1$ vertices in $T$ which they hit any model of any element of $F$ in $T$. This result was actually one of the first generalizations to Helly-type properties.

Similarly, another interesting question is to find a relation between the number of disjoint cycles and the size of feedback vertex set (FVS). It is clear that if the number of vertex disjoint cycles is big then a FVS is also big, but what about another way around? Is there any bound on the size of disjoint cycles with respect to the size of FVS?

The novel result of Erdős and Pósa answers to the above question. They show there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for every $k$, every undirected graph $G$ contains $k$ pairwise vertex disjoint cycles or a set $T$ of at most $f(k)$ vertices such that $G - T$ is a forest. In fact in their proof $f(k)$ belongs to $O(k \log k)$ [58].

Erdős and Pósa theorem is kind of generalization of Helly-type properties: If many parts intersect to each other then they all intersect in a few points.

To prove their theorem, they provide a lower bound on $f$ by using probabilistic and counting arguments. We are more interested in the upper bound though. For the upper-bound, roughly speaking, they did the following. Take maximum size set of disjoint cycles $C$ in $G$, remove all of their edges. Consider their vertices as principal vertices of Dirac’s theorem. So the set $\mathcal{H}$ of vertices which hit all disjoint paths between principal vertices together with one vertex from each cycle $c \in C$, provides the bound for the size of FVS in $G$. But if $|\mathcal{H}|$ is too big compared to the size of $C$, then we can rearrange the paths and cycles and find another set of disjoint cycles $\mathcal{C}'$ such that $|C| < |C'|$, a contradiction to the choice of size of $C$.

This property named after them as Erdős-Pósa property. The more general question is the following. Let $H$ be a fixed graph, is there any relation between the number of disjoint minor models of $H$ in any graph $G$ and the size of a hitting set which hits all minor models of $H$ in $G$?
This natural generalization answered by Robertson and Seymour: a graph $H$ has the Erdős-Pósa property if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that every graph $G$ either has $k$ disjoint copies of $H$ as a minor or contains a set $T$ of at most $f(k)$ vertices such that $H$ is not a minor of $G - T$. They showed that a graph $H$ has the Erdős-Pósa property in this sense if, and only if, it is planar. In fact, their proof implies that $H$ has the Erdős-Pósa property for minors if, and only if, there is an integer $h = f(|H|)$ such that a $(h \times h)$-grid contains $H$ as a minor.

Positive part of their proof is based on the grid theorem (see [136]), which roughly speaking, states that if the tree-width of a given graph $G$ is at least $f(h)$, then $G$ has a $(h \times h)$-grid as a minor and if there is no $(h \times h)$-grid minor, then the tree-width of $G$ is at most $f(h)$.

For any fixed planar graph $H$, we know that there is a grid $G_H$ which contains $H$ as a minor. So if the tree-width of $G$ is very big, then there are many disjoint subgrid minors in $G$ such that they contain a model of $H$. On the other hand, they generalized the results in [1] and showed that if the treewidth of $G$ is small we can use its structure (as it looks like a fat tree) and either find $k$ disjoint minor models of $H$ in $G$ or a small hitting set.

All above was for undirected graphs. In [159], Younger conjectured the generalization of Erdős and Pósa’s original result to directed graphs and directed cycles. Some intermediate results for special case of Younger’s conjecture was provided in [116] and eventually Younger’s conjectured proven by Reed et al. in [131]. In fact, they provide a computable function $f : \mathbb{N} \to \mathbb{N}$ such that, every graph either contains $k$ vertex disjoint directed cycles or its minimum feedback vertex set size is bounded by $f(k)$. In their proof, the function $f(k)$ is super-exponential, it uses nested Ramsey arguments [129] and even there, exponents are proportional to $k!$. Even though this provides a great mathematical proof for Younger’s conjecture, it is very far from the best known lower bound of $\Omega(k \log k)$. On special class of planar digraphs, Reed and Shepherd [132] show that in every planar graph $G$ either there are $k$ disjoint directed cycles or there is a set of vertices $S$ of size at most $O(k)$ such that $G - S$ is acyclic.

In this chapter, we consider the generalization of Younger’s conjecture to arbitrary digraphs $H$. For undirected graphs, there is an agreed notion of minor, but for directed graphs there are several notion of minors. Here we study the Erdős-Pósa property for two common notions of directed minors, topological minors and butterfly minors.

**Definition 4.1.1.** A digraph $H$ has the Erdős-Pósa property for topological minors if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for all $k \geq 0$, every digraph $G$ either contains $k$ disjoint subgraphs each containing $H$ as a topological minor or there is a set $S \subseteq V(G)$ of at most $f(k)$ vertices such that $G - S$ does not contain $H$ as a topological minor. The definition for butterfly minors is analogous.

For both concepts of minors we give a complete characterization of the strongly connected digraphs which have the Erdős-Pósa property. It turns out that our characterization is essentially the analogy of the above mentioned Robertson-Seymour theorem for undirected graphs. We prove that a digraph $H$ has the Erdős-Pósa property for topological minors (butterfly minors), if, and only if, there is an integer $h = f(|H|)$ such that a cylindrical wall (grid) of order $h$ contains $H$ as a topological minor (butterfly minor).

The first main theorem we prove in this chapter is the following.

**Theorem 4.3.1.** Let $H$ be a strongly connected digraph. $H$ has the Erdős-Pósa property for
butterfly (topological) minors if, and only if, there is a cylindrical grid (wall) \( G_c \), for some constant \( c = c(H) \), such that \( H \) is a butterfly (topological) minor of \( G_c \).

Furthermore, for every fixed strongly connected digraph \( H \) satisfying these conditions and every \( k \) there is a polynomial time algorithm which, given a digraph \( G \) as input, either computes \( k \) disjoint (butterfly or topological) models of \( H \) in \( G \) or a set \( S \) of \( \leq h(k) \) vertices such that \( G - S \) does not contain a model of \( H \).

This result is particularly interesting as there is no similar classification known for undirected graphs in terms of topological minors.

As a side conclusion, the above result, eventually answers the following conjecture positively.

**Conjecture 4.1.1.** There exists a function \( f : \mathbb{N} \to \mathbb{N} \) such that for every \( \ell, k \in \mathbb{N} \), every directed graph \( G \) either has \( k \) vertex disjoint cycles each of length at least \( \ell \), or there is a set of vertices \( S \subseteq V(G) \) such that \( G - S \) does not contain a cycle of length at least \( \ell \) and \( |S| \leq f(k + \ell) \).

In fact we provide the function \( f \). For previous results on special cases of above conjecture see [67, 115, 131].

On the other hand, in the proof of Theorem 4.3.1, to provide an algorithm, we introduce a new notion of directed minors which we name it tree-like model. With a short proof, we show that tree-like model is same as butterfly model, but provides much simpler and deeper understanding for algorithmic purposes. This might be of independent interests for future algorithmic works on digraphs.

If \( H \) is not strongly connected, then our techniques described above fails. In fact they fail already in the bounded directed tree width case. So in the next parts we study a larger class of vertex-cyclic digraphs, i.e. digraphs without trivial strong components (components consisting of a single vertex only). For this natural class of digraphs we obtain the following result (see Section 4.4 for details).

**Theorem 4.4.1.** Let \( H \) be a weakly connected vertex-cyclic digraph. If \( H \) has the Erdős-Pósa property for butterfly (topological) minors, then it is ultra-homogeneous (see 4.4 for definition) with respect to butterfly (topological) embeddings, its maximum degree is at most 3 and every strong component of \( H \) is a butterfly (topological) minor of some cylindrical grid (wall) \( G_k \).

We also obtain a positive result as an example of a digraph satisfying the properties in the previous theorem. This theorem is by using directed tree decompositions algorithmically in a novel way which may be of independent interest.

**Theorem 4.5.2.** Let \( H \) be a digraph consisting of two disjoint cycles joined by a single edge. There is a function \( h : \mathbb{N} \to \mathbb{N} \) such that for every integer \( k \) and every graph \( G \) either there are \( k \) vertex-disjoint topological models of \( H \) in \( G \) or there is a set \( S \subseteq V(G) \) such that \( |S| \leq h(|H| + k) \) and \( H \not\preceq_t G - S \).

Furthermore, for every \( H \) and \( k \) there is a polynomial-time algorithm which either finds \( k \) distinct topological models of \( H \) in \( G \) or finds a set \( S \subseteq G \) of vertices of size at most \( h(|H| + k) \) which hits every topological model of \( H \) in \( G \).
4.2. Directed Minors, Directed Grids and Directed Tree-Width

Directed Minors:
In this section we introduce two different kinds of minors, butterfly minors (see [86]) and topological minors, and establish same basic properties needed below.

Definition 4.2.1 (butterfly minor). Let $G$ be a digraph. An edge $e = (u, v) \in E(G)$ is butterfly-contractible if $e$ is the only outgoing edge of $u$ or the only incoming edge of $v$. In this case the graph $G'$ obtained from $G$ by butterfly-contracting $e$ is the graph with vertex set $(V(G) - \{u, v\}) \cup \{x_{u,v}\}$, where $x_{u,v}$ is a fresh vertex. The edges of $G'$ are the same as the edges of $G$ except for the edges incident with $u$ or $v$. Instead, the new vertex $x_{u,v}$ has the same neighbours as $u$ and $v$, eliminating parallel edges. A digraph $H$ is a butterfly-minor of $G$, denoted $H \preceq_b G$, if it can be obtained from a subgraph of $G$ by butterfly contraction.

We aim at an alternative characterization of butterfly minors. Let $H, G$ be digraphs such that $H \preceq_b G$. Let $G'$ be a subgraph of $G$ such that $H$ can be obtained from $G$ by butterfly contraction and let $E' \subseteq E(G')$ be the set of edges contracted in $G'$ to form $H$. Then we can partition $E'$ into disjoint sets $F_1, \ldots, F_h$ such that the edges in each $F_i$ are contracted to form a single vertex. Hence, $h = |V(H)|$ and no two edges $e_1, e_2$ from different sets $F_i \neq F_j$ share an endpoint. The edges of $G'$ not in any $F_i$ are in one-to-one correspondence to the edges of $H$. Hence, we can also define butterfly minors by a map $\mu$ which assigns to every edge $e \in E(H)$ an edge $e \in E(G)$ and to every $v \in V(H)$ a subgraph $\mu(v) \subseteq G$ which is $G'[F_i]$ for some $i$ as above. We call this a butterfly model or image of $H$ in $G$. As shown in the following lemma, we can always choose an image such that $\mu(v)$ are particularly simple.

Lemma 4.2.1. Let $H, G$ be digraphs such that $H \preceq_b G$. Then there is a function $\mu$ which maps $E(H)$ to $E(G)$ and vertices $v \in V(H)$ to subgraphs $\mu(v) \subseteq G$ such that

- $nosep \mu(u) \cap \mu(v) = \emptyset$ for any $u \neq v \in E(H),$
- $nosep$ for all $e = (u, v) \in E(H)$ the edge $\mu(e)$ has its tail in $\mu(u)$ and its head in $\mu(v),$
- $nosep$ for all $v \in V(H), \mu(v)$ is the union of an in-branching $T_i$ and out-branching $T_o$ which only have their roots in common and such that for every $e \in E(H), \mu$ is the head of $e$ then the head of $\mu(e)$ is a vertex in $T_i$ and if $v$ is the tail of $e$ then the tail of $\mu(e)$ is in $T_o.$

We call a map $\mu$ as above a tree-like model of $H$ in $G$. We define $\mu(H) := \bigcup_{f \in E(H) \cup V(H)} \mu(f).$

Proof. Suppose the claim was false. Then there are digraphs $H, G'$ such that $H \preceq_b G'$ but $H$ has no tree-like model in $G'$. We call such a pair a counter example. Choose such a pair and fix $H$. Within all $G$ such that $(H, G)$ is a counter example let $G$ be a digraph of minimal order and, subject to this, with a minimal number of edges.

For any model $\mu$ of $H$ in $G$ let us call the complexity of $\mu$ the number of edges that are contracted. Let $\mu$ be an image of $H$ in $G$ of minimal complexity. We prove by induction on the complexity that $\mu$ is tree-like. Clearly, if the complexity is 0, i.e. no edges need to be contracted, then $\mu$ is tree-like. So suppose the complexity is at least 1. Let $G' := \mu(H) \subseteq G$ be the minimal subgraph of $G$ containing all of $\mu$. By the choice of $G$, we have $G' = G$. Choose an edge $e = (u, v) \in E(\mu(v))$ for some $v \in V(H)$ that is butterfly-contractible in
We call $G$ and let $G^*$ be the digraph obtained from $G$ by contracting $e$. Let $x$ be the new vertex generated by contracting $e$. Then $H \preceq_b G^*$ and, as $G^*$ has lower order than $G$, there is a tree-like model $\mu^* \in H$ in $G^*$. If $x$ is not in $\bigcup_{v \in V(H)} \mu(v)$, then $\mu^*$ is a model of $H$ in a proper subgraph of $G$, contradicting the choice of $G$. So there is a $z \in V(H)$ such that $x \in \mu^*(z)$. Let $F^* = E(\mu^*(v))$.

We define a set $F \subseteq E(G)$ as follows. Every edge in $F^*$ is either an edge in $G$ or has $x$ as one endpoint. If $e = (w, x) \in F^*$ then $(w, u) \in E(G)$ or $(w, v) \in E(G)$ (or both). If $(w, u) \in E(G)$ and $(w, v) \in F^*$, for some $w \in V(G^*)$, then at least one of $(u, w)$ or $(v, w)$ is in $E(G)$. If $(w, w) \in E(G)$, we add it to $F$ and otherwise we add $(w, v)$. Similarly, if $(x, w) \in F^*$, then we add $(x, w)$ to $F$. For all $v' \in E(V(H))$ we set $\mu(v') := \mu^*(v')$ and we set $\mu(z) := G[F]$. Finally, for all edges $e \in V(H)$, if $\mu^*(e) \in E(G)$ then we still have a tree-like model. Hence, the only case where $\mu^*(e)$ does not contain $x$ as an endpoint we set $\mu(e) := \mu^*(e)$. If $\mu^*(e) := e'$ with $e' = (w, x)$ then $(w, u)$ or $(w, v)$ exist in $E(G)$. If $(w, u) \in E(G)$, we set $\mu(e) := (w, u)$ and otherwise we set $\mu(e) := (w, v)$. If $e = (x, w)$ we proceed analogously, setting $\mu(e) := (v, w)$ if it exists and otherwise $\mu(e) := (u, w)$.

We claim that $\mu$ is a tree-like model of $H$ in $G$. Suppose not. We know that $\mu^*$ is a tree-like model. Hence, for every $v' \neq z$, $\mu(v')$ is tree-like. So only $\mu(z)$ may violate the tree-condition. Furthermore, the edges in $\mu^*(z)$ induce a tree-like model, i.e. $\mu^*(v)$ consists of the union of an in-branching $T_i$ and an out-branching $T_o$ as in the statement of the lemma. One of the vertices in $\mu^*(z)$ is the fresh vertex $x$. Suppose first that $x \in V(T_i) \setminus V(T_o)$. If all incoming edges of $x$ in $\mu^*$ have been replaced by edges with head $u$ and the unique out-going edge by an edge with tail $v$, then $\mu(v)$ is tree-like. So at least one incoming edge of $x$ has been replaced by an edge $e_i = (w, v)$ or the unique out-going edge $e_o$ of $x$ has been replaced by $(u, w)$, for some $w \in V(G)$. If only the out-going edge has been replaced by $(u, w)$, then $v$ has no incoming and only one out-going edge to $u$, so we can simply delete $v$ from $\mu(z)$ and obtain a model. But this would violate the choice of $G$. Hence, at least one edge $(w, x)$ has been replaced by $(w, v)$. However, if the out-going edge of $x$ has been replaced by an edge $(w, v)$, then we still have a tree-like model. Hence, the only case where $\mu$ is not tree-like is if the out-going edge of $x$ in $\mu^*$ has been replaced by an edge $(u, w)$ and at least one in-coming edge of $x$ has been replaced by $(w, v)$. However, in this case the edge $(u, v)$ would not have been butterfly contractible in $G$ as it would neither be the only out-going edge of $u$ nor the only incoming edge of $v$, contradicting the choice of the edge $(u, v)$.

The other cases, i.e. if $x \in V(T_o) \setminus V(T_i)$ or $x$ is the root of $T_i$ and of $T_o$ are similar. This concludes the proof.

Hence by Lemma 4.2.1, we can from now on assume that butterfly-models are always tree-like as in the previous lemma. We will now define the other kind of minors considered in this chapter.

**Definition 4.2.2** (topological minor). Let $H$, $G$ be digraphs. $H$ is a topological minor of $G$, denoted $H \preceq_t G$, if there is a mapping $\mu$ which maps every vertex $v \in V(H)$ to a vertex $\mu(v) \in V(G)$ and assigns to every edge $e \in E(H)$ a directed path $\mu(e) \subseteq G$ such that

1. For all $v \neq w \in V(H)$ and $\mu(v) \neq \mu(w)$
   
2. For all edge $e \in E(H)$ then $\mu(e)$ is a path linking $\mu(v)$ to $\mu(w)$ and $\mu(e) \cap \left( \bigcup_{v \in V(H)} \mu(v) \cup \bigcup_{e' \neq e \in E(H)} \mu(e') \right) = \{ \mu(v), \mu(w) \}$.

We call $\mu$ a topological model of $H$ in $G$ and define $\mu(H) := \bigcup_{e \in E(H) \cup V(H)} \mu(f)$. 


That is, \( H \) is a topological minor of \( G \) if \( H \) is a subdivision of a subgraph of \( G \). We also need the following result.

**Lemma 4.2.2.** Let \( H \) be a digraph of maximum degree at most 3. If \( H \preceq_{h} G \), for some digraph \( G \), then \( H \preceq_{t} G \).

**Proof.** Let \( H \preceq_{h} G \). Hence, there is a tree-like model \( \mu \) of \( H \) in \( G \). Clearly, for \( v \in V(H) \), we can choose the in-branching \( T_{i} \) and the out-branching \( T_{o} \) comprising \( \mu(v) \) so that there are at most 3 leaves. For, if a leave of \( T_{o} \) is not the tail of an edge \( \mu(e) \), for some \( e \in E(H) \), then we can delete it from the model, unless it is the only vertex of \( T_{o} \). Similarly, we can delete leaves of \( T_{i} \) which are not the head of any \( \mu(e), e \in E(H) \). But this implies that \( T_{i} \cup T_{o} \) has only at most 3 leaves and therefore contains only one vertex \( v' \) of degree > 2. We can therefore map \( v \) to \( v' \) and edges of \( H \) to corresponding paths to obtain \( H \) as a topological minor of \( G \). \( \square \)

**Directed Tree-Width:** We briefly recall the definition of directed tree width from \([86]\).

By an *arborescence* we mean a directed graph \( R \) such that \( R \) has a vertex \( r_{0} \), called the *root* of \( R \), with the property that for every vertex \( r \in V(R) \) there is a unique directed path from \( r_{0} \) to \( r \). Thus every arborescence arises from a tree by selecting a root and directing all edges away from the root. If \( r, r' \in V(R) \) we write \( r' > r \) if \( r' \neq r \) and there exists a directed path in \( R \) with initial vertex \( r \) and terminal vertex \( r' \). If \( e \in E(R) \) we write \( r' > e \) if either \( r' = r \) or \( r' > r \), where \( r \) is the head of \( e \).

Let \( G \) be a digraph and let \( Z \subseteq V(G) \). We say that a set \( S \subseteq (V(G) - Z) \) is *Z-normal* if there is no directed walk in \( G - Z \) with the first and the last vertex in \( S \) that uses a vertex of \( G - (Z \cup S) \). It follows that every \( Z \)-normal set is the union of the vertex sets of strongly connected components of \( G - Z \). It is straightforward to check that a set \( S \) is \( Z \)-normal if, and only if, the vertex sets of the strongly connected components of \( G - Z \) can be numbered \( S_{1}, S_{2}, \ldots, S_{d} \) in such a way that

\[
\text{nosep if } 1 \leq i < j \leq d, \text{ then no edge of } G \text{ has head in } S_{i} \text{ and tail in } S_{j}, \text{ and}
\]

\[
\text{nosep either } S = \emptyset, \text{ or } S = S_{i} \cup S_{i+1} \cup \cdots \cup S_{j} \text{ for some integers } i, j \text{ with } 1 \leq i \leq j \leq d.
\]

**Definition 4.2.3.** A directed tree decomposition of a digraph \( G \) is a triple \( (T, \beta, \gamma) \), where \( T \) is an arborescence, \( \beta : V(T) \to 2^{V(G)} \) and \( \gamma : E(T) \to 2^{V(G)} \) are functions such that

\[
\text{nosep } \{ \beta(t) : t \in V(T) \} \text{ is a partition of } V(G) \text{ and}
\]

\[
\text{nosep if } e \in E(T), \text{ then } \bigcup \{ \beta(t) : t \in V(T), t > e \} \text{ is } \gamma(e)\text{-normal.}
\]

For any \( t \in V(T) \) we define \( \Gamma(t) := \beta(t) \cup \bigcup \{ \gamma(e) : e \sim t \} \), where \( e \sim t \) if \( e \) is incident with \( t \).

The width of \( (T, \beta, \gamma) \) is the least integer \( w \) such that \( |\Gamma(t)| \leq w + 1 \) for all \( t \in V(T) \). The directed tree width of \( G \) is the least integer \( w \) such that \( G \) has a directed tree decomposition of width \( w \).

The sets \( \beta(t) \) are called the *bags* and the sets \( \gamma(e) \) are called the *guards* of the directed tree decomposition. If \( t \in V(T) \) we write \( T_{t} \) for the subtree of \( T \) rooted at \( t \) (i.e. the subtree containing all vertices \( s \) such that the unique path from the root of \( T \) to \( s \) contains \( t \)) and we
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

Figure 4.1.: Cylindrical grid $G_4$.

define $\beta(T_t) := \bigcup_{s \in V(T_t)} \beta(s)$. It is easy to see that the directed tree width of a subdigraph of $G$ is at most the directed tree width of $G$.

We close the section on directed tree-width by the following lemma, which we need below.

Lemma 4.2.3. Let $T := (T, \beta, \gamma)$ be a directed tree decomposition of a digraph $G$ and let $H$ be a strongly connected subgraph of $G$. Let $S \subseteq T$ be the subgraph of $T$ induced by $\beta^{-1}(H) := \{ t \in V(T) : \beta(t) \cap V(H) \neq \emptyset \}$ and let $U \subseteq T$ be the (inclusion) minimal subtree of $T$ containing all of $S$. Then $\Gamma(t) \cap V(H) \neq \emptyset$ for every $t \in V(U)$.

Proof. Let $S$ and $U$ be as defined in the statement of the lemma. Towards a contradiction suppose that there is some $u \in V(U)$ such that $\Gamma(u) \cap V(H) = \emptyset$. Clearly, $u \notin V(S)$. By construction of $U$ this implies that there are vertices $s, t \in U$ and $v, v' \in V(H)$ with $v \in \beta(s)$, $v' \in \beta(t)$ and $s, t$ are in different components of $U - u$. Let $P_1, P_2$ be two paths in $H$ with $P_1$ linking $v$ to $v'$ and $P_2$ linking $v'$ to $v$.

As $T$ is a tree at least one of $s, t$ must be in the subtree of $T$ rooted at a child of $u$. Let $c$ be this child and assume w.l.o.g. that $s$ is in the subtree of $T$ rooted at $c$. But then $P_1 \cdot P_2$ is a directed walk starting and ending in $\beta(T_c)$ which contains a vertex, namely $v'$, not in $\beta(T_c)$. Hence, by the definition of directed tree-decompositions, $P_1 \cdot P_2 \cap \Gamma(u) \neq \emptyset$, contradicting the assumption that $\Gamma(u) \cap V(H) = \emptyset$. \hfill \qed

The following theorem follows from [86]. A linkage in a digraph $G$ is a set $\mathcal{L}$ of pairwise internally vertex disjoint directed paths. The order $|\mathcal{L}|$ is the number of paths in $\mathcal{L}$. Let $\sigma := \{(s_1, t_1), \ldots, (s_k, t_k)\}$ be a set of $k$ pairs of vertices in $G$. A $\sigma$-linkage is a linkage $\mathcal{L} := \{P_1, \ldots, P_k\}$ of order $k$ such that $P_i$ links $s_i$ to $t_i$.

Theorem 4.2.1. Let $G$ be a digraph and $T := (T, \beta, \gamma)$ be a directed tree-decomposition of $G$ of width $w$. Let $k \geq 1$ and $\sigma$ be a set of $k$ pairs of vertices in $G$. Then it can be decided in time $O(|V(G)|)^O(k^2 + w)$ whether $G$ contains a $\sigma$-linkage.

From this, we obtain the following algorithmic result that will be needed later.

Theorem 4.2.2. Let $H$ be a fixed digraph. There is an algorithm running in time $|G|^O(|H|) \cdot w$ which, given a digraph $G$ of directed tree-width at most $w$ as input, computes a butterfly model (topological model) of $H$ in $G$ or determines that $H \not\preceq_b G$. 48
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

Proof. The proof for both minor models is nearly identical. We therefore only consider the more complicated cases of butterfly minors. Let $H$ be given and let $G$ be a digraph. If $H \preceq_b G$ then, by Lemma 4.2.1, there is a tree-like model $\mu$ of $H$ in $G$. Hence, every edge $e \in E(H)$ is mapped to an edge $\mu(e) \in E(G)$ and every vertex $v \in V(H)$ is mapped to the union $\mu(v)$ of an in- and out-branching $T_i \cup T_o$. Clearly, the branchings can be chosen so that they have at most $d_H(v)$ leaves and therefore they contain at most $d_H(v)$ vertices of degree more than 2. In total, therefore there are at most $2|E(H)|$ vertices of degree more than 2 in $\bigcup_{v \in V(H)} \mu(v)$. Hence, any tree-like model of $H$ in $G$ consists of the $2|E(H)|$ endpoints of the edges $\mu(e), e \in E(H)$, of the at most $2|E(H)|$ vertices of degree more than 2 and a set of directed pairwise disjoint paths connecting them in a suitable way to form a butterfly model. Hence, to determine whether $H \preceq_b G$ we can simply iterate over all choices of $4|E(H)|$ vertices as candidates for the endpoints of edges and high degree vertices and then apply the algorithm in Theorem 4.2.1 to check for suitable disjoint directed paths. Clearly, for any fixed $H$ and fixed value of $w$ this runs in polynomial time.

Directed Grids: A natural dual to directed tree width are cylindrical grids which we define next.

Definition 4.2.4 (cylindrical grid and wall). A cylindrical grid of order $k$, for some $k \geq 1$, is a digraph $G_k$ consisting of $k$ directed cycles $C_1, \ldots, C_k$, pairwise vertex disjoint, together with a set of $2k$ pairwise vertex disjoint paths $P_1, \ldots, P_{2k}$ such that

1. Each path $P_i$ has exactly one vertex in common with each cycle $C_j$,
2. The paths $P_1, \ldots, P_{2k}$ appear on each $C_i$ in this order
3. For odd $i$ the cycles $C_1, \ldots, C_k$ occur on all $P_i$ in this order and for even $i$ they occur in reverse order $C_k, \ldots, C_1$.

For $1 \leq i \leq k$ and $1 \leq j \leq 2k$ let $x_{i,j}$ be the common vertex of $P_j$ and $C_i$.

A cylindrical wall of order $k$ is the digraph $W_k$ obtained from the cylindrical grid $G_k$ of order $k$ by splitting every vertex $v$ of total degree 4 as follows: we replace $v$ by two fresh vertices $v_i, v_h$ plus an edge $(v_i, v_h)$ so that every edge $(w, v) \in E(G_k)$ is replaced by an edge $(w, v_i)$ and every edge $(v, w) \in E(G_k)$ is replaced by an edge $(v_h, w)$.

We will also need the following result. The second part follows using Lemma 4.2.2.

Theorem 4.2.3 ([90]). There is a function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that every digraph of directed tree width at least $f(k)$ contains a cylindrical grid of order $k$ as a butterfly minor and a cylindrical wall $W_k$ as topological minor.

Finally, we need the following acyclic variant of a cylindrical grid.

Definition 4.2.5 (acyclic grid). An acyclic grid of order $k$ is a pair $(\mathcal{P}, \mathcal{Q})$ of sets $\mathcal{P} = \{P_1, \ldots, P_k\}$, $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ of pairwise vertex disjoint paths such that

- nosep for $1 \leq i \leq k$ and $1 \leq j \leq k$, $P_i \cap Q_j$ is a single vertex $v_{ij}$,
- nosep for $1 \leq i \leq k$, the vertices $v_{i1}, \ldots, v_{ik}$ are in order in $P_i$, and
- nosep for $1 \leq j \leq k$, the vertices $v_{1j}, \ldots, v_{kj}$ are in order in $Q_j$. 

49
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

4.3. The Erdős-Pósa property for Strongly Connected Digraphs

The main result of this section is the following theorem.

**Theorem 4.3.1.** Let $H$ be a strongly connected digraph. $H$ has the Erdős-Pósa property for butterfly (topological) minors if, and only if, there is a cylindrical grid (wall) $G_c$, for some constant $c = c(H)$, such that $H$ is a butterfly (topological) minor of $G_c$.

Furthermore, for every fixed strongly connected digraph $H$ satisfying these conditions and every $k$ there is a polynomial time algorithm which, given a digraph $G$ as input, either computes $k$ disjoint (butterfly or topological) models of $H$ in $G$ or a set $S$ of $\leq h(k)$ vertices such that $G - S$ does not contain a model of $H$.

We will split the proof of this theorem into two parts. We first show that strongly connected digraphs have the Erdős-Pósa property within any class of digraphs of bounded directed tree width. Here, a digraph $H$ has the Erdős-Pósa property within a class $C$ of digraphs if the condition of Definition 4.1.1 is satisfied for every $G \in C$.

**Lemma 4.3.1.** Let $C$ be a class of digraphs of bounded directed tree width. Then every strongly connected digraph has the Erdős-Pósa property within $C$ with respect to butterfly and topological minors.

**Proof.** We prove the case for butterfly minors, the case for topological minors is almost identical. Let $w$ be an upper bound of the directed tree width of all $G \in C$. We claim that we can take $f(k) = k \cdot (w + 1)$ as function witnessing the Erdős-Pósa property. We provide an algorithm which either finds a set $S$ of vertices of size at most $f(k)$ or finds $k$ disjoint copies of $H$ in $G$ as a butterfly minor.

Let $G$ be a digraph such that $\text{directed tree width}(G) \leq w$ and let $(T, \beta, \gamma)$ be a directed tree decomposition of $G$ of width $w$. We prove the claim by induction on $k$. Clearly, for $k = 0$ or $k = 1$ there is nothing to show. So suppose $k > 1$. If $H \not\preceq_b G$ then again there is nothing to show.

So suppose $H \preceq_b G$. Let $t \in V(T)$ be a node of minimal height such that $G[\beta(T_t)]$ (see the paragraph following Definition 4.2.3) contains $H$ as a butterfly minor. By the choice of $t$, $G[\beta(T_t)] - \Gamma(t)$ does not contain a model of $H$. Furthermore, by Lemma 4.2.3, no model of $H$ in $G - \Gamma(t)$ can contain a vertex in $\beta(T_t) \setminus \Gamma(t)$ and a vertex of $G - (\beta(T_t) \cup \Gamma(t))$.

Hence, all remaining models of $H$ in $G - \Gamma(t)$ must be contained in $G' := G - (\beta(T_t) \cup \Gamma(t))$. By induction hypothesis, either $G'$ contains $k - 1$ disjoint models of $H$ as butterfly minor or a set $S$ of $f(k - 1)$ vertices such that $G' - S$ does not contain $H$ as a butterfly minor. In the first case we have found $k$ disjoint copies of $H$ as butterfly minor in $G$ and in the second case the set $S' := S \cup \Gamma(t)$ hits every model of $H$. As $|S'| \leq w + 1 + f(k - 1) \leq k \cdot (w + 1) = f(k)$ the claim follows.

The next theorem follows from the previous lemma and Theorem 4.2.3.

**Theorem 4.3.2.** Let $H$ be a strongly connected digraph. If there is a $c > 0$ such that $H \preceq_b G_c$ (or $H \preceq_t G_c$), where $G_c$ is the cylindrical grid of order $c$, then $H$ has the Erdős-Pósa property for butterfly minors (resp. topological minors).

Furthermore, for every fixed strongly connected digraph $H$ satisfying these conditions and every $k$ there is a polynomial time algorithm which, given a digraph $G$ as input, either
computes $k$ disjoint models of $H$ in $G$ or a set $S$ of $\leq h(k)$ vertices such that $G - S$ does not contain a model of $H$.

**Proof.** Let $g : \mathbb{N} \to \mathbb{N}$ be the function from Theorem 4.2.3 and let $g(k, w) = k \cdot w$ be the function as defined in Lemma 4.3.1. We claim that the function $h(k) = g(k, f(k \cdot (c + 1)))$ witnesses the Erdős-Pósa property for $H$. Towards this aim, let $G$ be a digraph. If $\text{directedwidth}(G) \geq f(k \cdot (c + 1))$ then $G$ contains $k$ copies of $G_c$ each of which contains $H$ as butterfly minor. Otherwise, $\text{directedwidth}(G) < f(k \cdot (c + 1))$ and we can apply Lemma 4.3.1.

Note that, for every fixed $c$, any tree-like butterfly model of $G_c$ in a graph $G$ has directed tree-width bounded by $O(c)$. Hence, we can compute a model of $H$ in any model of $G_c$ in $G$ by Theorem 4.2.2.

We now show the converse to the previous result.

**Theorem 4.3.3.** Every strongly connected digraph $H$ which is not a butterfly minor of some cylindrical grid does not satisfy the Erdős-Pósa property.

**Construction of $G_k$:** Let $G_k = (C_1, \ldots, C_k, P_1, \ldots, P_{2k})$ be a cylindrical grid, where the $C_i$ are the concentric cycles (ordered from the inside out in a fixed embedding of $G_k$ on the plane) and the $P_i$ are the alternating paths, ordered in clockwise order on the cycles $C_j$, so that for odd $i$, the path $P_i$ traverses the cycles in order $C_1, \ldots, C_k$, i.e. from the inside out, whereas for even $i$ the cycles appear on $P_i$ in the reverse order. For $1 \leq i \leq k$ and $1 \leq j \leq 2k$ let $x_{i,j}$ be the common vertex of $P_j$ and $C_i$.

Recall that a cylindrical wall $W_k$ is obtained from $G_k$ by splitting degree 4 vertices. Note that the outer cycle $C_k$ does not have any degree 4 vertices, and therefore the following construction can also be applied to a wall $W_k$.

**Definition 4.3.1 (The digraphs $G_{i,n}^{H,e}$ and $W_{i,n}^{H,e}$).** Let $H$ be a digraph and let $e \in E(H)$ be an edge. The digraph $G_{i,n}^{H,e}$ is obtained from the disjoint union of $k$ isomorphic copies of $H$, say $H_1, \ldots, H_k$, and the grid $G_k$ as follows. In each copy $H_i$, we delete the edge $e_i = (u_i, v_i)$ corresponding to $e$. Furthermore, in $G_k$ we delete all edges $(x_{k,2i-1}, x_{k,2i})$, for $1 \leq i \leq k$. Finally, for all $1 \leq i \leq k$, we add an edge $(u_i, x_{k,2i})$ and an edge $(x_{2i-1}, v_i)$. We call $G_{i,n}^{H,e}$ the attachment of $H$ to $G_k$ and refer to the graphs $H_i$ with the edge $e_i$ deleted plus the two new edges as the $i$-th copy of $H$ in $G_{i,n}^{H,e}$.

We can apply the same construction using $W_k$ instead of $G_k$. We denote the resulting graph by $W_k^{H,e}$ and call it the attachment of $H$ to $W_k$.

See Figure 4.2 for a schematic overview of $G_k^{H,e}$. We are now ready to prove Theorem 4.3.3.

**Proof of Theorem 4.3.3.** Let $H$ be a strongly connected digraph such that $H \not\preceq_b G_k$ for all $k \geq 0$. Let $e \in E(H)$. Towards a contradiction, suppose $H$ had the Erdős-Pósa property, witnessed by a function $f : \mathbb{N} \to \mathbb{N}$. Choose a value $k > f(2)$ and let $G := G_k^{H,e}$.

We first claim that for any set $S \subseteq V(G)$ of at most $f(2)$ vertices, $G - S$ contains $H$ as a butterfly minor. To prove this, let $S$ be such a set. As $|S| < f(2)$, there is an index $1 \leq i \leq k$ such that $S$ does not contain a vertex of $C_i \cup P_{2i-1} \cup P_{2i} \cup H_i$, where $H_i$ is the $i$-th copy of $H$ in $G_k^{H,e}$. But then, $H \not\preceq_b C_i \cup P_{2i-1} \cup P_{2i} \cup H_i$. 

51
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

Figure 4.2.: Counter example to EP-property for a graph $H$. Just $H'_1$ is shown in the figure. Edge $e = (u_1, v_1)$ from $H$ deleted and edges $(u_1, y_1)$ and $(x_1, v_1)$ are added to form a connection of $H'_1$ to cylindrical grid.

To complete the proof we show next that $G$ does not contain two disjoint butterfly models of $H$. Let $\mu$ be a tree-like model of $H$ in $G$. As $H \not\preceq_b G_k$, by assumption, $\mu$ must contain a vertex $v$ in some copy $H_i$ of $H$ in $G_k^{H,e}$. But as $H$ is strongly connected and $H_i$ has fewer edges than $H$, $\mu$ must also use both edges $(u_i, x_{2i})$ and $(x_{2i-1}, v_i)$ and a directed path in $G_k$ linking $x_{2i}$ to $x_{2i-1}$. We view $G_k$ as being embedded in the plane. Then this path induces a closed curve from $x_{2i}$ to $x_{2i-1}$ in the plane splitting $G_k$ into two disjoint parts. Furthermore, the part containing the rest of the outer cycle $C_k$ not on the curve is acyclic. Hence, there cannot be a second model of $H$ in $G - \mu(H)$.

Theorems 4.3.2 and 4.3.3 together imply the proof of Theorem 4.3.1 for butterfly minors. To prove it for the topological minors, it is easily seen that the same construction as in the Theorem 4.3.3 where the grid $G_k$ is replaced by a wall $W_k$ proves that if a strongly connected digraph $H$ is not a topological minor of some fixed directed wall $W$, then $H$ does not have the Erdős-Pósa property for topological minors.

We also obtain the following consequence.

**Corollary 4.3.1.** For strongly connected digraphs, the Erdős-Pósa property (for butterfly and topological minors) is closed under strongly connected subgraphs, i.e. if a strongly connected graph $H$ does not satisfy the Erdős-Pósa property and $H \preceq_b G$ then $G$ does not satisfy Erdős-Pósa property.

4.4. The EP-Property for Vertex-Cyclic Digraphs

In this section we extend the results of the previous section to the more general class of vertex cyclic digraphs. A digraph is vertex cyclic if it does not contain a trivial strong component, i.e. if every vertex lies on a cycle. Clearly, every strongly connected digraph is
vertex cyclic but the converse is not true. For simplicity, in this section we only consider weakly connected digraphs, i.e. where the underlying undirected graph is connected.

Let $G$ be a digraph and let $e \in E(G)$. Let $n \geq 1$. We define $G^n_e$ as the digraph obtained from $G$ by subdividing $e$ $n$ times. Given digraphs $H$ and $G$, we say that $H$ is topologically s-embeddable in $G$ if there is an edge $e \in E(G)$ such that $H \preceq_e G^n_e$. We say that $H$ is butterfly s-embeddable in $G$ if there is an edge $e \in E(G)$ such that $H \preceq_b G^n_e$.

A digraph $G$ is ultra-homogeneous with respect to topological (or butterfly) minors, if the block graph of $G$ is a simple directed path without parallel edges and any two components of $G$ are pairwise topologically (or butterfly, respectively) s-embeddable into each other and furthermore if the length of the block graph is at least 3, then all of the components except the first and the last components, w.r.t. topological order, have the same size and also none of those has smaller size than the first or the last component.

**Definition 4.4.1.** A digraph $G$ is ultra-homogeneous with respect to topological (or butterfly) minors, if the block graph of $G$ is a simple directed path without parallel edges and any two components of $G$ are pairwise topologically (or butterfly, respectively) s-embeddable into each other and furthermore if the length of the block graph is at least 3, then all of the components except the first and the last components, w.r.t. topological order, have the same size and also none of those has smaller size than the first or the last component.

Our main classification result of this section is the following.

**Theorem 4.4.1.** Let $H$ be a weakly connected vertex-cyclic digraph. If $H$ has the Erdős-Pósa property for butterfly (topological) minors, then it is ultra-homogeneous with respect to butterfly (topological) embeddings, its maximum degree is at most 3 and every strong component of $H$ is a butterfly (topological) minor of some cylindrical grid (wall) $G_k$.

The first result we prove is the following.

**Lemma 4.4.1.** Let $H$ be a vertex-cyclic digraph. If $H$ contains a vertex of degree at least 4, then $H$ does not have the Erdős-Pósa property for topological minors.

**Proof.** Let $H$ be vertex-cyclic and let $v \in V(H)$ be a vertex of degree at least 4 in $H$. Furthermore, let $e$ be an incident edge of $v$. Towards a contradiction suppose $H$ had the Erdős-Pósa property witnessed by a function $f : \mathbb{N} \to \mathbb{N}$. As in the proof of Theorem 4.3.3, let $k > f(2)$. Let $W^{H,e}_k$ be the digraph defined in Definition Definition 4.3.1.

We show first that $W^{H,e}_k$ does not contain two disjoint topological models of $H$. Let $A_1, \ldots, A_l$ be the strong components of $H$ in topological order, i.e. there is no edge from $A_i$ to $A_j$ whenever $j < i$, and let $A_s$ be the component containing $v$. Let $\mu$ be a topological model of $H$ in $W$. Note that in $W^{H,e}_k$ no vertex $w \in V(W_{k})$ has degree $\geq 4$. Hence $\mu(v)$ must be in some copy $H'$ of $H$ in $W^{H,e}_k$. More precisely, $\mu(v)$ must be in the strong component of $H'$ corresponding to $A_s$. For, suppose $\mu(v)$ was in a strong component corresponding to some $A_i$ with $i < s$ such that $A_s$ is reachable from $A_i$ in $H$. Then for every $w \in V(H)$ from which $v$ is reachable in $H$, $\mu(w)$ must be in a component $A_j$ in the copy $H'$ such that $A_i$ is reachable from $A_j$. But this is impossible for cardinality reasons. Similarly, we can show that $\mu(v)$ cannot be in any other component except for $A_s$. It follows that every edge and every vertex of $A_i$ must be mapped to either the copy of $A_i$ in $H'$ or to some vertex of the
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

Wall or $A_i$ in another copy of $H$. In any case, $\mu(A_i)$ is strongly connected and therefore $\mu(A_i)$ contains both edges connecting $H'$ to the wall and a directed path between them. We can therefore argue as in the proof of Theorem 4.3.3.

To conclude the argument, we can argue as in the proof of Theorem 4.3.3 that for every set $S \subseteq V(W_k^{H,e})$ of order $< k$ the graph $W_k^{H,e} - S$ contains $H$ as topological minor. \hfill $\Box$

Note that this result does not necessarily extend to butterfly minors. We now introduce a construction that will frequently be applied below.

**Definition 4.4.2.** Let $A_k := ((P_1, \ldots, P_k), (Q_1, \ldots, Q_k))$ be the acyclic grid of order $k$ as defined in Definition 4.2.5. Recall that $V(P_i) \cap V(Q_j) = \{v_{ij}\}$. Let $H$ be a digraph and let $C_i$ and $C_j$ be distinct non-trivial strong components of $H$ so that there is an edge $e = (u, v)$ for $u \in V(C_i)$ and $v \in V(C_j)$.

nosep Let $e_2 \in E(C_2)$ be an edge incident to $v$. The left acyclic attachment graph $A_{n,H,C_1,C_2}$ of $H$ through $e$ and $e_2$ of order $n$ is defined as follows. Take a copy of $A_n = (P, Q)$ and $n$ disjoint copies $H_1, \ldots, H_n$ of $H$. For every $v \in V(H)$ we write $v^i$ for its isomorphic copy in $H_i$, and likewise we write $e^i$ for the copy of an edge $e$ in $H_i$. For all $1 \leq i \leq n$, we delete the edges $e^i = (u^i, v^i)$ and $e^i_2 = (x, y)$ and instead add the edge $(u^i, v_{i,1})$ and identify the topmost vertex $v_{1,i}$ of the $i$-th column of $A_k$ with $x$ and the last vertex $v_{k,i}$ of this column with $y$.

nosep Now let $e_1 \in E(C_1)$ be an edge incident to $u$. The right acyclic attachment graph $A_{n,H,C_1,C_2}$ of $H$ through $e$ and $e_1$ of order $n$ is defined analogously, but now the edge $e^1$ in the copy $H_i$ is replaced by an edge $(v_{i,k}, v^i)$ and the end vertices of the edge $e_1 = (x, y)$ are identified with $v_{1,i}$ and $v_{k,i}$ respectively.

It is easily seen that for all $n > 1$, $H \leq_t A_{n,H,C_1,C_2}$ and hence $H \leq_t A_{n,H,C_1,C_2}$ for all choices of $C_1, C_2, e, e_1$ as in the definition and furthermore, for all $S \subseteq V(A_{n,H,C_1,C_2})$ of order $< n$, $H \leq_s A_{n,H,C_1,C_2} - S$ and hence $H \leq_s A_{n,H,C_1,C_2} - S$. However, for some choices of $H, C_1, C_2$, $A_{n,H,C_1,C_2}$ may contain many disjoint models of $H$, for instance if $H$ only consists of two cycles $C_1, C_2$ connected by an edge.

**Lemma 4.4.2.** Let $H$ be a vertex cyclic digraph and let $C$ be the set of its components. If $H$ satisfies any of the following conditions, then it does not have the Erdős-Pósa property neither for butterfly nor for topological minors.

1. There are $C, C_1, C_2$, all distinct, and edges $e_1, e_2$ such that $e_1$ links $C$ to $C_1$, for $l = 1, 2$, or $e_1$ links $C_1$ to $C$, for $l = 1, 2$.

2. $H$ contains two components $C$ and $C'$ with two distinct edges linking $C$ to $C'$.

3. $H$ contains two distinct components $C, C'$ such that $C$ is not embeddable into $C'$ (with respect to topological minor).

4. $H$ contains a strong component $C$ such that for all $k \geq 1$, $C \not\leq_b G_k$ (resp. $C \not\leq_t W_k$).

**Proof.** We prove the cases for butterfly minors, the cases of topological minors are analogous. Towards a contradiction, suppose that $H$ has the Erdős-Pósa property witnessed by a
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

function \( f : \mathbb{N} \rightarrow \mathbb{N} \). For each item, we construct a counterexample \( A \) such that after deleting \( f(2) \) vertices from \( A \), it still has a model of \( H \).

**Proof of Item 1:** We first consider the case where there is a component \( C \) of \( H \) and two other components \( C_1, C_2 \) with an edge from \( C \) to \( C_1 \) and from \( C \) to \( C_2 \). A terminal component of \( H \) is a strong component without any outgoing edges. Let \( T \) be a terminal component with a minimal number of edges and among these with a minimal number of vertices. Let \( C \) be a component of \( H \) with edges to two distinct components and such that \( T \) is reachable from \( C \) by a path \( P \) (\( C \) exists as the block graph of \( H \) is not a path, by assumption, and \( G \) is weakly connected) and let \( S \) be the unique component of \( H \) such that \( P \) contains an edge \( e = (s, t) \in E(H) \) with tail \( s \in V(S) \) and head \( t \in V(T) \). Let \( e' = (w, t) \in E(T) \) be an edge with head \( t \), which exists as \( T \) is not a trivial component. Let \( k > f(2) \) and let \( A := A_{\rho, H, S, T} \) be the left acyclic attachment as defined in Definition 4.4.2.

See Figure 4.4 for an illustration.

Let \( H_1, \ldots, H_k \) be the copies of \( H \) in \( A \). For each vertex \( v \in V(H) \) and edge \( e \in E(H) \) we write \( e', e'' \) for the corresponding vertex or edge in \( H_i \). Note that \( e', e'' \) do not exist as they were deleted in the construction of \( A \). We denote by \( T^i \) the copy of \( T \) in \( H_i \) with the edge \( e' \) removed and by \( T_i \) the copy of \( T \) in \( H_i \) plus the path \( Q_i \) of the grid by which \( e' \) was replaced.

As noted above, after deleting a set \( D \) of \( f(2) \) vertices from \( A, H \leq_b A - D \). Hence it suffices to show that there are no two distinct butterfly models of \( H \) in \( A \).

Let \( \rho \) be a minimal tree-like butterfly model of \( H \) in \( A \), i.e. a tree-like model such that no proper subgraph of \( \rho(H) \) contains a model of \( H \). As \( \rho \) is tree-like, every \( \rho(v) \) is the union of an in-branching and an out-branching which only share their root \( r_v \). As \( \rho \) is minimal, if \( X \subseteq H \) is strongly connected, then \( \rho(X) \) contains a maximal strongly connected subgraph \( \rho(X) \) which contains every root \( r_v \) for \( v \in V(X) \). It follows that for no component \( X \) of \( H \) we have \( \rho(X) \subseteq T^i \), as \( T \) was the component with the minimal number of edges and \( T^i \) has one edge less. This implies that if for some vertex \( v \in V(H) \), \( \rho(v) \cap T^i \neq \emptyset \), for some \( 1 \leq i \leq k \), then \( T_i \subseteq \rho(X_v) \) where \( X_v \) is the component of \( H \) containing \( v \).

Now consider \( \mu(C) \). As \( \mu(C) \) is strongly connected, if \( \mu(C) \) contains a vertex of the acyclic grid \( A_k \) contained in \( A \), then \( \rho(C) = T^i \) for some \( 1 \leq i \leq k \). Let \( C_1, C_2 \) be two components of \( H \) such that \( H \) contains an edge \( e_1 \) from \( C \) to \( C_1 \) and an edge \( e_2 \) from \( C \) to \( C_2 \). But as \( T \) was chosen minimal, \( \rho(C) \cup \rho(C_1) \not\subseteq T^i \), for \( l \in \{1, 2\} \). Hence, \( \rho(C_1) \subseteq T^{j_1} \) and \( \rho(C_2) \subseteq T^{j_2} \) for some \( j_1 \neq j_2 \) different from \( i \), as otherwise there was no path from \( \rho(C) \) to \( \rho(C_1) \) and \( \rho(C_2) \) in \( A \). But as each of \( \mu(C), \mu(C_1), \mu(C_2) \) contains an entire column of the acyclic grid \( A_k \) in \( A \), this is impossible.

It follows that \( \rho(C) \) must be contained in some \( H_i \setminus \hat{T}_i \). As we cannot have \( \rho(H) \subseteq H_i \setminus \hat{T}_i \), it follows that for some \( j, T_j \subseteq \rho(H) \) and therefore \( \rho(H) \) also includes the edge from \( H_i \) to the vertex \( x_{i,1} \) of the grid and a path \( L_i \) from \( x_{i,1} \) to \( T_j \).

Now suppose \( \mu' \) is a second model of \( H \) in \( A \), which again we assume to be minimal and tree-like. By the same argument, \( \mu'(H) \) must contain an entire column \( Q_j \) and path \( L_{i'} \) from some vertex \( x_{i',1} \) to \( Q_j \). But then, if \( j' < j \), then \( Q_j \) has a non-empty intersection with \( i' \) and if \( j < j' \) then \( Q_j \) has a non-empty intersection with \( L_{i'} \). Hence, \( \mu \) and \( \mu' \) are not disjoint.

This concludes the case where \( H \) contains a component \( C \) with two outgoing edges to two distinct other components. The case where there is a component \( C \) with incoming edges
from two other distinct components is analogous, using the right acyclic attachment instead of the left acyclic attachment.

**Proof of Item 2:** Let $C$ and $C'$ be as in the statement of the Item 2 chosen so that from $C'$ no component $X$ of $H$ is reachable such that $X$ has two edges to another component $Y$. Let $c_1 = (s_1, t_1)$ and $c_2 = (s_2, t_2)$ be two distinct edges with tail in $C$ and head in $C'$.

By Item 1 we can assume that the block graph of $H$ is a directed path with parallel edges between components.

Let $k > f(2)$ and let $A_{2k} = ((P_1, \ldots, P_{2k}), (Q_1, \ldots, Q_{2k}))$ be the acyclic grid of order $2k$. Again, $V(P_i) \cap V(Q_j) = \{x_{i,j}\}$. Let $G_k$ be the graph obtained from $A_{2k}$ by adding $k$ disjoint copies $H_1, \ldots, H_k$ of $H$. For $v \in V(H)$ or $e \in E(H)$ let $v'$ and $e'$ be the vertex or edge corresponding to $v$ and $e$ in the copy $H_i$, respectively. For all $1 \leq i \leq k$ we delete the edges $e_1', e_2'$ and add edges $(s_1', x_{2i-1,1}), (s_2', x_{2i,1})$ and $(x_{2k,2i-1}, t_1'), (x_{2i,2k}, t_2')$. See Figure 4.3 for an illustration.

As $A_{2k}$ contains two disjoint paths $P_1'$ linking $x_{2i-1,1}$ to $x_{2k,2i-1}$ and $P_2'$ linking $x_{2i,1}$ to $x_{2i,2k}$, $G_k$ contains $H$ as a butterfly minor. Furthermore, it is easily seen that $H \preceq H - S$ and hence the models are not disjoint.

Hence, we only need to show that $G_k$ does not contain two distinct butterfly models of $H$. Let $\mu$ be a minimal tree-like butterfly model of $H$ in $A$, i.e. a tree-like model such that no proper subgraph of $\mu(H)$ contains a model of $H$. As $\mu$ is tree-like, every $\mu(v)$ is the union of an in-branching and an out-branching which only share their root $r_v$. As $\mu$ is minimal, if $X \subseteq H$ is strongly connected, then $\mu(X)$ contains a maximal strongly connected subgraph $\rho(X)$ which contains every root $r_v$ for $v \in V(X)$. Let $X_1, \ldots, X_k$ be the components of $H$ reachable from $C'$ in topological order. By the choice of $C$ and $C'$, between $C'$ and $X_1$ and between $X_1$ and $X_{i+1}$, for all $i < k$, there is exactly one edge.

Now, $\mu(C)$ contains a maximal strongly connected subgraph $\rho(C)$ that contains every root $r_v$ for $v \in V(C)$. As $C$ has two outgoing edges, it follows that for all $1 \leq i \leq k$, $\rho(C) \cap V(C \cup X_1 \cup \cdots \cup X_i) = \emptyset$. Clearly, $\rho(C) \cap A_{2k} = \emptyset$. Hence, there is an $1 \leq i \leq k$ such that $\rho(C)$ is entirely contained in $H_i - V(C \cup X_1 \cup \cdots \cup X_i)$. But then, $\mu(H)$ must contain the edges $(s_1', x_{2i-1,1}), (s_2', x_{2i,1})$ and $(x_{2k,2i-1}, t_1')$, $(x_{2i,2k}, t_2')$ and two disjoint paths $P_1'$ linking $x_{2i-1,1}$ to $x_{2k,2i-1}$ and $P_2'$ linking $x_{2i,1}$ to $x_{2i,2k}$.

Now let $\mu'$ be another minimal tree-like model of $H$ in $G_k$. By the same argument there must be an index $j$ such that $\mu'(H)$ contains the edges $(s_1', x_{2j-1,1}), (s_2', x_{2j,1})$ and $(x_{2k,2j-1}, t_1'), (x_{2j,2k}, t_2')$ and two disjoint paths $P_1'$ linking $x_{2j-1,1}$ to $x_{2k,2j-1}$ and $P_2'$ linking $x_{2j,1}$ to $x_{2j,2k}$. But clearly, $(P_1 \cup P_2) \cap (P_1' \cup P_2') \neq \emptyset$ and hence the models are not disjoint.

**Proof of Item 3:** Let $H$ and $C, C'$ be as in the statement of the Item 3. By Item 1 and Item 2, we can assume that the block graph of $H$ is a simple directed path without parallel edges.

Choose $C$ and $C'$ such that $C$ does not embed into $C'$ with respect to butterfly embeddings or vice versa and among all such pairs choose $C'$ so that it is the latest such component in the block graph of $H$, i.e. no component $C''$ which is part of such a pair is reachable from $C'$. We assume that $C$ has no butterfly embedding into $C'$ as defined above. The other case is analogous using right acyclic attachments instead.

Let $S \neq C'$ be the component of $H$ such that $H$ contains an edge $e = (s, t)$ with $s \in V(S)$ and $t \in V(C')$. Let $e' = (w, t)$ be any edge in $C'$ with head $t$, which must exist as $C'$ is not trivial. Now let $k > f(2)$ and let $A = A_{k, H, S, C'}$ be the left acyclic attachment as defined in
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

Definition 4.4.2. As before, $H \preceq_k A - D$ for any set $D$ of $< k$ vertices. We will show that $H$ has no two disjoint butterfly models in $A$.

Let $\mu$ be a minimal tree-like butterfly model of $H$ in $A$. Let $H_1, \ldots, H_k$ be the disjoint copies of $H$ in $A$ and as before we write $v^i, e^i$ for the copy of a vertex $v \in V(H)$ or edge $e \in E(H)$ in the $i$-th copy. Furthermore, as in the previous proofs, as $\mu$ is tree-like and minimal, every $\mu(v)$ is the union of two branchings sharing only their root $r_v$ and for every strongly connected subgraph $X \subseteq H$ the model $\mu(X)$ contains a maximal strongly connected subgraph $\rho(X)$ which contains all roots $r_v$ of $v \in X$. Let $C^\mu$ be the copy of $C'$ in $H_i$ with the edge $e^i$ removed and let $\hat{C}^\mu$ be the copy of $C'$ in $H_i$ where the edge $e'$ is replaced by the column $Q_i$ of the grid $A_k = ((P_1, \ldots, P_k, Q_1, \ldots, Q_k))$ used to construct $A$. As $C$ has no butterfly embedding in $C'$, $\rho(C)$ cannot be contained in $\hat{C}^\mu$ for any $1 \leq i \leq k$ and therefore $\rho(C) \subseteq H_i - V(C^\mu \cup X_1^i \cup \cdots \cup X_l^i)$, for some $1 \leq i \leq k$, where $X_1, \ldots, X_l$ are the components of $H$ reachable from $C'$. On the other hand, $\mu(H) \not\subseteq H_j - V(C^\nu \cup X_1^j \cup \cdots \cup X_l^j)$, for any $1 \leq j \leq k$. Hence, $\mu(C)$ must contain $\hat{C}^\nu$ for some $j$ and a path $L_i$ from $x_{i,1}$ to a vertex on $Q_j$, where $x_{i,j}$ is the unique vertex in $V(P_i) \cap V(Q_j)$, for all $1 \leq i, j \leq k$.

Now let $\mu'$ be another butterfly model of $H$ in $A$. By the same argument, $\mu'(H)$ must contain a column $Q_{j'}$ and a path $L_{i'}$ from $x'_{i',1}$ to a vertex on $Q_{j'}$. But then $\mu$ and $\mu'$ are not distinct.

Proof of Item 4: A construction very similar to the construction in the proof of Theorem 4.3.3 shows that this case holds and we omit the details.

Proof of Theorem 4.4.1 follows from Lemmas 4.4.1 and 4.4.2. In the following lemmas we narrow the provide some more subclasses of vertex digraphs that they do not admit the Erdős-Pósa property.

We abuse a notation and call a graph $H$ a cycle with a chord if $H$ is obtained from subdivision of a directed cycle with a chord.

Lemma 4.4.3. Let $H$ be a vertex cyclic graph consisting of two components $C_1, C_2$ connected to each other by a single edge $e$ (or a directed path $P$). Let $C_i$, for $i = 1, 2$, be a directed cycle with a chord. $H$ does not have the Erdős-Pósa property neither for butterfly nor for topological minors.

Proof. Let $e = (x, y)$ be an edge with the tail in $C_1$ and the head in $C_2$. Let $c_1$ be a chord in $C_1$ with $s(c_1) = u_1, l(c_1) = v_1$ and $c_2$ a chord in $C_2$ with $s(c_2) = u_2, l(c_2) = v_2$.

We may call a directed path with the start point $w$ and the end point $z$ a $S_{w,z}$ segment is a.

We prove the cases for butterfly minors, the cases of topological minors are analogous. Towards a contradiction, suppose that $H$ has the Erdős-Pósa property witnessed by a function $f : \mathbb{N} \to \mathbb{N}$.

We make a case distinction with respect to the connector edge $(x, y)$, and its vertices relative position with respect to $u_1, v_1, u_2, v_2$.

For each case we construct a counterexample $A$ such that after deleting $f(2)$ vertices from $A$, it still has a model of $H$.

Case 1. If $x$ appears in the $S_1 = S_{u_1,v_1}$ segment of the cycle $C_1$ and $y$ appears in a $S_2 = S_{u_2,v_2}$ segment of cycle $C_2$, then $H$ does not have Erdős-Pósa property.
Figure 4.3.: Illustration of the construction in the proof of Item 2 in Lemma 4.4.2.
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

Figure 4.4.: Illustration of the construction in the proof of Item 1 in Lemma 4.4.2.
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

Figure 4.5.: Two cycles with a chord. The black one is the component (not necessary strong), which has been created by the subdivisions and edge additions. The red one shows that there is a model of $C_1$ in such a component.

Proof. Let $k > f(2) + |H|$ we provide a counter example $A$ as shown in the Figure 4.6. The construction of $A$ is as follows.

We take $k$ disjoint copies $C^1_1, \ldots, C^k_1$ of $C_1$ and $k$ disjoint copies $C^1_2, \ldots, C^k_2$ of $C_2$. Subdivide the edge with endpoint at $x$ in the $S_{u,v}$-segment of each copy of $C_1$ for $2k$ times, let call new created vertices as $x_1, \ldots, x_{2k}$, we use only first $k$ of such vertices the extra vertices are to ensure some length requirements. Similarly we subdivide the edge with tail $u$ of $S_{v,u}$-segment of each copy of $C_1$ for $k$ times and we label those new vertices with $v_1, \ldots, v_k$, after that we delete the edge $(v_k, u)$ from each copy. We subdivide the first edge of the chord of each of $C^i_1$’s for $k$ times and we label those newly created vertices as $u_1, \ldots, u_k$. At the end for all $i \in [k]$ we connect each $v_i$ to $u_{k-i+1}$ and each $u_i$ to $x_i$. Let call the new graphs after the subdivisions and edge additions as $C^i_1, \ldots, C^k_1$. One of such graphs is demonstrated in the Figure 4.5.

We Add the following edges to complete the counterexample construction.

1. For each $i \in [k-1]$ add directed edge from $x_i \in C^i_1$ to $v_i \in C^i_2$.
2. For $i \in [k]$ add directed edges from $x_i \in C^k_1$ to $y \in C^i_2$.

There are is a minor model of $H$ in $A$ and sample example is shown in the Figure 4.7. Every hitting set which hits all minor models of $H$ in $G$ has size at least $k$: such a hitting set should block every path from each extended copy of $C_1$ to copies of $C_2$ and this cannot happen except using at least $k$ vertices. We prove that there are no two distinct minimal minor models of $H$ in $G$.

Every minimal minor model $H'$ of $H$ in $A$ has two strong components and furthermore one $C^i_1$ (for some $i \in [k]$) should be in the $H'$, because there is no directed path between any two $C_2$ copies in $A$.

We have the following two simple claims. The first Claim 1 states that there cannot be a model of $H$ in just one of the $C^i_1$’s.

Claim 1. $H \not\models_{\delta} C^i_1$, for all $i \in [k]$.

Proof. Removing a single vertex (vertex $v$ in Figure 4.6) from any $C^i_1$ makes $C^i_1$ acyclic, hence there are no two disjoint non-trivial strong components in $C^i_1$ and therefore $H \not\models_{\delta} C^i_1$. ⊢
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

The second claim is to show that if we use a path which goes through any extended copy of $C_1$ then we cannot use that extended copy later for strong component of any topological model of $H$ in $G$.

Claim 2. For all $\ell, i \in [k], j \in [2k]$, any $v_i, u_j$ path in $C_1^j$ intersects every cycle in $C_1^i$.

Now we claim no two distinct $C_1^i, C_1^j$ together with a directed path connecting them, provide a minimal minor model of $H$. Suppose the claim is wrong and there is such a minor model $H'$. Then there is a maximal directed path $P$ in $H'$ as a model of the edge $(x, y)$ of $H$. First suppose $P$ starts at $x_s \in C_1^i$ and ends at $v_t \in C_1^j$. $C_1^j$ should be a model of $C_2^j$ in $H$. But then the head of chord of $C_2$ which we can obtain from $C_1^j$ is $v$. But the segment $S_{v,u}$ contains the endpoint of $P$, but this is not isomorphic to the minor model of $H$.

Figure 4.8 shows that such a model of $H$ do not exists.

Let suppose $P$ starts at a vertex $x_s \in C_1^i$ and ends at a vertex $x_t \in C_1^j$ (or $u_t$). By the Claim 2, the path $P$ will hit every cycle in $C_1^j$ and hence $C_2 \not\preceq_b C_1^j - P$.

Hence, $H'$ has $C_1^i, C_2^j$ as its strong components. Suppose there is another model $H''$ with $C_1'^i, C_2'^j$ as its strong components. Without loss of generality suppose $i < i'$. Then any path that connects $C_1^i$ to $C_2^j$ intersects $C_1'^i$. Therefore there are no two disjoint minimal minor models of $H$ in $A$, consequently there are no two disjoint minor models of $H$ in $A$. This completes the proof of this case.
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

Figure 4.7.: Instance of $H$ in $A$, Case 1.

Figure 4.8.: We cannot match strong components of $H$ in two distinct $C_i^1, C_i'^1$'s.
Case 2. If $x$ appears in the $S_1 = S_{u_1,v_1}$ segment of the cycle of $C_1$ and $y$ appears in a $S_2 = S_{v_2,u_2}$ segment of cycle of $C_2$ then $H$ does not have Erdős-Pósa property.

Proof. This is analogous to the Case 1 so we omit the proof. We provide a similar counter example as the Case 1, but except that we subdivide only the $S_{u,v}$ Segments and we make the connections between $C_1^i, C_1^{i+1}$'s through those subdivided vertices. Figure 4.9 is the schematic of such a counter example. \[\square\]
Case 3. If \( x = v \in C_1, y = u \in C_2 \) then \( H \) does not have Erdős-Pósa property.

Proof. We construct a counterexample which is very similar to the one in Case 2 and we only prove the major difference relative to the Case 2.

Let \( k > f(2) + |H| \). Take \( k \) disjoint copies \( C_1^1, \ldots, C_1^k \) of \( C_1 \) and \( k \) disjoint copies of \( C_2 \) as \( C_2^1, \ldots, C_2^k \). We subdivide the last edge of \( S_{u,v} \)-segment of each copy of \( C_1 \), \( k \) times. For each \( i \in [k-1] \) add directed edge from \( x_i \in C_1^i \) to \( x_{i+1} \in C_1^{i+1} \). For \( i \in [k] \) add directed edges from \( x_i \in C_1^i \) to \( y \in C_2^i \). There is a minor model of \( H \) in \( A \). Also it is easy to verify that any hitting set which hits all minor models of \( H \) in \( G \) has size at least \( k \).

Claim 3. Let \( G \) be a subgraph of \( A \) where \( C_1^i \subseteq G \) and it has an edge from \( x_j \in C_1^i \) to \( x_j \in C_1^{i+1} \). Then the directed path \( P \) in \( G \) which starts at \( u \in C_1^i \) and ends at \( x_j \in C_1^{i+1} \) is not butterfly contractable to a single vertex.

Proof. If we consider \( P \) as a single edge, it is neither the only outgoing nor the only incoming vertex of its endpoints, therefore it is not butterfly contractable.

By Claim 3 and with a similar line of arguments as in the Case 1 we can show that there are no two disjoint models of \( H \) in \( A \).

All other cases will follow from the previous cases by applying a small changes so we omit their proof.

\[ \square \]

Lemma 4.4.4. Let \( H \) be an ultrahomogeneous directed graph with at least two non-trivial strong components. If an strong component of \( H \) has at least three edge disjoint cycles then \( H \) does not have the Erdős-Pósa property for butterfly (topological) minors.

Proof. We prove the cases for butterfly minors, the cases of topological minors are analogous. Towards a contradiction, suppose that \( H \) has the Erdős-Pósa property witnessed by a function \( f : \mathbb{N} \rightarrow \mathbb{N} \). Let \( k = f(2) + 1 \). We provide a counter example \( G \) such that for a set \( S \) of vertices of size at most \( f(2) \) vertices, \( H \preceq_b G - S \) but there are no two vertex disjoint models of \( H \) in \( G \).

Let \( C, C' \) be the last two strong components of \( H \) in the topological order of its strong components. Suppose there is an edge \( e = (x,y) \) with tail in \( C \) and head in \( C' \). Block graph of undirected graph \( Z \) is a graph obtained from contracting two connected components of \( Z \) into a single vertex. Clearly every block graph is a tree. By ?? the underlying undirected graph of \( C, C' \) \( (U(C) \text{ and } U(C')) \) has no minor of a cycle with a chord, therefore every leaf in the block graph of \( U(C) \) (resp. \( U(C') \)) corresponds to a cycle in \( C \) \( (C') \), we call such cycles as leaf cycle. Either \( x \) belongs to a leaf cycle \( c_1 \) or it does not lie on any leaf cycle, similar situation holds for \( y \) so totally there are four possibilities for \( x, y \) w.r.t the cycle that they lie on. To provide a counter example \( G \) we do a case destination.

Case 1: Both \( x, y \) are on a leaf cycle. Suppose \( x \in c_1, y \in c_1' \), where \( c_1 \subseteq C, c_1' \subseteq C' \) are leaf cycles. But there are cycles \( c_2 \in C, c_2' \in C \) which are not leaf cycles and \( c_1 \cap c_2 = \{u\}, c_2 \cap c_2' = \{v\} \). This is ensured by the structure of \( H \). Let \( e' = (u',v') \in E(c_2) \) be an edge.

Take an acyclic grid \( A^{k+1} \) and \( k \) disjoint copies \( H_i = (V(H), E(H) - e - e') \) for \( i \in [k] \). Attach \( H_i \) to \( A^{k} \) by adding edges \( e_i = (x_i, v_{i+1}), e_{i+2} = (v_{k+1}, y_i) \) and identifying \( u_i' \) with
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

Let $v_{i,1}$ and identifying $v_i'$ with $v_{i,k+1}$, where $z_i \in H_i$ is a vertex corresponding to the vertex $z$ in $H$.

It is easily seen that by any set $S$ of vertices of size at most $f(2)$, $H \preceq_b G - S$. On the other chaining together any many number of instances of $H_i$’s as above we cannot obtain model of $H$ as we need $C,C'$ have a connection together via leaf cycles. Hence for any instance of $H$ a model of $C$ which obtained from $H_i \cup A^k$ will be used and also a model of $C'$ from $H_j$, but it is easy to see that every such to models will intersect each other and we omit the proof of this.

**Case 2**: $x$ is on a leaf cycle but $y$ is not on a cycle. In this case we construct a counter example similar to the Case 4.4, except that we choose a vertex $e' \in c_1$ where $c_1$ is a leave cycle which contains $x$.

**Case 3**: $y$ is on a leaf cycle but $x$ is not on a cycle. Similar approach to previous counter examples can work here and we omit the proof.

**Case 4**: Neither of $x,y$ are on leaf cycles. Similar structure as before works here so we omit the proof.

We do a case distinction to achieve our goal.

---

### 4.5. Positive Instances for the Erdős-Pósa Property on Weakly Connected Digraphs

For weakly connected graphs showing that a graph has the Erdős-Pósa property is more complicated than the strongly connected digraphs. But there are cases we can see easily.

Let call a path of length exactly $\ell$ as $\ell$-path.

**Observation 4.5.1.** An $\ell$-path has Erdős-Pósa.

**Proof.** Let $k \in \mathbb{N}$ and suppose a graph $G$ is given. We either find $k$ disjoint $\ell$-paths in $G$ or a set of vertices $S$ of size at most $k\ell$ which hits all such paths.

Set $S := \emptyset$. If there is no $\ell$-path in $G$ we are done, otherwise find such a path $P_1$ and set $S := S \cup V(P_1)$ and set $G := G - S$. While there is a path of length $\ell$, repeat the above process until either there is no $\ell$-path or we already find $k$ such paths. In the former case the set $S$ hits all such paths and in the latter case, we have $k$ disjoint $\ell$-paths.

Observation 4.5.1 rely on one important fact:

*If there is a minor model of a path $P$ in $G$ then there is a subgraph of $G$ isomorphic to $P$.*

In fact, the butterfly contraction is not important here. We use this fact later in this section to prove one of our theorems.

Let $G$ be a strongly connected graph. Let $P = \{P_1,\ldots,P_t\}$ be a set of vertex disjoint directed paths. We attach $P_1,\ldots,P_t$ to $G$ by identifying either of their start or end vertices with a vertex of $G$. We call the new created graph $H$ a strong star. $G$ is the core of $H$ and paths in $P$ are its arms. In fact, we blow up a center of star graph by a strong component and we replace its edges with directed paths (in either of directions).

In this section, we aim to prove two main theorems. First, we classify all strong stars. This is the easier one and provides intuitions to study the second one.

**Theorem 4.5.1.** Let $H$ be a strong star. $H$ has Erdős-Pósa property if and only if its core is butterfly minor of a cylindrical grid.
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

The second and more technical theorem is a generalization of Erdős-Pósa property from cycles to two cycles connected to each other by a single edge. Recall that if there are two edges between two cycles, then the corresponding graph does not have Erdős-Pósa property.

**Theorem 4.5.2.** Let $H$ be a digraph consisting of two disjoint cycles joined by a single edge. There is a function $h: \mathbb{N} \rightarrow \mathbb{N}$ such that for every integer $k$ and every graph $G$ either there are $k$ distinct topological models of $H$ in $G$ or there is a set $S \subseteq V(G)$ such that $|S| \leq h(|H| + k)$ and $H \not\subseteq G - S$.

Furthermore, for every $H$ and $k$ there is a polynomial-time algorithm which either finds $k$ distinct topological models of $H$ in $G$ or finds a set $S \subseteq G$ of vertices of size at most $h(|H| + k)$ which hits every topological model of $H$ in $G$.

First, we prove Theorem 4.5.1. To do so, similar to the strongly connected digraphs we first show that in the class of graphs of bounded directed tree-width, strong star has Erdős-Pósa.

### 4.5.1. Strong Star

This section is dedicated to the proof of Theorem 4.5.1. We first provide some definitions and then provide a proof sketch and later we provide a concrete proof.

Let $G$ be a graph of directed tree-width at most $w$ and let $H$ be a strong star with core $C$ and arms $P = \{P_1, \ldots, P_\ell\}$ and let $k$ be an integer. Suppose $D = (T, \beta, \gamma)$ be a directed tree decomposition of $G$ of width at most $w$.

**Proof Sketch**

Similar to the strongly connected case, we find some lowest node $t$ in $T$ (a tree decomposition of $G$) such that it has some good attributes. Unlike the other section, here we are not looking for a model $H'$ of $H$ in $G[\beta(T_t)]$, rather we want to have the core of $H'$ in $G[\beta(T_t)]$ and its arms can be out of $G[\beta(T_t)]$. We eliminate all vertices in bag $t$ from $G$. We also eliminate all arms of $H'$ from $G$. We find the next such $t$ and proceed recursively. After all when the algorithm terminates, if we find many such $t$’s, it is possible to show that there are many disjoint models of $H$ in $G$. We obtain the latter by a generalized version of the Observation 4.5.1.

**Technical Proof**

For a node $t \in V(T)$ we say $t$ has a good model of $C$ if there is a minimal strong component $C' \subseteq G[\beta(T_t)]$ such that $C \subseteq C'$ and furthermore there are $\ell$ directed paths $P'_1, \ldots, P'_\ell$ in $G$ which one of their endpoints is on $C'$ and $|P'_\ell| = |P_\ell|$ and furthermore they together with $C'$ make a model $H'$ of $H$. We say $H'$ is a good model for $t$.

We say a node $t \in T$ is a good node if it has a good model of $C$ and it is lowest height node among all those nodes in $T$. We run the following procedure which is similar to the one provided in the Lemma 4.3.1. Let $l := \Sigma_{p \in P} |\ell(p)|$.

By the following algorithm we either find $k$ disjoint models of $H$ in $G$ or a hitting set of size at most $f(k + w + |H|)$ of vertices which they hit all minor models of $H$ in $G$. By proving the claim of algorithm we prove the strong star theorem.
Algorithm 5.

Find Either \( k \) Disjoint Models of \( H \) in \( G \) or a Hitting Set of Order at Most \( w \cdot k \cdot |H| \)

Set \( S := \emptyset, M := \emptyset \).
Set \( L := \sum_{p \in P} \ell(p) \).

Repeat \( k \cdot (L + 1) \) times:
1. Find a good node \( t \) of \( T \).
2. Find a good model \( H' \) of \( t \).
3. Set \( G := G - \Gamma(t) - \bigcup_{1 \leq i \leq \ell} P'_i \) where \( P'_i \)'s are the arms of model \( H' \).
4. Update \( T \) w.r.t. changes in \( G \) accordingly.
5. Set \( M := M \cup \{H'\} \).

If \( |M| = k \cdot (L + 1) \) then there are \( k \) disjoint models of \( H \) in \( G \).
Otherwise \( S \) hits every model of \( H \) in \( G \).

Before proving the proof of correctness of the algorithm we have one important observation.

Lemma 4.5.1. In the Algorithm 5, in each iteration, the new model \( H' \) does not share a vertex of its core with any other model which is available in \( M \), furthermore \( H' \) intersects at most \( L \) other models in \( M \).

Proof. If the core of \( H' \) intersects any other core then it violates the statement of Lemma 4.2.3. If core of \( H' \) intersects an arm of some \( H'' \in M \) then this contradicts the fact that the algorithm already deleted all arms of \( H'' \) from \( G \). Hence the only intersection of \( H' \) with other elements in \( M \) can happen via its arms. In fact its arms can intersect core of others. But as no two cores and no two arms can intersect, each vertex in each of arms can intersect at most one other element of \( M \). But there are at most \( L \) such vertices in all arms of \( H' \) so the second claim of lemma follows.

Now we are ready to prove that the Algorithm 5 works correctly.

Lemma 4.5.2. The Algorithm 5 either finds a set of \( k \) disjoint models of \( H \) in \( G \) or a set of vertices \( S \subseteq V(G) \) of size \( |S| \leq k(L + 1)w \) which they hit all minor models of \( H \) in \( G \).

Proof. Let \( m := k \cdot (L + 1) \) We prove that if \( |M| = m \) we can find \( k \) disjoint minor models of \( H \) in \( M \). Let order the elements of \( M \) according to the reverse order of their addition to \( M \) as \( H_1, \ldots, H_m \).

Construct a graph \( Z \) on \( L \) vertices \( v_{H_1}, \ldots, v_{H_m} \) as follows. If the path \( p_j \) in \( H_i \) uses a vertex \( u \) of core of \( H_j \) then add an edge \( \{v_{H_i}, v_{H_j}\} \) to \( E(Z) \). By choice of our ordering of \( H_i \)'s we know that \( v_{H_1} \) has at most \( L \) edges. In fact by the choice of the ordering and the Lemma 4.5.1 the graph \( Z \) has degeneracy at most \( L \) and this can be obtained by the linear order \( \sqsubseteq \) as \( v_{H_1}, \ldots, v_{H_m} \). But then there is an independent set of size \( k \) in \( Z \). This independent set can be obtained as follows:

Repeat \( k \) times.
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

1. Take the smallest element \( v \) w.r.t. \( \subseteq \) and put it in the independent set.
2. Remove \( v \) and all its neighbors from \( Z \).

In each step we find an element which is independent of previously chosen elements, furthermore, in each step, we remove at most \( L + 1 \) vertices in \( Z \). On the other hand, each vertex in \( Z \) corresponds to a model of \( H \) in \( G \) and by the construction, independent set in \( Z \) corresponds to a set of disjoint models of \( H \) in \( G \), so we are done in the first case.

But if the size of \( M \) is not as desired then the set \( S \) from the Algorithm 5 has size at most \( w \cdot k \cdot (L + 1) \) and hits all models of \( H \), which completes the proof.

**Proof of Strong Star Theorem:**

4.5.1 Let \( H \) be a strong star and \( C \) its core and \( P \) a set of its arms. Similar to the Theorem 4.3.3 we can show that if \( C \) is not butterfly minor of a cylindrical grid then \( H \) does not have the Erdős-Pósa. So we prove there is a function \( f : \mathbb{N} \to \mathbb{N} \) such that for every integer \( k \) and every graph \( G \), either there is a hitting set \( S \subseteq V(G) \) of size \( f(k + |H|) \) where it hits every butterfly model of \( H \) in \( G \) or there are \( k \) disjoint butterfly models of \( H \) in \( G \).

Let \( g \) be a function from the Directed Grid Theorem 4.2.3. Let suppose \( C \) is embeddable in a cylindrical grid of order \( c \), then \( H \) is embeddable in the cylindrical grid of order \( c|H| \). Therefore, if the directed tree-width of \( G \) is at least \( g(c|H|) \cdot (k + 1) \) then there are at least \( k \) disjoint minor models of \( H \) in \( G \).

On the other hand if directed tree-width of \( G \) is at most \( g(c|H| \cdot (k + 1)) \), by the Lemma 4.5.2 either we find a hitting set \( S \subseteq V(G) \) of size at most \( k \cdot (L + 1) \cdot g(c|H| \cdot (k + 1)) \) which hits all minor models of \( H \) in \( G \) or \( k \) disjoint models of \( H \) in \( G \).

**4.5.2. Two Cycles Connected by a Single Edge**

This section is dedicated to the proof of Theorem 4.5.2. In this section, \( H \) is a graph consisting of two disjoint cycles, connected to each other by a single edge.

**Proof Ideas (Part I)**

The proof of theorem is quite technical and a bit lengthy. But at the first step we explain some general ideas we used in the proof.

The proof scheme is similar to the proof of other positive instances: Solve the problem for bounded directed tree-width graphs first. To do this we find a minimal node in tree decomposition that has some good attributes. Remove vertices of that bag (or maybe some other vertices) and proceed recursively, until there is no minor model of \( H \) in \( G \). But there are two main issues here.

First, we should define what is a good attribute. As expected a node \( t \) has a good attribute if it is the lowest node that has a model of \( H \) in the corresponding subgraph, a subgraph obtained on vertices in bags in \( t \) and below \( t \).

Second issue and the main issue is how to proceed recursively? The issue here is that the node \( t \) does not necessarily contain any vertex of a model of \( H \). On the other hand we cannot relay on multiple good nodes to kill the cycles easily: there can be many cycles, and
there are many ways to connect them to form a model of \( H \). In this case we can use Menger like structures and kill the paths between cycles instead of cycles.

What if there are many cycles which all of them pairwise intersect? There is no problem with this. Exploiting the theorem from strong connected graphs we can kill all of them by few vertices.

What if there are many cycles and many but not all of them intersect? Here there is another observation. If there are two disjoint cycles where both of them intersect a third cycle, then we have a model of \( H \). So in the latter case we have many models of \( H \). With this ideas in mind we proceed to the formal proof in the following. Whenever it is not intuitive we provide a sketch.

We need some lemmas and definitions first. Recall that, an \( \ell \)-cycle is a cycle of length at least \( \ell \). An \( \ell \)-cluster in a graph \( G \) is a maximal subgraph of \( G \) consisting of \( \ell \)-cycles such that every two of them intersect each other. For a set \( C \) of \( \ell \)-clusters in \( G \) we write \( G[C] \) to denote the subgraph of \( G \) induced by the set of vertices occurring in an \( \ell \)-cluster in \( C \).

For three disjoint \( \ell \)-cycles \( C_1, C_2, C_3 \subseteq G \), the cycle \( C_1 \) is an \( \ell \)-transit cycle of \( C_2, C_3 \) if there is a path \( P \) in \( G \) which connects \( C_2 \) to \( C_3 \) and \( P \cap C_1 \neq \emptyset \). A cycle \( C \) is an \( \ell \)-transit cycle if it is an \( \ell \)-transit cycle for some pair \( C_2, C_3 \) of disjoint \( \ell \)-cycles. A set \( \mathcal{C} \) of \( \ell \)-clusters in \( G \) is bipartite if all \( C \in \mathcal{C} \) are pairwise vertex disjoint and there is no \( \ell \)-transit cycle in \( G[\mathcal{C}] \). A graph \( G \) is \( \ell \)-cluster bipartite graph if all of its \( \ell \)-clusters together form a bipartite set of \( \ell \)-clusters. By Theorem 4.3.2, a single \( \ell \)-cycle has the Erdős-Pósa property, witnessed by some function \( f_1: \mathbb{N} \to \mathbb{N} \). In particular \( f_1(2) \) means that in a given graph \( Z \) either there are two disjoint cycles of length at least \( \ell \) or there is a set \( S_1 \) of size at most \( f_1(2) \) such that there is no cycle of length at least \( \ell \) in \( Z - S_1 \).

The following lemma is required just for algorithmic aspects of Theorem 4.5.2. Our proof is constructive and from the proof and the following lemma, one can provide an algorithm.

**Lemma 4.5.3.** There is an algorithm which for a given \( \ell \)-cluster bipartite graph \( G \) of directed tree-width at most \( w \), finds all of its \( \ell \)-clusters in time and space \( O(w^{\ell+1}) \).

**Proof.** We first observe that we can check whether a vertex \( v \in V(G) \) lies in an \( \ell \)-cycle or not, and if it is in some \( \ell \)-cycle find at least one of those cycles, namely, its corresponding \( \ell \)-cycle. To see this for a vertex \( v \in V(G) \) we guess \( l-1 \) distinct other vertices which together with \( v \) form a model of \( \ell \)-cycle. By Theorem 4.2.1 we can check if they form an \( \ell \)-cycle in time and space \( |G|^{O(w\ell)} \). So we can find corresponding \( \ell \)-cycle of each vertex \( v \in V(G) \). We put two vertices \( u, v \in V(G) \) in one \( \ell \)-cluster if their corresponding \( \ell \)-cycles intersect. Recall that in an \( \ell \)-cluster bipartite graph it is impossible to have three \( \ell \)-cycles \( C_1, C_2, C_3 \) such that \( C_1 \cap C_2 \neq \emptyset \) and \( C_2 \cap C_3 \neq \emptyset \) and \( C_1 \cap C_3 = \emptyset \), because then \( C_2 \) is an \( \ell \)-transit cycle.

We use the following essential lemma in the rest of this section.

**Proof Ideas (Part II):** Before providing the proof of the following lemma, we explain a bit about the spirit of the lemma and its usages. Lemma is somehow generalization of duality between vertex cover and matching: either there is a big matching or there is a small vertex cover. In fact \( \ell \)-cluster bipartite graph resembles this structure: each vertex corresponds to an \( \ell \)-cluster and the paths between \( \ell \)-clusters are edges. If there is a big matching in this auxiliary graph then there are many disjoint models of \( H \) in the original graph. Otherwise,
we can provide a small hitting set: the proof of this part has some nice technical details that we provide them formally later. We also provide intuition for those parts in the proof itself.

On the other hand how do we use this lemma in the rest of this section? We cannot remove all instances of $H$ simply without deleting many vertices. To ease this operation, we remove some vertices from the original graph, to divide it in smaller subgraphs each of them is an $\ell$-cluster bipartite graph. Once we have this good attribute, we apply the previous trick to either find many disjoint minor models of $H$ or a small hitting set.

Assumptions: In the rest of this section, $\ell, s$ are the length of largest and shortest cycles of $H$, respectively.

Lemma 4.5.4. Given an integer $k$ and $\ell$-cluster bipartite graph $G$. Either there are $k$ disjoint minor models of $H$ in $G$ or there is a set $S_k \subseteq V(G)$ such that it hits every model of $H$ in $G$. Furthermore $|S_k| \leq 2(k - 1) \cdot f_1(2) + (\ell - 1)k(\max\{f_1(2), \ell\}) + k - 1$.

Proof. We break the proof into three steps. In the first step, we kill all minor models of $H$ which have two $\ell$-cycles. After that we kill every minor model $H'$ of $H$ which is completely in one $\ell$-cluster. At the end we kill all remaining minor models of $H$. Note that in the last case we have a crucial information, first of all the smaller cycle of any remaining instance has at most $\ell - 1$ vertices. This allows us to kill such instances by removing all their vertices. In any step if we do not manage to kill all instances of $H$ with few vertices, we provide $k$ disjoint minor models of $H$. Having these intuitions in mind we proceed to the formal proof.

Step I: First we either find $k$ disjoint models of $H$ which have 2 $\ell$-cycles or a set $S_1 \subseteq G$ of size at most $k - 1 + (k - 1) \cdot f_1(2)$ which hits every model of $H$ in $G$ which has two $\ell$-cycles. To ease to read we call such models as $\ell$-models.

We know every such model has its cycles in two different $\ell$-clusters.

Let $C$ be the set of all $\ell$-clusters in $G$. If there is an element in $C$ which does not have a path to (or from) any other element of $C$ in $G$ then it is not part of any minor of $\ell$-model of $H$ so we can ignore them. We partition the rest of $C$ into two partitions $C_1, C_2$ such that for any element in $C_1$ there is a path to some element in $C_2$. As none of the elements has an $\ell$-transit cycle, we always have this bi-partition.

We add a vertex $v_1$ to $G$ and for each $C' \in C_1$ an edge from $v_1$ to a vertex in $C'$. Similarly, add a vertex $v_2$ and for each $C'' \in C_2$ an edge from one vertex of $C''$ to $v_2$.

By Menger’s theorem, either there are $k$ disjoint paths from $v_1$ to $v_2$ or there is a set of vertices of size at most $k - 1$ which disconnects $v_1$ from $v_2$.

First we claim that if there are $k$ disjoint paths from $v_1$ to $v_2$, then we have $k$ disjoint copies of $H$ as required. It is clear that any path from $v_1$ to $v_2$ corresponds to a model of $H$ in $G$ with both cycles in $G[C]$. On the other hand, two models from two disjoint paths may intersect only if they go through each others components in $C_1$ or $C_2$. But this cannot happen, as otherwise we have an $\ell$-transit cycle in $C$.

Similarly, if there is a vertex set $W$ of size at most $k - 1$ that disconnects $v_1$ and $v_2$, then for every $v \in W \cap V(G[C_1 \cup C_2])$ let $S_v$ be the set of vertices of size at most $f_1(2)$ which hits every cycle of length at least $\ell$ in the $\ell$-cluster that $v$ belongs to. Let $S_1 = W \cup \bigcup_{v \in W} S_v$. Then $S_1$ is a hitting set of every model of $H$ in $G$ which obtained from 2 disjoint $\ell$-cycles. But size of $S_1$ is at most $k - 1 + (k - 1) \cdot f_1(2)$ as claimed.

Step II: In the second step we consider each $\ell$-cluster indivisually in $G - S_1$. 

70
Each $\ell$-cluster is strongly connected. Suppose there are $t$ disjoint $\ell$ clusters $C_1, \ldots, C_t$ such that for all $i \in [t] : H \not\subseteq C_i$. If $t \geq k$ then we have $k$ disjoint models of $H$ in $G$.

Otherwise for all $i \in [t]$ we can choose a set $S_i' \subseteq V(C_i)$ of size at most $f_1(2)$ vertices such that $C_i - S_i'$ has no $\ell$-cycle. Let $S_2 = \bigcup_{i=1}^t S_i'$. We have $|S_2| \leq (k - 1)f_1(2)$. In $G - S_1 - S_2$ there is no model of $H$ which has both of its cycles in one $\ell$-cluster.

**Step III:** In the third step we proceed on $G - S_1 - S_2$. Take a set $C'$ of all $\ell$-clusters in $G - S_1 - S_2$. Take a set of corresponding small cycles $C''$ of maximum size which consisting of disjoint cycles of size at least $s$ in $G - S_1 - S_2 - C'$. By our choice of $S_1, S_2, C'$ it is clear that every corresponding small cycle has length at most $\ell - 1$. Like a first step, add a vertex $v_1$ to $G - S_1 - S_2$ and for each $C' \in C'$ an edge from $v_1$ to a vertex in $C'$. Add a vertex $v_2$ and for each $C'' \in C''$ an edge from one vertex of $C''$ to $v_2$. By Menger theorem either there are $(\ell - 1)k$ internally vertex disjoint paths $P$ from $v_1$ to $v_2$ or there is a hitting set of size at most $k(\ell - 1) - 1$ which hits every path from $v_1$ to $v_2$.

In the first case we can find $k$ disjoint models of $H$ as follows. For $P = \{v_1, u_1, \ldots, u_n, v_2\} \in P$, we say $u_1$ is the start point and $u_n$ is the end point of the path $P$. We know that each path in $P$ denotes a model of $H$. Furthermore, by the first step (choice of $S_1$) start point of each two paths are on two disjoint $\ell$-cycles $c_1, c_2$ and there is no path between $c_1, c_2$.

Each corresponding small cycle can route at most $\ell - 1$ paths. We give the following recursive algorithm to find a set $H$ of at least $k$ disjoint models of $H$ in $G - S_1 - S_2$. Take a path $P \in P$ and let $c$ be its endpoint which corresponds to the small cycle. Suppose $P_1', \ldots, P_\ell' \in P$ intersecting $c$. We know that $t \leq \ell - 1$ as the size of $c$ is at most $\ell - 1$. Put the corresponding model of $H$ w.r.t. $P$ in $H$. Set $P := \{P_1', \ldots, P_\ell'\}$ and recurse.

In each step, the algorithm finds a model of $H$ which is disjoint from any other model which is already in $H$, so at the end $H$ consists of disjoint models of $H$. Furthermore, in each step algorithm eliminates at most $\ell - 1$ paths from $P$. So algorithm will run for at least $k$ steps, that follows $H$ has at least $k$ disjoint models of $H$.

If there is a hitting set $S$, then for every $v \in S$ we create a set $S_v$ as follows. We set $S_v := \{v\}$. If $v \in S \cap C'$ set the $S_v \subseteq V(G)$ of size at most $f_1(2)$ which hits every $\ell$-cycle in strongly connected component of $v$. If a vertex $v \in S \cap C''$ then $v \in c$ for some $c \subseteq C''$ and we set $S_v := V(c)$, in this case we have $|S_v| \leq \ell - 1$. Set $S_3 := \bigcup_{v \in S} S_v$. We claim $S_3$ hits every model of $H$ in $G - S_1 - S_2$.

Suppose there is a model $H'$ of $H$ in $G - S_1 - S_2 - S_3$ consisting of cycles $c_1, c_2$ with a path from $c_1$ to $c_2$. In our construction, there is no path between $v_1, v_2$ by the choice of $S_3$, so either the $c_1$ has no incoming edge from $v_1$ or the $c_2$ has no edge to $v_2$.

We claim either $c_1$ or $c_2$ does not exist so there is no such $H'$ at all. Suppose $c_1$ exists. We know that $c_2$ is not in any $\ell$-cluster of $G - S_1 - S_2$ (recall the choice of $S_2$). So $c_2$ is a cycle disjoint from any $\ell$-cluster and therefore either is in $C''$ or intersects $c \subseteq C''$. As $c_1$ exists, it means we did not take any vertex from its $\ell$-cluster into $S_3$, so there is a path from $v_1$ to $c_1$ and therefore to $c_2$. In order to destroy connections from $v_1$ to $v_2$ we chose a vertex $v \in c$ by Menger algorithm and therefore $V(c) \cap S_3 = V(c)$, but then $c_2 \cap S_3 \neq \emptyset$, so $c_2$ does not exist.

The size of $S_3$ is at most $(\ell - 1)(k - 1)(\max\{f_1(2), \ell\})$. There is no model of $H$ in $G - S_1 - S_2 - S_3$, we set $S_k = S_1 \cup S_2 \cup S_3$. The size of $S_k$ is at most $2(k - 1) \cdot f_1(2) + (\ell - 1)k(\max\{f_1(2), \ell\}) + k - 1$ as claimed. □

Now we are ready to prove Erdős-Pósa property holds for $H$ in the class of bounded
directed tree width graphs. For the proof of the following lemmas, we need a special form of directed tree decompositions. A directed tree-decomposition \((T, \beta, \gamma)\) is special, if

\[
\forall e = (s, t) \in E(T) \text{ the set } \beta(T_t) := \bigcup_{t \leq t'} \beta(t') \text{ is a strong component of } G - \gamma(e)
\]

and

\[
\forall t \in V(T) \text{ the set } \bigcup_{t \leq t'} \beta(t') \cap \bigcup_{e \subseteq t} \gamma(e) = \emptyset \text{ for every } t \in V(T).
\]

It is possible to show that every digraph of directed tree width \(w'\) has a special directed tree decomposition of width at most \(5w' + 10\).

**Proof Sketch (Part III)**

As usual in such cases we find a smallest node of \(T\) which has a model of \(H\) in the subgraph corresponding to vertices in bags of subtree \(T_t\) rooted at \(t\). In this proof sketch we simply call this instance of \(H\) in \(T_t\). In \(T_t\) we may have many minor models of \(H\). We further restrict \(T_t\), we consider the minimum subtree \(T'_t\) of \(T_t\) which has the smallest instance of \(H\), w.r.t. topological order of children of \(t\). First we observe that we can kill all instances of \(H\) fully contained in \(T'_t\) by few vertices. Then we proceed on \(T_t\) and eliminate all instances which are in \(T_t - T'_t\) by few vertices, and after that we proceed all instances in \(T'_t\): they should have a vertex in \(T_t - T'_t\) and a vertex in \(T'_t\). This property and the fact that \(T'_t\) is topologically smallest subtree, gives tools to be able to eliminate instances in \(T_t\) by few vertices. And then we remove all instances in \(T - T_t\) by induction. However, it remains to remove instances which have a vertex in \(T_t - T'_t\) and a vertex in \(T'_t\), but we can do this by **Lemma 4.5.4**. Note that we use the **Lemma 4.5.4** in all other steps. In all above cases if we fail to kill models of \(H\) by few vertices we find \(k\) disjoint models of \(H\).

**Assumptions**

In the rest, for the definition of \(G_0, G_1, \ldots\) we refer the reader to the proof of **Lemma 4.5.5**. Additionally, we abuse the notation and simply use \(S_k\) to denote the size of the set \(S_k\) in the proof of **Lemma 4.5.4**.

**Lemma 4.5.5** (lemma). There is a function \(f(k, w) : \mathbb{N} \times \mathbb{N} \to \mathbb{N}\) such that for every \(k, w \geq 0\), every digraph \(G\) of directed tree-width at most \(w\) either contains \(k\) disjoint topological models of \(H\) or a set of at most \(f(k, w)\) vertices hitting every model of \(H\).

**Proof.** We divide the proof of the lemma into three smaller lemmas. Before breaking it down, we provide some general definitions and observations here.

Let \(G\) be a digraph of directed tree-width at most \(w\) and let \((T, \beta, \gamma)\) be a directed tree-decomposition of \(G\) of width \(w\). For \(t \in T\) let \(G_t := G[\beta(T_t)]\). We prove by induction on \(k\). Clearly, for \(k = 0\) or \(k = 1\) there is nothing to show. In fact we do not need to provide a small hitting set, either there is an instance or there is no model of \(H\) in \(G\). We set \(f(0, w) := f(1, w) := 0\) and for \(k > 1\) we set \(f(k, w) := 5w + 10 + 2f_1(2) + f(k - 1, w) + 3|S_k|\), where \(S_k\) is as provided in the **Lemma 4.5.4**.

If \(H \not\leq_t G\), then again there is nothing to show. Otherwise, we define \(t\) be a node in \(T\) of minimal height such that \(H \not\leq_t G_t\). By definition of special directed tree-decompositions, for
every successor $c$ of $t$ the digraph $G_c$ is strongly connected. So if $c$ is a successor of $t$ then $G_c$ does not contain two disjoint $\ell$-cycles as otherwise $H \preceq_\ell G_c$ contradicting the choice of $t$. So there is a hitting set $S_c$ of size at most $f_1(2)$ such that $G_c - S_c$ has no cycle of length at least $\ell$.

Let $\subseteq$ be a linearisation of the topological order of the children of $t$. Let $F = (c_1, \ldots, c_m)$ be the tuple of children of $t$ satisfying the following conditions:

(i) $H \preceq_\ell F(t) := \Gamma(t) \cup \bigcup_{c \in F} G_c$.

(ii) Subject to 1(i), $F$ is the lexicographically smallest tuple w.r.t. $\subseteq$.

It is easy to see that there are no 3 distinct nodes $c_1, c_2, c_3 \in F$ such that there is a cycle of length at least $\ell$ in $G_{c_1} - \Gamma(t), G_{c_2} - \Gamma(t), G_{c_3} - \Gamma(t)$, as otherwise we could choose a smaller set $F$ satisfying the conditions, contradicting the fact that $F$ satisfies the second condition.

So suppose there are at most two nodes $c_1, c_2$ in $F$ containing a cycle of length at least $\ell$ in $G_{c_2} - \Gamma(t)$ and $G_{c_2} - \Gamma(t)$, respectively. Let $S(t) := \Gamma(t) \cup S_{c_1} \cup S_{c_2}$. By construction, $S(t)$ hits every cycle of length at least $\ell$ in $F(t)$. Hence, in $G_0 := F(t) - S(t)$ there is no minor of $H$ but there is a minor of $H$ in $F(t)$.

If $G - F(t)$ contains $k - 1$ disjoint topological models of $H$ then this implies that $G$ has $k$ disjoint models of $H$ and we are done. Otherwise, by induction hypothesis, there is a set $S \subseteq V(G - F(t))$ of order at most $f(k - 1, w)$ such that $H \preceq_\ell G - F(t) - S$. Note that every model of $H$ in $G - S - S(t)$ must contain vertices of $G_0$ and also vertices of $G - S - S(t) - F(t)$. Let $G_1 := (G - S - S(t) - F(t)) \cap G[\beta([T])]$ and $G_2 := (G - S - S(t) - F(t)) - G_1$. In the rest of the proof, we will first construct a hitting set for every model of $H$ in $G_{0,1} := G_0 \cup G_1$, then construct a hitting set of models of $H$ which have both of their cycles in $G_2$ connected by a path containing vertices of $G_1 \cup G_0$ and finally find a hitting set of models of $H$ which have one cycle in $G_2$ and the other in $G_1$. In any of the three cases, if we fail to find the required hitting set, we output $k$ disjoint models of $H$. As no other choice of any model of $H$ remains, we are done with the proof. By Lemma 4.5.6 we have that $G_{0,1}$ is an $\ell$-cluster bipartite graph so by Lemma 4.5.4 either it has $k$ disjoint minor models of $H$ or it there is a set of vertices $S_{G_0,1}$ of size at most $S_k$ such that $H \not\preceq_\ell G_{0,1} - S_{G_0,1}$. By Lemmas 4.5.8, 4.5.4 either there are $k$ disjoint models of $H$ in $G - S - S(t) - S_{G_0,1}$ such that both of their cycles are in $G_2$ or there is a set of vertices $S_{G_2} \subseteq V(G - S - S(t))$ such that there is no model of $H$ in $G' := G - S - S(t) - S_{G_0,1} - S_{G_2}$ with both of its cycles in $G_2 - S_{G_0,1} - S_{G_2}$. In the first case we are done. So suppose we have $S_{G_2}$ as in Lemma 4.5.4.

Any model of $H$ in $G'$ must map one cycle of $H$ to $G_{0,1} - S_{G_0,1} - S_{G_2}$ and the other to the $G_2 - S_{G_0,1} - S_{G_2}$.

Let $C$ be the set of clusters in $G - S - S(t) - S_{G_0,1} - S_{G_2}$. All clusters in $C$ are pairwise disjoint.

There is no path between the two clusters $c_1, c_2 \in G[C] \cap G_{0,1} - S_{G_0,1} - S_{G_2}$ because there is no such path fully in $G_{0,1}$ and it cannot go through a vertex $v \in G_2 - S_{G_0,1} - S_{G_2}$ by Lemma 4.2.3. There is no cluster $c$ in $G_{0,1} - S_{G_0,1} - S_{G_2}$ such that it has a path to a cluster $c' \in G_2 - S_{G_0,1} - S_{G_2}$ and a path from a cluster $c'' \in G_2 - S_{G_0,1} - S_{G_2}$, as otherwise there is a path between $c', c''$ in $G'$. So $C$ is a bipartition and by Lemma 4.5.4 either there are $k$ disjoint minors of $H$ in $G'$ or there is a hitting set $S_{G'}$ of size at most $w + 2f_1(2) + f(k - 1, w) + 3|S_k|$ which hits every minor of $H$ in $G$. 

\qed
Lemma 4.5.6. $G_{0,1}$ is an $\ell$-cluster bipartite graph.

Proof. By construction there is no cycle of length at least $\ell$ in $G_0$. Also by construction there is no minor of $H$ in each of $G_0, G_1, G_2$. By Lemma 4.2.3 there is no path $P$ in $G[G_0 \cup G_1 \cup G_2]$ with start and end point in $G_1$ such that $P \cap G_2 \neq \emptyset$.

If $H \preceq G_{0,1}$, then let $C_{G_{0,1}}$ be the set of all $\ell$-clusters in $G_{0,1}$. As $G_0$ does not contain any cycle of length at least $\ell$, the clusters in $C_{G_{0,1}}$ are all contained in $G_1$. Furthermore, no two distinct clusters can share a vertex as otherwise there would be a minor of $H$ in $G_1$. Finally, in $G_{0,1}$ there cannot be an $\ell$-transit cycle as otherwise the choice of $F$ would not have been minimal w.r.t. $\subseteq$. For, suppose there was an $\ell$-transit cycle $C_1$ in $G_{0,1}$, i.e. there are $\ell$-cycles $C_1, C_2, C_3$ in $G_{0,1}$ and a path from $C_2$ to $C_3$ containing a vertex of $C_1$. As $G_0$ does not contain any $\ell$-cycle, $C_1, C_2, C_3$ are all in $G_1$. But as $G_1$ does not contain $H$ as a topological minor, the subpath of $P$ from $C_2$ to $C_1$ and also the subpath of $P$ from $C_1$ to $C_3$ must contain a vertex of $G_0$. But this implies that $F$ does not satisfy the second condition.

So $G_{0,1}$ is an $\ell$-cluster bipartite graph.

We need the following useful lemma for both proof of Lemma 4.5.8 and Lemma 4.5.5.

Lemma 4.5.7. There are no three cycles $c_1, c_2, c_3$ of length at least $\ell$ in $G_2$ such that there is a path $P_1$ from $c_1$ to $c_2$ and a path $P_2$ from $c_2$ to $c_3$ in $G - S - S(t)$.

Proof. We know that there is no minor of $H$ in $G_2$. If there are 3 cycles as stated in the claim, then both paths $P_1, P_2$ must contain a vertex of $G_0 \cup G_1$. But then there is a path between two vertices of $G[\beta(T_t)] - \Gamma(t)$ which does not go through $\Gamma(t)$ but intersects $G - G[\beta(T_t)]$. But this is impossible by Lemma 4.2.3.

Now we have to consider all $\ell$-clusters in $G_2$.

Lemma 4.5.8. $G_2$ is an $\ell$-cluster bipartite graph.

Proof. This follows from the Lemma 4.5.7 and the fact that all $\ell$-clusters in $G_2$ are vertex disjoint.

Proof of Theorem 4.5.2. Let $H$ be as in the statement of the theorem with two cycles $C_1, C_2$. Let $\ell, s$ be the length of $C_1, C_2$ resp. such that $\ell \geq s$. W.l.o.g suppose there is an edge from $C_1$ to $C_2$.

Let $g : \mathbb{N} \rightarrow \mathbb{N}$ be the function as defined in Theorem 4.2.3. We claim that $\hat{h} : \mathbb{N} \rightarrow \mathbb{N}$ defined by $\hat{h}(k) := f(k, g((k + 1) \cdot \ell))$ witnesses the Erdős-Pósa property of $H$. To see this, let $G$ be any digraph and let $k \geq 1$. If the directed tree-width of $G$ is at least $g((k + 1) \cdot \ell)$, then by Theorem 4.2.3, $G$ contains the cylindrical wall $W_{(k+1) \cdot \ell}$ of order $(k + 1) \cdot \ell$ as topological minor, which contains $k$ disjoint copies of $H$ as topological minor. Otherwise, i.e. if the directed tree-width of $G$ is $< g((k + 1) \cdot \ell)$, then by Lemma 4.5.5, $G$ contains $k$ disjoint topological models of $H$ or a set $S$ of at most $f(k, g((k + 1) \cdot \ell))$ vertices such that $H \not\preceq G - S$.
4. On Dualities in Digraphs: Erdős-Pósa property in Digraphs

4.6. Conclusion and Future Work

We studied a special case of duality in digraphs. The duality between multiple occurrences of a specific pattern and the number of nodes to delete from the network to eliminate all instances of that pattern. We provide a full classification of Erdős-Pósa property on strongly connected digraphs. Our classification was analogous to one for undirected graphs. On the other hand, we also studied the class of vertex cyclic graphs. We provide a good classification for those graphs but yet not a complete classification.

For the case of strongly connected graphs, it is interesting to check whether there is an algorithm which can decide a given graph has Erdős-Pósa property. We show that every graph which embeds in a cylindrical grid has Erdős-Pósa property, but recognizing whether a graph is embeddable on a cylindrical grid is an interesting open problem. In some sense, this can lead to a theorem, analogous to Kuratowski’s theorem.

The problem gets more interesting in vertex cyclic graphs. We show that many simple structures in vertex cyclic graphs do not admit Erdős-Pósa property. On the positive side, we show that if we have two cycles with an edge connecting those two, then this graph has Erdős-Pósa property. This positive result extends the Younger’s conjecture from a single cycle to weakly connected cycles. The proof is a bit sophisticated but provides new ideas on how to use directed treewidth and structural properties of directed graphs.

There is a very simple structure, in vertex cyclic graphs, which we do not know if it has an Erdős-Pósa property. This structure is consisting of 3 directed cycles chained together. In another word, take a directed path of length two, and replace its vertices with directed cycles. Such a simple graph is a new mystery. At first glance, this looks like the case of two cycles. In two cycles connected together with an edge, it was not important which vertex is the head or tail of that edge. Unlike two cycles, for three cycles, it is important to know which vertex is the head/tail of edges which are connected to the second cycle. In fact, we should consider the distance between those two vertices. We believe that the idea which can solve the three cycles chained together can lead to a complete characterization of vertex cyclic digraphs. Clearly, still there are missing pieces in the puzzle, but it seems that they are not that big issue.

Up to here, we did not discuss efficiency of our approaches. The directed grid theorem [90] exhausts a superexponential dependency to the size of the grid. Hence, our algorithmic approaches are not efficient in practice and they have more theoretical values. This is not for our general case, the more restricted case of proof of Younger’s conjecture, which was only for directed cycles [131], has also a superexponential dependency to the number of disjoint cycles \( n \) we are looking for. On the other hand, the best known lower bound is \( O(n \log n) \) [131]. This already shows a big gap between the lower bound and upper bound for the simplest possible case. A general question concerns about narrowing this gap. In fact, it is interesting to know what is a tight upper bound on the ratio between the number of vertices one should remove from a graph to make it acyclic and the maximum number of disjoint cycles in the graph.
Part III.

Routing and Connectivity Problems
5. On Disjoint Paths Problem in Digraphs

5.1. Introduction

The disjoint paths problem is an extensively studied problem in graph theory and theoretical computer science and it also has important practical applications, e.g. as we already mentioned in the introduction, it appears in VLSI design and various routing problems. In designing network algorithms one need to route paths through a small number of hubs, which introduces the length bounded variant of the disjoint paths problem, e.g. see [24, 28, 61, 107]. The disjoint paths problem also finds application in scheduling and timetabling [60].

Unlike Menger’s theorem, which deals with \( k \)-connectivity, the \( k \)-(vertex/edge) disjoint paths problem is intractable [110], it is NP-complete even on undirected planar graphs. For directed graphs the problem remains hard even for two disjoint paths [65]. On the positive side, several algorithms have been developed for the 2-disjoint paths problem on undirected graphs [126, 143, 144, 151]. There is also a parallel algorithm for this problem [91].

We parametrize the problem by the number of paths \( k \). Robertson and Seymour proved in Graph Minors XIII [135], as part of their graph minors project, that the \( k \)-disjoint paths problem is fixed parameter tractable on undirected graphs. Their proof is not constructive and until today no constructive proof is known. They introduced a technique of irrelevant vertex elimination.

Here we roughly explain this technique. It is based on the following case distinction. If the graph has a small tree-width, we exploit the standard dynamic programming technique for bounded tree width graphs. If the tree-width of the graph is big then it contains a big clique or a large grid minor. In the former case, the idea is that we can change every solution such that it does not use all the vertices of the clique. Then we delete irrelevant vertices of the clique: those vertices which are not part of a solution. If the graph does not have a big clique minor, it has a big grid minor (as the tree width is large) [136, 137]. If any solution uses a middle vertex of this grid, it is possible to reroute the solution to avoid the middle vertex, otherwise it is possible to show that there is a big clique minor in the graph. This way we eliminate a vertex per iteration. Hence, after a while the tree-width of the graph is bounded by a function of the number of paths and we are in the first case.

As we already mentioned, in directed graphs the situation is not as desirable as for undirected graphs. The problem is hard already for 2 pairs of paths. In directed graphs, we cannot use irrelevant vertex technique directly. The main difference is that we do not have a duality between the tree-width and a bidirected grid. Instead, we have a duality between the directed tree-width and the cylindrical grid. It is not clear how to reroute paths along the cylindrical grid. Slivkins [145] proved W[1]-hardness of the problem already on acyclic graphs. However, we can simply provide an \( |G|^{O(k)} \) algorithm for \( k \)-disjoint paths problem on acyclic graphs as follows.

Instead of solving the disjoint paths problem between \( k \) source and terminal pairs given as input we solve a more general problem. We find disjoint paths for every \( t \)-tuple of
source-terminal pairs for $t \in [k]$. At first, topologically sort the vertices of the graph and divide them into two parts $A, B$ such that every vertex in $A$ is smaller than every vertex in $B$ and such that $||A| - |B|| \leq 1$. Solve the generalized disjoint paths problem in each part recursively and store the results. Merge the results in the following way to solve the generalized disjoint paths problem in $G$. To merge those two sets we carry out the following case distinction. Suppose we want a solution for a particular $t$-tuple $\tau$ of source-terminal pairs. If all of sources of $\tau$ are in $A$ and all of its terminals are in $B$, then if there is a solution for $\tau$, there are some border vertices $\sigma_A$ in $A$ and some border vertices $\sigma_B$ in $B$ and a perfect matching $M$ from $\sigma_A$ to $\sigma_B$ such that there is a set of disjoint paths $P_A$ from sources of $\tau$ to $\sigma_A$ and also there is a set of disjoint paths $P_B$ from $\sigma_B$ to the terminals of $\tau$ such that the union of $P_B, P_A, M$ provides a solution for $\tau$ in $G$. With the solutions inside $A$ and inside $B$ available, it remains to consider all $|G|^{O(k)}$ possible matchings between the parts and merge the solutions if possible. All in all this leads to an algorithm with time complexity roughly $T(|G|) = 2T(|G|/2) + |G|^{O(k)}$ which is a $|G|^{O(k)}$ time algorithm. A more complex algorithm for acyclic graphs was provided in [66], its running time is $O(|G|^k)$.

Finding the largest class of digraphs that admits a parameterized algorithm for the disjoint paths problem is one of the main challenges in this area of research over recent decades. Towards this direction, Schrijver proved one of the foremost results [141]. He showed that the directed $k$-disjoint paths problem admits an XP-algorithm in planar digraphs. More precisely there is an algorithm running in time $|G|^{f(k)}$ which solves the disjoint paths problem on planar digraphs. His algorithm is mainly of theoretical interest as its running time makes it impractical already for few source and terminal pairs even in a small digraph. After a long time, Cygan et al. [47] achieved a major breakthrough and provided an FPT algorithm for the disjoint paths problem in planar digraphs.

Despite the theoretical importance of the latter result, it is not a realistic practical algorithm: neither it is a straightforward algorithm to implement nor it has a good running time. In fact, the running time of the algorithm is in $2^{O(2^{k^2})} \text{Poly}(|G|)$. The hidden constant in $O$ notation is huge and it is practically impossible to get a result already for $k = 2$ in the worst case scenario with today’s hardware. Even the polynomial dependency on the size of the graph is big as it uses parts of Schrijver’s algorithm as a subroutine. Prior to this work, Adler et al. [2] showed that in planar undirected graphs, any algorithm that uses the irrelevant vertex technique is at least double exponential. So one cannot hope for better FPT algorithm based on this technique on directed graphs. In this thesis we aim to achieve faster algorithms, hence we do not use the irrelevant vertex technique. We introduce two new algorithmic techniques: some new combinatorial arguments and a new rerouting approaches. However, to be able to use these techniques we restrict our graph classes even more.

Although the disjoint paths problem in directed graphs is hard, if we parametrize it with the length of the paths and the number of terminal pairs, it will be fixed parameter tractable [73]. This can be obtained by utilizing the color-coding technique [6].

We can relax the problem and allow a certain amount $c$ of congestion on the vertices (edges) of the graph. This means that every vertex /edge can route up to $c$ paths, so the paths are not totally disjoint. We call this problem the congestion $c$ routing problem. The vertex disjoint paths problem is equivalent to the congestion 1 routing problem, on the other hand if we allow congestion $k$, where $k$ is the number of paths, then the problem is equivalent to finding a path for each source and terminal pair. Routing with congestion is very well
5. On Disjoint Paths Problem in Digraphs

studied in the literature [35–37, 89].

There are some other active directions of researches on disjoint paths. Another closely related variation of the problem is the partial disjoint paths problem: a certain set of pairs of paths are allowed to have an intersection. Optimization versions of the problem are also interesting. One can assign edge weights and ask for the solution with minimum total edge weights or a solution which minimizes the maximum path. These optimization versions can be solved in some restricted cases: in a special case of planar graphs, more precisely when there are some constrains on positioning of sources and terminals [51, 94] or 2-disjoint paths problem and its variations [23, 83, 104, 148]. Last but not least, we can consider different variations of congestion $c$ routing in different graph classes.

Problems Under Our Consideration

First we study the disjoint paths problem on a smaller class of planar digraphs, the class of upward planar digraphs. These are graphs that have a plane embedding such that every directed edge points “upward”, i.e. each directed edge is represented by a curve that is monotone increasing in the $y$ direction. Upward planar digraphs are very well studied in a variety of settings, in particular in graph drawing applications (see e.g. [53]). Finding a planar embedding for a planar graph is solvable in linear time, however, the problem of finding an upward planar embedding is NP-complete in general [70]. Much work has gone into finding even more restricted classes inside the upward planar class that allow to find such embeddings in polynomial time [20, 71].

First we show that the problem remains NP-complete on upward planar graphs. Our construction, with a bit of changes, even shows that the problem is NP-complete on directed grids, even if every vertex in the graph belongs to one of the paths.

Then we show the problem is fixed-parameter tractable with respect to parameter $k$ if we are given an upward planar graph together with an upward planar embedding. This is not surprising by having the more general result for planar graphs [47], but the interesting part is that our algorithm runs in linear time with a single exponential parameter dependency. The idea of the algorithm is simple but the proof of its correctness is not trivial. On the other hand, it is very easy to implement it and the hidden constant in the running time is small. Although the author did not implement it, the worst case scenario that leads to an $O(k!)$ running time is very pessimistic and we expect that the algorithm works for a reasonable number of sources and terminals in practice.

Next, we are interested in exact solutions for high congestion routing on acyclic digraphs. It is not very difficult to show that, for every fixed $c \geq 1$, we can generalise the $n^{O(1)}$ time algorithm of Fortune et al. [65] to $(k,c)$-CONGESTION ROUTING. By modifying the W[1]-hardness proof of Slivkins [146] and making appropriate modifications, we can show the problem is W[1]-hard for every fixed congestion $c \geq 1$. Moreover, by performing the proof in a more modern way (reducing from general subgraph isomorphism instead of maximum clique and invoking a lower bound of Marx [113]), we can show that the $n^{O(k)}$ time algorithm is essentially best possible with respect to the exponent of $n$. This lower bound holds under the Exponential-Time Hypothesis (ETH) see [46, 82, 106] for more background.

On the other hand, intuitively, one can expect the problem to become simpler if $c$ is almost as large as $k$: the problem is trivial if $c \geq k$. Therefore, we study the complexity of the problem in settings close to this extreme case. We provide an algorithm that for $d \geq 1$,
solves the \((k, k - d)\)-Congestion Routing in time \(\|G\|^{O(d)}\) on acyclic digraphs. That is, the exponent of the polynomial bounding the running time of the algorithm only depends on \(d\) (difference). In fact, unlike available algorithms for disjoint paths problem on the DAGs, it does not depend on the number of paths \(k\).

Another closely related problem is the \(s,t\)-induced path problem. A (directed) path \(P\) is an induced path in a graph \(G\) if \(G[P] = P\). In the \(s,t\) induced path problem the input is a directed graph \(G\) and the task is to find an induced path \(P\) starting at \(s\) and ending at \(t\). First, we take a look at the optimization version of the problem. We suppose the edges of the input graph are equipped with a weight function. We show the hardness of approximating shortest induced path problem on DAGs. Then we modify a proof and show that the induced path problem is already NP-complete even if the underlying graph has directed tree width 1. This is interesting as the problem is trivial on DAGs: any shortest path (w.r.t. the number of edges) between \(s,t\) is also an induced path on DAGs. On the other hand finding a simple path between two vertices in a graph \(G\) has a simple \(O(\|G\|)\) algorithm.

We provide a formal definition of all of abovementioned problems in the following section.

5.2. Preliminaries

An embedding of a graph \(G = (V,E)\) in the real plane is a mapping \(\varphi\) that maps vertices \(v \in V\) to points \(\varphi_v \in \mathbb{R}^2\) and edges \(e = (u,v) \in A\) to continuous functions \(\varphi_e : [0,1] \to \mathbb{R}^2\) such that \(\varphi_e(0) = \varphi_u\) and \(\varphi_e(1) = \varphi_v\). A plane embedding is an embedding such that \(\varphi_e(z) = \varphi_{e'}(z')\) if \(z, z' \in \{0,1\}\) for all \(e \neq e' \in E\). An upward plane embedding is a plane embedding such that \(\varphi_e(z) = (x',y')\) and \(z' > z\), then \(y' \geq y\). An upward planar graph is a graph that has an upward plane embedding. To improve readability, we will draw all graphs in this paper from left to right, instead of upwards.

The \(k\)-vertex disjoint paths problem on upward planar graphs is the following problem.

**Definition 5.2.1** (Vertex Disjoint Paths on Upward Planar Graphs (UpPLAN-VDPP)).

1. An upward planar graph \(G\) together with an upward plane embedding, \((s_1,t_1),\ldots,(s_k,t_k)\) are given as input,

2. Decide whether there are \(k\) pairwise internally vertex disjoint paths \(P_1,\ldots,P_k\) linking \(s_i\) to \(t_i\), for \(i \in [k]\).

In the next sections first we show that UpPLAN-VDPP is NP-complete.

**Theorem 5.2.1.** UpPLAN-VDPP is NP-complete.

Then We provide a linear FPT algorithm for UpPLAN-VDPP.

**Theorem 5.2.2.** The problem UpPLAN-VDPP can be solved in time \(O(k! \cdot k \cdot n)\), where \(n := |V(G)|\).

The disjoint paths problem with congestion is defined as follows.
Definition 5.2.2 (Disjoint Paths with Congestion).

1. Let \( G \) be a digraph and let \( I := \{(s_1,t_1), \ldots, (s_k,t_k)\} \) be a set of pairs of vertices. Let \( c \geq 1 \). A \( c \)-routing of \( I \) is a set \( \{P_1, \ldots, P_k\} \) of paths such that, for all \( 1 \leq i \leq k \), path \( P_i \) links \( s_i \) to \( t_i \) and no vertex \( v \in V(G) \) appears in more than \( c \) paths from \( \{P_1, \ldots, P_k\} \).

2. Let \( k, c \geq 1 \). In the \((k,c)\)-Congestion Routing problem, a digraph \( G \) is given in the input together with a set \( I := \{(s_1,t_1), \ldots, (s_k,t_k)\} \) of \( k \) pairs of vertices (the demands); the task is to decide whether there is a \( c \)-routing of \( I \) in \( G \).

In the next sections we consider \((k,c)\)-Congestion Routing on acyclic digraphs and we show two following theorems. Note that we just provide a rough explanation for the proof of the first theorem (as the main idea and proof was not by the author of thesis), but we provide the detailed proof for the theorem Theorem 5.2.3.

Theorem 5.2.3. For any fixed integer \( c \geq 1 \), \((k,c)\)-Congestion Routing is \( \mathsf{W[1]} \)-hard parametrized by \( k \) and, assuming ETH, cannot be solved in time \( f(k)n^{o(k/\log k)} \) for any computable function \( f \).

Theorem 5.6.2. For any fixed integer \( c \geq 1 \), \((k,c)\)-Congestion Routing is \( \mathsf{W[1]} \)-hard parametrized by \( k \) and, assuming ETH, cannot be solved in time \( f(k)n^{o(k/\log k)} \) for any computable function \( f \).

Then we study the \((s,t)\)-induced path problem in directed graphs. We first consider the weighted version of the problem.

Definition 5.2.3. Length Bounded-\((s,t)\)-Induced Path Problem (LBISTP)

\textbf{Input:} A directed graph \( D \), \( s,t \in V(D) \), \( w : E(D) \to \mathbb{N} \), \( k \in \mathbb{N} \setminus \{0\} \).

\textbf{Question:} Is there an induced path \( P \) from \( s \) to \( t \) in \( D \) with \( w(P) := \sum_{e \in E(P)} w(e) \leq k \) in \( D \)?

Respectively we prove the following lemma.

Theorem 5.2.4. LBISTP is \( \mathsf{NP} \)-complete in strong sense even on weighted DAGs.

Then based on the previous lemma we prove the following theorem.

Theorem 5.2.5. Finding an induced \((s,t)\)-path in a class of directed graphs with feedback vertex set of size 1 is \( \mathsf{NP} \)-complete.

5.3. \( \mathsf{NP} \)-Completeness of UpPlan-VDPP

This section is dedicated to the proof of the following theorem:

Theorem 5.3.1. UpPlan-VDPP is \( \mathsf{NP} \)-complete.

\textbf{Proof Sketch:} The proof of \( \mathsf{NP} \)-completeness is by a reduction from SAT, the satisfiability problem for propositional logic, which is well-known to be \( \mathsf{NP} \)-complete [69]. On a high level, our proof method is inspired by the \( \mathsf{NP} \)-completeness proof in [110] but the fact that we are
working in a restricted class of planar graphs requires a number of changes and additional
gadgets.

Let \( V = \{V_1, \ldots, V_n\} \) be a set of variables and \( C = \{C_1, \ldots, C_m\} \) be a set of clauses over the
variables from \( V \). For \( 1 \leq i \leq m \) let \( C_i = \{L_{i,1}, L_{i,2}, \ldots, L_{i,n_i}\} \) where each \( L_{i,t} \) is a literal, i.e. a
variable or the negation thereof. We will construct an upward planar graph \( G_C = (V, E) \) together with a set of
pairs of vertices in \( G_C \) such that \( G_C \) contains a set of pairwise vertex
disjoint directed paths connecting each source to its corresponding target if, and only if, \( C \) is
satisfiable. The graph \( G_C \) is roughly sketched in Figure 5.1.

We will have the source/target pairs \((V_i, V_i') \in V^2 \) for \( i \in [n] \) and \((C_j, C_j') \in V^2 \) for \( j \in [m] \),
as well as some other source/target pairs inside the gadgets \( G_{i,j,t} \) that guarantee further
properties. As the picture suggests, there will be two possible paths from \( V_i \) to \( V_i' \), an upper
path and a lower path and our construction will ensure that these paths cannot interleave.
Any interpretation of the variable \( V_i \) will thus correspond to the choice of a unique path
from \( V_i \) to \( V_i' \). Furthermore, we will ensure that there is a path from \( C_j \) to \( C_j' \) if and only if
some literal is interpreted such that \( C_j \) is satisfied under this interpretation.

We need some additional gadgets which we describe first to simplify the presentation of
the main proof.

**Gadgets, Construction of Graph Instance and Hardness Proof**

**Routing Gadget:** The rôle of a routing gadget is to act as a planar routing device. It has
two incoming connections, the edges \( e_t \) from the top and \( e_l \) from the left, and two outgoing
connections, the edges \( e_b \) to the bottom and \( e_r \) to the right. The gadget is constructed in a
way that in any solution to the disjoint paths problem it allows for only two ways of routing
a path through the gadget, either using \( e_t \) and \( e_b \) or \( e_l \) and \( e_r \).

Formally, the gadget is defined as the graph displayed in Figure 5.2 with source/target pairs
\((i, i') \) for \( i \in [4] \). Immediately from the construction of the gadget we get the following lemma
which captures the properties of routing gadgets needed in the sequel.

**Lemma 5.3.1.** Let \( R \) be a routing gadget with source/target pairs \((i, i') \) for \( i \in [4] \).
5. On Disjoint Paths Problem in Digraphs

Figure 5.2.: The routing gadget. In the following, when a routing gadget appears as a subgadget in a figure, it will be represented by a black box as shown on the left.

1. There is a solution of the disjoint paths problem in \( R \).

2. Let \( P_1, \ldots, P_4 \) be any solution to the disjoint paths problem in \( R \), where \( P_i \) links vertex \( i \) to \( i \). Let \( H := R \setminus \bigcup_{i=1}^{4} P_i \).
   a) \( H \) neither contains a path which goes through \( e_t \) to \( e_r \) nor a path which goes through \( e_l \) to \( e_b \).
   b) \( H \) either contains a unique path \( P \) which goes through \( e_t \) to \( e_b \) or a unique path \( P' \) in \( H \) which goes through \( e_l \) to \( e_r \), and not both.

Crossing Gadget: A crossing gadget has two incoming connections to its left via the vertices \( H^{in} \) and \( L^{in} \) and two outgoing connections to its right via the vertices \( H^{out} \) and \( L^{out} \). Furthermore, it has one incoming connection at the top via the vertex \( T \) and outgoing connection at the bottom via the vertex \( B \). Intuitively, we want that in any solution to the disjoint paths problem, there is exactly one path \( P \) going from left to right and exactly one path \( P' \) going from top to bottom. Furthermore, if \( P \) enters the gadget via \( H^{in} \) then it should leave it via \( H^{out} \) and if it enters the gadget via \( L^{in} \) then it should leave it via \( L^{out} \). Of course, in a planar graph there cannot be such disjoint paths \( P, P' \) as they must cross at some point. We will have to split one of the paths, say \( P \), by removing the outward source/sink pair and introducing two new source/sink pairs, one to the left of \( P' \) and one to its right.

Formally, the gadget is defined as the graph displayed in Figure 5.3. The following lemma follows easily from Lemma 5.3.1.

Lemma 5.3.2. Let \( G \) be a crossing gadget.

1. There are uniquely determined vertex disjoint paths \( P_1 \) from \( H^{in} \) to \( W \), \( P_2 \) from \( T \) to \( B \) and \( P_3 \) from \( X \) to \( Y \). Let \( H := G \setminus \bigcup_{i=1}^{3} P_i \). Then \( H \) contains a path from \( Z \) to \( H^{out} \) but it does not contain a path from \( Z \) to \( L^{out} \).
5. On Disjoint Paths Problem in Digraphs

2. There are uniquely determined vertex disjoint paths $Q_1$ from $L^{in}$ to $W$, $Q_2$ from $T$ to $B$ and $Q_3$ from $X$ to $Y$. Let $H := G \setminus \bigcup_{i=1}^3 Q_i$. Then $H$ contains a path from $Z$ to $L^{out}$ but it does not contain a path from $Z$ to $H^{out}$.

The next lemma shows that we can connect crossing gadgets in rows in a useful way. It follows easily by induction from Lemma 5.3.2.

Let $G_1, \ldots, G_s$ be a sequence of crossing gadgets drawn from left to right in that order. We address the inner vertices of the gadgets by their names in the gadget equipped with corresponding subscripts, e.g., we write $H^{in}_{1}$ for the vertex $H^{in}$ of gadget $G_1$. For each $j \in [s-1]$, we add the edges $(H^{out}_{j}, H^{in}_{j+1})$ and $(L^{out}_{j}, L^{in}_{j+1})$ and call the resulting graph a row of crossing gadgets. We equip this graph with the source/target pairs $(X_j, Y_j)$, $(Z_j, W_{j+1})$ for $j \in [s-1]$ to obtain an associated vertex disjoint paths problem $P_r$ (the subscript $r$ stands for row). Denote by $P^+_r$ the problem $P_r$ with additional source/target pair $(H^{in}_{1}, W_{1})$ and by $P^-_r$ the problem $P_r$ with additional source/target pair $(L^{in}_{1}, W_{1})$.

Lemma 5.3.3. Let $G$ be a row of crossing gadgets. Then both associated vertex disjoint paths problems $P^+_r$, $P^-_r$ have unique solutions. For all $i \in [t-1]$, each path in the solution of $P^+_r$ from $Z_i$ to $W_{i+1}$ passes through $H^{in}_{i+1}$ and each path in the solution of $P^-_r$ from $Z_i$ to $W_{i+1}$ passes through $L^{in}_{i+1}$.

The next lemma shows that we can force a relation between rows and columns of crossing gadgets.

Let $G_1, \ldots, G_t$ be a sequence of crossing gadgets drawn from top to bottom in that order. For each $i \in [t-1]$, we add the edge $(B_i, T_{i+1})$ and call the resulting graph a column of crossing gadgets. We equip this graph with the source/target pairs $(X_i, Y_i)$ for $i \in [t]$ and with the pair $(T_1, B_t)$ to obtain an associated vertex disjoint paths problem $P$.
Lemma 5.3.4. Let $G$ be a column of crossing gadgets. Let $P_1, \ldots, P_t$ be a sequence of vertex disjoint paths such that for $i \in [t]$, $P_i$ connects either $H_{i}^{in}$ or $L_{i}^{in}$ to $W_i$. Let $H := G \cup \bigcup_{i=1}^{t} P_i$.

1. The vertex disjoint paths problem $\mathcal{P}$ on $H$ has a solution.

2. There is a unique path $Q$ connecting $T_i$ to $B_i$ which for all $i \in [t]$ uses edge $e^+$ in $G_i$ if and only if $P_i$ starts at $H_{i}^{in}$ and the edge $e^-$ in $G_i$ if and only if $P_i$ starts at $L_{i}^{in}$.

Note that the paths $P_i$ as stated in the lemma exist and they are uniquely determined by Lemma 5.3.2.

We are now ready to construct a vertex disjoint paths instance for any SAT instance $C$.

Definition 5.3.1. Let $C$ be a SAT instance over the variables $\mathcal{V} = \{V_1, \ldots, V_n\}$ and let $\{C_1, \ldots, C_m\}$ be its set of clauses. For $j \in [m]$ let $C_j = \{L_{j,1}, L_{j,2}, \ldots, L_{j,n_j}\}$, where each $L_{j,s}$ is a literal, i.e., a variable or the negation thereof.

1. The graph $G_C$ is defined as follows.
   - For each variable $V \in \mathcal{V}$ we introduce two vertices $V$ and $V'$.
   - For each clause $C \in \mathcal{C}$ we introduce two vertices $C$ and $C'$.
   - For each variable $V_i$ and each literal $L_{i,j,t}$ in clause $j$ we introduce a crossing gadget $G_{i,j,t}$.
     - For $i \in [n]$ we add the edges $(V_i, H_{i,1,1})$, $(V_i, L_{i,1,1})$, $(H_{i,m,n_m}^{out}, V_j')$ and $(L_{i,m,n_m}^{out}, V_j')$.
     - For $j \in [m], t \in [n_j]$ we add the edges $(C_j, T_{1,j,t})$ and $(B_{n,j,t}, C_j')$.
     - Finally, we delete the edge $e^+$ for all $i \in [n], j \in [m], t \in [n_j]$ in $G_{i,j,t}$ if $L_{j,t}$ is a variable and the edge $e^-$ if it is a negated variable.

   We draw the graph $G_C$ as shown in Figure 5.1.

2. We define the following vertex disjoint paths problem $\mathcal{P}_C$ on $G_C$. We add all source/target pairs that are defined inside the routing gadgets. Furthermore:
   - For $i \in [n], j \in [m], t \in [n_j-1]$, we add the pairs
     - $(V_i, W_{i,1,1})$,
     - $(Z_{i,m,n_m}, V_j')$,
     - $(X_{i,j,t}, Y_{i,j,t})$ and
     - $(Z_{i,j,t}, W_{i,j,t+1})$.
   - For $i \in [n], j \in [m-1]$, we add the pairs $(Z_{i,j,n_j}, W_{i,j+1,1})$.
   - For $j \in [m]$, we add the pairs $(C_j, C_j')$.

The proof of the following theorem is based on the fact that in our construction, edge $e^+$ is present in gadget $G_{i,j,t}$, if and only if $C_j$ does not contain variable $V_i$ negatively and $e^-$ is present in gadget $G_{i,j,t}$, if and only if $C_j$ does not contain variable $V_i$ positively (especially, both edges are present if the clause does not contain the variable at all). In particular, every column contains exactly one gadget where one edge is missing. Now it is easy to conclude with Lemmas 5.3.3 and 5.3.4.
Theorem 5.3.1. Let \( \mathcal{C} \) be a SAT-instance and let \( \mathcal{P}_C \) be the corresponding vertex disjoint paths instance on \( G_C \) as defined in Definition 5.3.1. Then \( \mathcal{C} \) is satisfiable if and only if \( \mathcal{P}_C \) has a solution.

It is easily seen that the presented reduction can be computed in polynomial time and this finishes the proof of Theorem 5.3.1.

If we replace the vertices \( C_i \) and \( C'_i \) with directed paths, then it is easy to convert the graph \( G_C \) to a directed grid graph, i.e. a subgraph of the infinite grid. This implies that the problem is NP-complete even on upward planar graphs of maximum degree 4.

5.4. A Linear Time Algorithm for UpPlan-VDPP for Fixed \( k \)

In this section we prove that the \( k \)-disjoint paths problem for upward planar digraphs can be solved in linear time for any fixed value of \( k \). In other words, the problem is fixed-parameter tractable by a linear time parametrized algorithm.

Theorem 5.2.2. The problem UpPlan-VDPP can be solved in time \( O(k! \cdot k \cdot n) \), where \( n := |V(G)| \).

Unlike most other parts of the this work we do not provide a proof sketch, as the algorithm is easy to understand even though the proof is not trivial.

For the rest of the section we fix a planar upward graph \( G \) together with an upward planar embedding and \( k \) pairs \((s_1,t_1),\ldots,(s_k,t_k)\) of vertices. We will not distinguish notationally between \( G \) and its upward planar embedding. Whenever we speak about a vertex \( v \) on a path \( P \) we mean a vertex \( v \in V(G) \) which is contained in \( P \). If we speak about a point on the path we mean a point \((x,y)\in \mathbb{R}^2\) which is contained in the drawing of \( P \) with respect to the upward planar drawing of \( G \). The algorithm is based on the concept of a path in \( G \) being to the right of another path which we define next.

Definition 5.4.1. Let \( P \) be a path in an upward planar drawing of \( G \). Let \((x,y)\) and \((x',y')\) be the two endpoints of \( P \) such that \( y \leq y' \), i.e. \( P \) starts at \((x,y)\) and ends at \((x',y')\). We define

\[
\text{right}(P) := \{(u,v) \in \mathbb{R}^2 : y \leq v \leq y' \text{ and } u' < u \text{ for all } u' \text{ such that } (u',v) \in P\}
\]

\[
\text{left}(P) := \{(u,v) \in \mathbb{R}^2 : y \leq v \leq y' \text{ and } u' > u \text{ for all } u' \text{ such that } (u',v) \in P\}.
\]

The next two lemmas follow immediately from the definition of upward planar drawings.

Lemma 5.4.1. Let \( P \) and \( Q \) be vertex disjoint paths in an upward planar drawing of \( G \). Then either \( \text{right}(P) \cap Q = \emptyset \) or \( \text{left}(P) \cap Q = \emptyset \).

Lemma 5.4.2. Let \( P \) be a directed path in an upward planar drawing of a digraph \( G \). For \( i = 1,2,3 \) let \( p_i := (x_i,y_i) \) be distinct points in \( P \) such that \( y_1 < y_2 < y_3 \). Then \( p_1,p_2,p_3 \) occur in this order on \( P \).

Definition 5.4.2. Let \( P \) and \( Q \) be two vertex disjoint paths in \( G \).

1. A point \( p = (x,y) \in \mathbb{R}^2 \setminus P \) is to the right of \( P \) if \( p \in \text{right}(P) \). Analogously, we say that \( (x,y) \in \mathbb{R}^2 \setminus P \) is to the left of \( P \) if \( p \in \text{left}(P) \).
2. The path \( P \) is to the right of \( Q \), denoted by \( Q \prec P \) if there exists a point \( p \in P \) which to the right of some point \( q \in Q \). We write \( \prec^* \) for the transitive closure of \( \prec \).

3. If \( \mathcal{P} \) is a set of pairwise disjoint paths in \( G \), we write \( \prec_{\mathcal{P}} \) and \( \prec_{\mathcal{P}}^* \) for the restriction of \( \prec \) and \( \prec^* \), resp., to the paths in \( \mathcal{P} \).

We show next that for every set \( \mathcal{P} \) of pairwise vertex disjoint paths in \( G \) the relation \( \prec^* \) is a partial order on \( \mathcal{P} \). Towards this aim, we first show that \( \prec \) is irreflexive and anti-symmetric on \( \mathcal{P} \).

**Lemma 5.4.3.** Let \( \mathcal{P} \) be a set of pairwise disjoint paths in \( G \).

1. The relation \( \prec_{\mathcal{P}} \) is irreflexive.

2. The relation \( \prec_{\mathcal{P}} \) is anti-symmetric, i.e. if \( P_1 \prec_{\mathcal{P}} P_2 \) then \( P_2 \not\prec_{\mathcal{P}} P_1 \) for any \( P_1, P_2 \in \mathcal{P} \).

**Proof.** The first claim immediately follows from the definition of \( \prec \). Towards the second statement, suppose there are \( P_1, P_2 \in \mathcal{P} \) such that \( P_1 \prec_{\mathcal{P}} P_2 \) and \( P_2 \prec_{\mathcal{P}} P_1 \).

Hence, for \( j = 1, 2 \) and \( i = 1, 2 \) there are points \( p^i_j = (x^i_j, y^i_j) \) such that \( p^i_j \in P_i \) and \( x^1_j < x^2_j, y^1_j = y^2_j \) and \( x^2_j > x^1_j, y^2_j = y^1_j \). W.l.o.g. we assume that \( y^1_j < y^2_j \). Let \( Q \subseteq P \) be the subpath of \( P \) from \( p^1_j \) to \( p^2_j \), including the endpoints. Let \( Q_1 := \{(x^1_j, z) : z < y^1_j\} \) and \( Q_2 := \{(x^2_j, z) : z > y^2_j\} \) be the two lines parallel to the y-axis going from \( p^1_j \) towards negative infinity and from \( p^2_j \) towards infinity. Then \( Q_1 \cup Q_2 \) separates the plane into two disjoint regions \( R_1 \) and \( R_2 \) each containing a point of \( P_2 \). As \( P_1 \) and \( P_2 \) are vertex disjoint but \( p^1_j \) and \( p^2_j \) are connected by \( P_2 \), \( P_2 \) must contain a point in \( Q_1 \) or \( Q_2 \) which, on \( P_2 \) lies between \( p^2_j \) and \( p^2_j \). But the y-coordinate of any point in \( Q_1 \) is strictly smaller than \( y^1_j \) and \( y^2_j \) whereas the y-coordinate of any point in \( Q_2 \) is strictly bigger than \( y^1_j \) and \( y^2_j \). This contradicts Lemma 5.4.2. \( \qed \)

We use the previous lemma to show that \( \prec_{\mathcal{P}}^* \) is a partial order for all sets \( \mathcal{P} \) of pairwise vertex disjoint paths.

**Lemma 5.4.4.** Let \( \mathcal{P} \) be a set of pairwise vertex disjoint directed paths. Then \( \prec_{\mathcal{P}}^* \) is a partial order.

**Proof.** By definition, \( \prec_{\mathcal{P}}^* \) is transitive. Hence we only need to show that it is anti-symmetric for which, by transitivity, it suffices to show that \( \prec_{\mathcal{P}}^* \) is irreflexive.

To show that \( \prec_{\mathcal{P}}^* \) is irreflexive, we prove by induction on \( k \) that if \( P_0, \ldots, P_k \in \mathcal{P} \) are paths such that \( P_0 \prec_{\mathcal{P}} \cdots \prec_{\mathcal{P}} P_k \) then \( P_k \not\prec_{\mathcal{P}} P_0 \). As for all \( P \in \mathcal{P} \), \( P \not\prec_{\mathcal{P}} P \), this proves the lemma.

Towards a contradiction, suppose the claim was false and let \( k \) be minimum such that there are paths \( P_0, \ldots, P_k \in \mathcal{P} \) with \( P_0 \prec_{\mathcal{P}} \cdots \prec_{\mathcal{P}} P_k \) and \( P_k \prec_{\mathcal{P}} P_0 \). By Lemma 5.4.3, \( k > 1 \).

Let \( R \) be the set of right ends of \( P_i \). Note that \( k \) is even, so \( R \) is not empty. Furthermore, as for all \( P, Q \) with \( P \prec_{\mathcal{P}} Q \), \( \text{right}(P) \cap \text{right}(Q) \neq \emptyset \), \( R \) is a connected region in \( \mathbb{R}^2 \) without holes. Let \( L := \bigcup_{i=1}^{k-1} \text{left}(P_i) \). Again, as \( k > 1 \), \( L \neq \emptyset \) and \( L \) is a connected region without holes.

As \( P_{k-2} \prec_{\mathcal{P}} P_{k-1} \), we have \( L \cap R \neq \emptyset \) and therefore \( L \cup R \) separates the plane into two unbounded regions, the upper region \( T \) and the lower region \( B \).
5. On Disjoint Paths Problem in Digraphs

The minimality of $k$ implies that $P_i \not\subset P_k$ for all $i < k - 1$ and therefore $R \cap P_k = \emptyset$. Analogously, as $P_k \not\subset P_i$ for any $i > 0$, we have $L \cap P_k = \emptyset$. Hence, either $P_k \subseteq B$ or $P_k \subseteq T$. W.l.o.g. we assume $P_k \subseteq B$. We will show that $left(P_0) \cap B = \emptyset$.

Suppose there was a point $(x, y) \in P$ and some $x' < x$ such that $(x', y) \in B$. This implies that $y < v$ for all $(u, v) \in L$. But this implies that $B$ is bounded by $right(P_0)$ and $L$ contradicting the fact that $right(P_{k-1}) \cap B \neq \emptyset$.

We have shown so far that $\prec^*$ is a partial order on every set of pairwise vertex disjoint paths.

**Remark 5.4.1.** Note that if two paths $P,Q \in \mathcal{P}$ are incomparable with respect to $\prec^*_P$, then one path is strictly above the other, i.e. $(right(P) \cup left(P)) \cap (right(Q) \cup left(Q)) = \emptyset$. This is used in the next lemma.

**Definition 5.4.3.** Let $s,t \in V(G)$ be vertices in $G$ such that there is a directed path from $s$ to $t$. The right-most $s$-$t$-path in $G$ is an $s$-$t$-path $P$ such that for all $s$-$t$-paths $P'$, $P \subseteq P' \cup right(P')$.

**Lemma 5.4.5.** Let $s,t \in V(G)$ be two vertices and let $P$ be a path from $s$ to $t$ in an upward planar drawing of $G$. If $P'$ is an $s$-$t$- path such that $P' \cap right(P) \neq \emptyset$ then there is an $s$-$t$- path $Q$ such that $Q \subseteq P \cup right(P)$ and $Q \cap right(P) \neq \emptyset$.

**Proof.** If $P' \subseteq P \cup right(P)$ we can take $Q = P'$. Otherwise, i.e. if $P' \cap left(P) \neq \emptyset$, then as the graph is planar this means that $P$ and $P'$ share internal vertices. In this case we can construct $Q$ from $P \cup P'$ where for subpaths of $P$ and $P'$ between two vertices in $P \cap P'$ we always take the subpath to the right.

**Corollary 5.4.1.** Let $s,t \in V(G)$ be vertices in $G$ such that there is a directed path from $s$ to $t$. Then there is a unique right-most $s$-$t$-path in $G$.

The corollary states that between any two $s$ and $t$, if there is an $s$-$t$ path then there is a rightmost one. The proof of Lemma 5.4.5 also indicates how such a path can be computed. This is formalised in the next lemma.

**Lemma 5.4.6.** There is a linear time algorithm which, given an upward planar drawing of a graph $G$ and two vertices $s,t \in V(G)$ computes the right-most $s$-$t$-path in $G$, if such a path exists.

**Proof.** We first use a depth-first search starting at $s$ to compute the set of vertices $U \subseteq V(G)$ reachable from $s$. Clearly, if $t \not\in U$ then there is no $s$-$t$-path and we can stop. Otherwise we use a second, inverse depth-first search to compute the set $U' \subseteq U$ of vertices from which $t$ can be reached. Finally, we compute the right-most $s$-$t$ path inductively by starting at $s \in U'$ and always choosing the right-most successor of the current vertex until we reach $t$. The right-most successor is determined by the planar embedding of $G$. As $G$ is acyclic, this procedure produces the right-most path and can clearly be implemented in linear time.

We show next that in any solution $\mathcal{P}$ to the disjoint paths problem in an upward planar digraph, if $P \in \mathcal{P}$ is a maximal element with respect to $\prec^*_\mathcal{P}$, we can replace $P$ by the right-most $s$-$t$ path and still get a valid solution, where $s$ and $t$ are the endpoints of $P$. 88
Lemma 5.4.7. Let \( G \) be an upward planar graph with a fixed upward planar embedding and let \((s_1,t_1),\ldots,(s_k,t_k)\) be pairs of vertices. Let \( \mathcal{P} \) be a set of pairwise disjoint paths connecting \((s_i,t_i)\) for all \( i \). Let \( P \in \mathcal{P} \) be a path connecting \( s_i \) and \( t_i \), for some \( i \), which is maximal with respect to \( \prec_P \). Let \( P' \) be the right-most \( s_i-t_i \)-path in \( G \). Then \( \mathcal{P} \setminus \{ P \} \cup \{ P' \} \) is also a valid solution to the disjoint paths problem on \( G \) and \((s_1,t_1),\ldots,(s_k,t_k)\).

Proof. All we have to show is that \( P' \) is disjoint from all \( Q \in \mathcal{P} \setminus \{ P \} \). Clearly, as \( P \) and \( P' \) are both upward \( s_i-t_i \) paths, we have \( P' \subseteq P \cup \text{left}(P) \cup \text{right}(P) \).

By the remark above, if \( P \) and \( Q \) are incomparable with respect to \( \prec_P \), then one is above the other and therefore \( Q \) and \( P' \) must be disjoint. Now suppose \( Q \) and \( P \) are comparable and therefore \( Q \prec_P P \). This implies that \((P \cup \text{right}(P)) \cap Q = \emptyset\) and therefore \( Q \cap P' = \emptyset \).

The previous lemma yields the key to the proof of Theorem 5.2.2:

Proof of Theorem 5.2.2. Let \( G \) with an upward planar drawing of \( G \) and \( k \) pairs \((s_1,t_1),\ldots,(s_k,t_k)\) be given. To decide whether there is a solution to the disjoint paths problem on this instance we proceed as follows. In the first step we compute for each \( s_i \) the set of vertices reachable from \( s_i \). If for some \( i \) this does not include \( t_i \) we reject the input as obviously there cannot be any solution.

In the second step, for every possible permutation \( \pi \) of \( \{1,\ldots,k\} \) we proceed as follows. Let \( i_1 := \pi(k),\ldots, i_k := \pi(1) \) be the numbers 1 to \( k \) ordered as indicated by \( \pi \) and let \( u_j := s_{i_j} \) and \( v_j := t_{i_j} \), for all \( j \in [k] \). We can view \( \pi \) as a linear order on \( 1,\ldots,k \) and for every such \( \pi \) we will search for a solution \( \mathcal{P} \) of the disjoint paths problem for which \( \prec_{\mathcal{P}} \) is consistent with \( \pi \).

For a given \( \pi \) as above we inductively construct a sequence \( \mathcal{P}_0,\ldots,\mathcal{P}_k \) of sets of pairwise vertex disjoint paths such that for all \( i \), \( \mathcal{P}_i \) contains a set of \( i \) paths \( P_1,\ldots,P_i \) such that for all \( j \in [i] \) \( P_j \) links \( u_j \) to \( v_j \). We set \( \mathcal{P}_0 := \emptyset \) which obviously satisfies the condition. Suppose for some \( 0 \leq i < k \), \( \mathcal{P}_i \) has already been constructed. To obtain \( \mathcal{P}_{i+1} \) we compute the right-most path linking \( u_{i+1} \) to \( v_{i+1} \) in the graph \( G \setminus \bigcup \mathcal{P}_i \). By Lemma 5.4.6, this can be done in linear time for each such pair \((s_{i+1},t_{i+1})\). If there is such a path \( P \) we define \( \mathcal{P}_{i+1} := \mathcal{P}_i \cup \{ P \} \). Otherwise we reject the input. Once we reach \( \mathcal{P}_k \) we stop and output \( \mathcal{P}_k \) as solution.

Clearly, for every permutation \( \pi \) the algorithm can be implemented to run in time \( O(k \cdot n) \), using Lemma 5.4.6, so that the total running time is \( O(k! \cdot k \cdot n) \) as required.

Obviously, if the algorithm outputs a set \( \mathcal{P} \) of disjoint paths then \( \mathcal{P} \) is indeed a solution to the problem. What is left to show is that whenever there is a solution to the disjoint path problem, then the algorithm will find one.

So let \( \mathcal{P} \) be a solution, i.e. a set of \( k \) paths \( P_1,\ldots,P_k \) so that \( P_i \) links \( s_i \) to \( t_i \). Let \( \leq \) be a linear order on \( \{1,\ldots,k\} \) that extends \( \prec_{\mathcal{P}} \) and let \( \pi \) be the corresponding permutation such that \((u_1,v_1),\ldots,(u_k,v_k)\) is the ordering of \((s_1,t_1),\ldots,(s_k,t_k)\) according to \( \leq \). We claim that for this permutation \( \pi \) the algorithm will find a solution. Let \( P \) be the right-most \( u_k-v_k \)-path in \( G \) as computed by the algorithm. By Lemma 5.4.7, \( \mathcal{P} \setminus \{ P_k \} \cup \{ P \} \) is also a valid solution so we can assume that \( P_k = P \). Hence, \( P_1,\ldots,P_{k-1} \) form a solution of the disjoint paths problem for \((u_1,v_1),\ldots,(u_{k-1},v_{k-1})\) in \( G \setminus P \). By repeating this argument we get a solution \( \mathcal{P}' := \{ P'_1,\ldots,P'_k \} \) such that each \( P'_i \) links \( u_i \) to \( v_i \) and is the right-most \( u_i-v_i \)-path in \( G \setminus \bigcup_{j<i} P'_j \). But this is exactly the solution found by the algorithm. This prove the correctness of the algorithm and concludes the proof of the theorem. □
5. On Disjoint Paths Problem in Digraphs

We remark that we can easily extend this result to “almost upward planar” graphs, i.e., to graphs such that the deletion of at most \( h \) edges yields an upward planar graph. As finding an upward planar drawing of an upward planar graph is NP-complete, this might be of use if we have an approximation algorithm that produces almost upward planar embeddings.

5.5. Disjoint Paths With Congestion on DAGs

In this section we prove the Theorem 5.2.3.

**Theorem 5.2.3.** For every fixed \( d \geq 1 \), the \((k, k - d)\)-Congestion Routing problem on acyclic digraphs can be solved in time \( n^{O(d)} \).

We first need some additional notation and prove some auxiliary lemmas.

**Definition 5.5.1.** Let \( G \) be a digraph and let \( \mathcal{L} \) be a set of paths in \( G \). For every \( v \in V(G) \) we define the congestion of \( v \) with respect to \( \mathcal{L} \) as the number of paths in \( \mathcal{L} \) containing \( v \).

The following lemma provides a simple extension of the algorithm from [65] for disjoint paths in acyclic digraphs.

**Lemma 5.5.1.** On acyclic digraphs \( G \) the \((k, c)\)-Congestion Routing problem can be solved in time \( n^{O(k)} \), where \( n := |G| \).

**Proof.** In [65], Fortune et al. proved that the \( k \)-disjoint paths problem can be solved in time \( n^{O(k)} \) on any \( n \)-vertex acyclic digraph \( G \).

Let \( G \), \( (s_1, t_1), \ldots, (s_k, t_k) \) and \( c \) be given. We construct a new digraph \( H \) with \( V(H) := V(G) \times \{1, \ldots, c\} \) and \( E(H) := \{((u, i), (v, j)) : (u, v) \in E(G), 1 \leq i, j \leq c\} \).

Then \( H \) contains \( k \) pairwise vertex disjoint paths \( P_1, \ldots, P_k \) such that \( P_i \) links \( (s_i, 1) \) to \((t_i, 1)\) if, and only if, there is a positive solution to the \((k, c)\)-Congestion Routing Problem on \( G \). By the algorithm in [65] we can decide whether the paths \( P_1, \ldots, P_k \) exist in \( H \) in time \( |V(H)|^{O(k)} \) and hence in time \((c \cdot n)^{O(k)} = n^{O(k)} \) as \( c \leq n \).

We will use this lemma in the form given in the next corollary.

**Corollary 5.5.1.** For \( c, k \geq 0 \) such that \( k \in O(c) \), the \((k, c)\)-Congestion Routing problem can be solved on any acyclic \( n \)-vertex digraph \( G \) in time \( n^{O(c)} \).

The next lemma provides the main reduction argument for proving Theorem 5.2.3.

**Lemma 5.5.2.** Let \( G \) be an acyclic directed graph and let \( d \geq 1 \) and \( k > 3d \). Let \( I := \{(s_1, t_1), \ldots, (s_k, t_k)\} \subseteq V(G) \times V(G) \) be a set of source/terminal pairs. There exists a \((k-d)\)-routing of \( I \) if, and only if, for every pair \( (s, t) \in I \) there is a path in \( G \) from \( s \) to \( t \) and there is a subset \( I' \subseteq I \) of order \( |I'| = k - 1 \) such that there is a \((k-d-1)\)-routing of \( I' \).

**Proof.** The if direction is easy to see. Let \( S' := \{P_1, \ldots, P_{k-1}\} \) be a \((k-d-1)\)-routing of a set \( I' \subseteq I \) of order \( k - 1 \). Let \( s, t \) be such that \( I = I' \cup \{(s, t)\} \). By assumption there is a simple path \( P \) from \( s \) to \( t \) in \( G \). Then \( S := S' \cup \{P\} \) is a \((k-d)\)-routing of \( I \).

For the reverse direction let \( I := \{(s_1, t_1), \ldots, (s_k, t_k)\} \) and let \( \hat{S} := \{\hat{P}_1, \ldots, \hat{P}_k\} \) be a \((k-d)\)-routing of \( I \) such that \( \hat{P}_i \) links \( s_i \) to \( t_i \) for all \( 1 \leq i \leq k \). We define a multi digraph
5. On Disjoint Paths Problem in Digraphs

$G'$ on the same vertex set $V(G)$ as $G$ as follows. For every pair $u, v \in V(G')$ such that $e = (u, v) \in E(G)$ and every $1 \leq i \leq k$, if $e$ occurs on the path $P_i \in S$, then we add a new edge $e' = (u, v)$ to $G'$. Hence, if any edge $e \in E(G)$ is used by $\ell$ different paths in $\hat{S}$, then $G'$ contains $\ell$ parallel edges between the endpoints of $e$. In the rest of the proof we will work on the multi digraph $G'$. We can now take a set $S := \{P_1, \ldots, P_h\}$ of pairwise edge disjoint paths, where $P_i$ is the path from $s_i$ to $t_i$ induced by the edge set $\{e^i : e \in E(\hat{P}_i)\}$. That is, by using the parallel edges, we can turn the routing $S$ into a $(k-d)$-routing of $I$ where the paths are mutually edge disjoint.

In the remainder of the proof we will construct a subset $I' \subseteq I$ of order $k-1$ and a $(k-d-1)$-routing of $I'$ in $G'$ which is pairwise edge disjoint. This naturally induces a $(k-d-1)$-routing of $I'$ in $G$. Note that in $G'$, if $L$ is any set of pairwise edge disjoint paths, then the congestion of any vertex with respect to $L$ is at most the congestion of the vertex with respect to $S$ (and thus $\hat{S}$) in $G'$ (and $G$, respectively). Indeed, every edge in $L$ has a corresponding path in $S$, so no vertex can be contained in more paths from $L$ than in $S$.

Let $\subseteq$ be a topological ordering of $G'$ and let $A := \{a_1, \ldots, a_\ell\}$ be the set of vertices of congestion $k-d$ with respect to $S$ such that $a_i \subseteq a_j$ whenever $i < j$. As $k > 3d$, for all $1 \leq i < \ell$ there is a path in $G$ from $a_i$ to $a_{i+1}$.

For $1 \leq i \leq k$, an atomic subpath of $P_i$ (with respect to $S$) is a subpath of $P_i$ that starts and ends in a vertex of $A \cup \{s_i, t_i\}$ and is internally vertex disjoint from $A$. Hence, every path $P_i \in S$ consists of the concatenation $P^1_i \cdots P^\ell_i$ of its atomic subpaths where we identify the last vertex of $P^1_i$ with the first vertex of $P^{\ell_i+1}_i$ for all $1 \leq j < \ell_i$. Note that any two atomic subpaths of paths $P_i, P_j$ in $S$ are pairwise edge disjoint.

Let $I' \subseteq I$ be a subset of order $k-1$. A routing $S' := \{P'_1, \ldots, P'_{k-1}\}$ of $I'$ is conservative with respect to $S$ if it consists of pairwise edge disjoint paths and every path in $S'$ consists of a concatenation of atomic subpaths of paths in $S$. In the sequel, whenever we speak of a conservative $I'$-routing we implicitly mean that it is conservative with respect to $S$.

If $S'$ is a conservative $I'$-routing with respect to $S$, then it consists of pairwise edge disjoint paths and hence for every $v \in V(G)$ the congestion of $v$ with respect to $S'$ is at most the congestion of $v$ with respect to $S$.

Let $1 \leq i_1 < i_2 \leq \ell$ and let $1 \leq j \leq k$. Let $S'$ be a conservative $I'$-routing. An $(i_1, i_2)$-jump of colour $j$ is a subpath $P'$ of $P_j$ from $a_{i_1}$ to $a_{i_2}$ such that for all $i$ with $i_1 < i < i_2$ the vertex $a_i$ is not on $P_j$. Note that any jump is an atomic subpath. We call the jump $P'$ free with respect to $S'$ if $P'$ is not used by any path in $S'$.

We are now ready to complete the proof of the lemma. Note first that, as $k > 3d$, for any three vertices $b_1, b_2, b_3 \in A$ there is a path $P \in S$ that contains $b_1, b_2, b_3$. Hence, we can choose an $h \in \{1, \ldots, k\}$ such that $a_1, a_2 \in V(P_h)$ and there is a vertex $a_r$ with $1 < r < \ell$ such that $a_r \in V(P_h)$. Let $I' := I \setminus \{(s_h, t_h)\}$. If $A \subseteq V(P_h)$, then $S' \setminus \{P_h\}$ is a $(k-d-1)$-routing of $I'$ and we are done. Otherwise, for every vertex $a_r \in A$ which has congestion $k-d$ with respect to $S \setminus \{P_h\}$ there are $i, j$ with $i < r < j$ and an $(i, j)$-jump $P$ of colour $h$. This follows as $a_1, a_\ell \in V(P_h)$. Note also that $a_1$ and $a_\ell$ have congestion $k-d-1$ in $S \setminus \{P_h\}$. Note that this jump $P$ is free with respect to $S \setminus \{P_h\}$.

Thus, it is easily seen that $S \setminus \{P_h\}$ satisfies the following two properties:

1. For every vertex $a_r$ of congestion $k-d$ with respect to $S \setminus \{P_h\}$ there are indices $i < r < j$ such that there is a free $(i, j)$-jump $P$ with respect to $S \setminus \{P_h\}$. 

91
2. For any three vertices \( b_1, b_2, b_3 \) of congestion \( k - d \) with respect to \( S \) there is a path \( Q = S \setminus \{P_h\} \) with \( \{b_1, b_2, b_3\} \subseteq V(Q) \).

Now let \( S' \) be a routing of \( I' \) which satisfies Condition 1 and 2 (with respect to \( S' \) instead of \( S \setminus \{P_h\} \)) and, subject to this, the number of vertices of congestion \( k - d \) with respect to \( S' \) is minimal.

We claim that \( S' \) is a \((k - d - 1)\)-routing of \( I' \). Let \( S' := \{Q_1, \ldots, Q_{k-1}\} \). Towards a contradiction, suppose there is a vertex \( a_r \) of congestion \( k - d \) with respect to \( S' \). As \( S' \) is conservative, we have \( a_r \in A \). Hence, by assumption, there are \( i < r < j \) and a free \((i, j)\)-jump \( P \) with respect to \( S' \).

Let \( Q_h \) be a path in \( S' \) that contains \( a_i, a_r \) and \( a_j \), which exists by Condition 2. Let \( Q_h := Q_h^1 \cup Q_h^2 \cup Q_h^3 \) where

- \( Q_h^1 \) is the initial subpath of \( Q_h \) from its first vertex to \( a_i \),
- \( Q_h^2 \) is the subpath starting at \( a_i \) and ending in \( a_j \) and
- \( Q_h^3 \) is the subpath starting in \( a_j \) and ending at the end of \( Q_h \).

We define \( Q'_h := Q_h^1 \cup P \cup Q_h^3 \), i.e. \( Q'_h \) is the path obtained from \( Q_h \) by replacing the inner subpath \( Q_h^2 \) by the \((i, j)\)-jump \( P \). Let \( \mathcal{L} := (S' \setminus \{Q_h\}) \cup \{Q_h'\} \). Then \( \mathcal{L} \) is a routing of \( I' \). It is also conservative as we have only rerouted a single path along a free jump.

We need to show that for all \( b_1, b_2, b_3 \) of congestion \( k - d \) with respect to \( \mathcal{L} \) there is a path \( Q \in \mathcal{L} \) containing \( b_1, b_2, b_3 \). By assumption, such a path \( Q' \) exists in \( S' \). If \( Q' \neq Q_h \), then we are done. So suppose \( Q_h = Q' \). But then this implies that \( b_s \notin \{a_{i+1}, \ldots, a_{j-1}\} \) for all \( 1 \leq s \leq 3 \) as otherwise the congestion of \( b_s \) would have dropped to \( k - d - 1 \) in \( \mathcal{L} \). But then \( b_1, b_2, b_3 \in V(Q_h) \).

It remains to show that for every vertex \( a_s \) of congestion \( k - d \) with respect to \( \mathcal{L} \) there is a free \((i, j)\)-jump for some \( i < s < j \). As before, by assumption, there are \( s_1 < s < s_2 \) and a free \((s_1, s_2)\)-jump with respect to \( S' \). If this jump is not \( P \), then it still exists with respect to \( \mathcal{L} \) and we are done. So suppose this jump is \( P \), which implies that \( i < s < j \). Furthermore, \( a_s \notin Q_h \) as otherwise the congestion of \( a_s \) in \( \mathcal{L} \) would be \( k - d - 1 \). But then, there must be indices \( i_1, i_2 \) with \( i \leq i_1 < s < i_2 \leq j \) such that \( a_{i_1}, a_{i_2} \in V(Q_h) \) and \( a_{i'} \notin V(Q_h) \) for all \( i_1 < s' < i_2 \). Hence, the atomic subpath \( Q'' \) of \( Q_h \) from \( a_{i_1} \) to \( a_{i_2} \) is an \((i_1, i_2)\)-jump as required. As \( Q'' \subseteq Q_h^3 \), this jump is now free.

Finally, the vertex \( a_r \) now has congestion \( k - d - 1 \) with respect to \( \mathcal{L} \) as \( a_r \) is not contained in \( Q_h' \). Hence, \( \mathcal{L} \) has fewer vertices of congestion \( k - d \) than \( S' \), contradicting the choice of \( S' \). Thus, \( S' \) must have been a \((k - d - 1)\)-routing of \( I' \) as required. This completes the proof of the lemma.

By repeatedly applying Lemma 5.5.2 we obtain the following corollary, which essentially implies Theorem 5.2.3.

**Corollary 5.5.2.** Let \( G \) be an acyclic digraph, \( d \geq 0, k \geq 3d \) and let \( I := \{(s_1, t_1), \ldots, (s_k, t_k)\} \) be a set of pairs of vertices such that for all \( 1 \leq i \leq k \) there is a path in \( G \) linking \( s_i \) to \( t_i \). Then \( G \) contains a \((k - d)\)-routing of \( I \) if, and only if, there is a subset \( I' \subseteq I \) with \( |I'| = 3d \) such that \( G \) contains a \( 2d \)-routing of \( I' \).

We are now ready to prove Theorem 5.2.3.
5. On Disjoint Paths Problem in Digraphs

Proof of Theorem 5.2.3. Let $G, k, d$ and $I := \{(s_1, t_1), \ldots, (s_k, t_k)\}$ be given. Let $n := |G|$. If for some $1 \leq i \leq k$ there is no path in $G$ from $s_i$ to $t_i$, then the answer is no and we are done. If $k \leq 3d$, then we can apply Corollary 5.5.1 to compute the answer in time $n^{O(d)}$ as required.

Otherwise, by Corollary 5.5.2, there is a $(k - d)$-routing for $I$ in $G$ if, and only if, there is a subset $I' \subseteq I$ of order $3d$ such that $I'$ has a $2d$-routing. There are $\binom{k}{3d} \leq k^{3d} \leq n^{3d}$ subsets $I'$ of order $3d$. By Corollary 5.5.1, we can decide for any such $I'$ of order $3d$ in time $n^{O(d)}$ whether a $2d$-routing of $I'$ exists. Hence, by iterating through all possible subsets $I'$, we can decide in time $n^{O(d)}$ whether there is a $(k - d)$-routing of $I$ in $G$.

5.6. Lower Bounds on Disjoint Paths with Congestion on DAGs

In this section, we prove Theorem 5.6.2 by a reduction from Partitioned Subgraph Isomorphism. The input of the Partitioned Subgraph Isomorphism problem consists of a graph $H$ with vertex set $\{u_1, \ldots, u_k\}$ and a graph $G$ whose vertex set is partitioned into $k$ classes $V_1, \ldots, V_k$. The task is to find a mapping $\mu : V(H) \rightarrow V(G)$ such that $\mu(u_i) \in V_i$ for every $1 \leq i \leq k$ and $\mu$ is a subgraph embedding, that is, if $u_i$ and $u_j$ are adjacent in $H$, then $\mu(u_i)$ and $\mu(u_j)$ are adjacent in $G$.

Theorem 5.6.1 ([113]). Assuming ETH, Partitioned Subgraph Isomorphism cannot be solved in time $f(k)n^{o(k/\log k)}$ (where $k = |V(H)|$) for any computable function $f$, even when $H$ is assumed to be 3-regular and bipartite.

To prove Theorem 5.6.2, we need a reduction from Partitioned Subgraph Isomorphism (for 3-regular bipartite graphs) to $(k, c)$-Congestion Routing, where the number $k$ of demands is linear in the number of vertices of $H$.

Given Theorem 5.6.1 and with a nice gadget construction it is possible to prove the following theorem, we refer the reader to the [9] for the exact proof.

Theorem 5.6.2. For any fixed integer $c \geq 1$, $k$-Congestion Routing is $\text{W}[1]$-hard parametrized by $k$ and, assuming ETH, cannot be solved in time $f(k)n^{o(k/\log k)}$ for any computable function $f$.

The Theorem 5.6.2 in fact states that the algorithm provided in Theorem 5.2.3 is essentially tight.

5.7. Induced Path Problem

Induced $(s, t)$--path $P$ in a graph $G$ is a path which starts at $s$ and ends at $t$ and the induced graph $G[V(P)]$ is exactly $P$. That means there is no edge between two non-consecutive vertices of $P$.

The most important theoretical roles of induced paths and cycles is in the strong perfect graph theorem [41] which states that perfect graphs are exactly graphs which they neither have an induced cycle of odd length (odd-hole) nor a complement of an odd-hole, which indeed helped researchers to provide a polynomial time algorithm for recognition of perfect graphs [44]. Beside such theoretical importance, induced paths (or cycles) have also some
Diracted graph $D$.

5. On Disjoint Paths Problem in Digraphs

practical applications, e.g. in VLSI design, a routing which the intermediate vertices have less amount of effect on each other is important, so we not only want to find paths which are disjoint from each other we want to find paths which their interconnection is also minimized.

Complexity wise, finding an induced path, between given pair of source and terminal pairs in undirected graphs is easy: just find the shortest path between source and terminal! A similar argument also works if the graph is anyclic. In this section, we answer the following question negatively.

**Question:** Is there a polynomial time algorithm to find an induced $(s, t)$-path in general digraphs?

We also provide a complexity of the optimization version of this problem. For this aim we define \textsc{Shortest $(s, t)$-Induced Path Problem} as follows.

**Definition 5.7.1 (Shortest $(s, t)$-Induced Path Problem (SISTP)).** Input is a directed graph $D$, $s, t \in V(D)$, $w : E(D) \rightarrow \mathbb{N}$, and the task is to find an induced path $P$ from $s$ to $t$ in $D$ which minimizes $w(P) := \sum_{e \in E(P)} w(e)$.

In the inapproximability result in this section we show that SISTP cannot be approximated within any polynomial factor in polynomial time (unless P = NP), even on DAGs. The decision version of the problem is defined as follows.

**Definition 5.7.2 (Length Bounded-$(s, t)$-Induced Path Problem (LBISTP)).** Input: A directed graph $D$, $s, t \in V(D)$, $w : E(D) \rightarrow \mathbb{N}$, $k \in \mathbb{N} \setminus \{0\}$. Question: Is there an induced path $P$ from $s$ to $t$ in $D$ with $w(P) := \sum_{e \in E(P)} w(e) \leq k$ in $D$?

The construction in the following proof is similar to the one in [88] where they show that finding two induced vertex disjoint paths without edges between the paths is NP-complete.

**Theorem 5.2.4.** LBISTP is NP-complete in strong sense even on weighted DAGs.

**Proof.** It is clear the problem belongs to NP, we show the hardness in the following. We use a reduction from 3-SAT to LBISTP. Let $C = C_1 \land \ldots \land C_m$ be an instance of 3-SAT with $n$ variables $X_1, \ldots, X_n$, where each variable $X_i$ appears positive ($x_i$) or negative ($\overline{x}_i$) in some clause $C_j$. In the following we abuse a notation and we create some vertices with labels $x_i$ and $\overline{x}_i$ which are actually corresponding to those literals.

We construct a weighted graph $D$ with weight function $w$ as follows. Assume w.l.o.g. that each clause has exactly 3 literals. For every variable $X_i$, there is a gadget with vertex set $\{x_i^\text{in}, x_i, \overline{x}_i, x_i^\text{out}\}$ and edges $\{(x_i^\text{in}, x_i), (x_i, x_i^\text{out}), (\overline{x}_i, x_i^\text{out})\}$, every edge having weight 1. For every clause $C_j$, there is a gadget with vertices $\{c_j^\text{in}, c_j^1, c_j^2, c_j^3, c_j^\text{out}\}$ and edges $\{(c_j^\text{in}, c_j^t), (c_j^t, c_j^\text{out}) \mid t \in [3]\}$, every edge having weight 1. To connect the gadgets to each other, we add the following edges to $D$. 

(i) For all $i \in [n-1]$ the edge $(x_i^\text{out}, x_{i+1}^\text{in})$ with weight $w(x_i^\text{out}, x_{i+1}^\text{in}) = 1$, 
(ii) for all $j \in [m-1]$ the edge $(c_j^\text{out}, c_{j+1}^\text{in})$ with weight $w(c_j^\text{out}, c_{j+1}^\text{in}) = 1$, 
(iii) the edge $(x_n^\text{out}, c_1^\text{in})$ with weight $w(x_n^\text{out}, c_1^\text{in}) = 1$, 
(iv) for all $i \in [n]$, the edge $(x_i, c_j^t)$ with weight $w(x_i, c_j^t) = (m + n)^2$ if the variable $X_i$ appears positively in $C_j$ at position $t$. 

94
(v) for all $i \in [n]$, the edge $(\bar{x}_i, c_j^i)$ with $w(\bar{x}_i, c_j^i) = (m + n)^2$ if the negation of $X_i$ appears in $C_j$ at position $t$.

It is clear that $D$ is a DAG and can be computed in polynomial time in the size of the given formula, even if we encode $D$ and the weight function and $k$ in unary. We show that $C$ is satisfiable if and only if there exists an induced path of total weight $2(m + n)$ from $x_1^i$ to the $c_m^i$ in $D$. Note that every induced path of total weight at most $2(m + n)$ does not use any edge of the form $(c_i, x_j^i)$ or $(\bar{c}_i, x_j^i)$. Furthermore, every such path contains all vertices of the form $c_i^1$, $c_i^2$, $x_j^1$ and $x_j^2$. Finally, for all $i \in [n]$, it contains exactly one of the vertices $c_i$, $\bar{c}_i$ and, for all $j \in [m]$, exactly one of the vertices $x_j^1$, $x_j^2$, $x_j^3$.

(i) For all $i \in [n]$, select $\bar{x}_i$ if $X_i = \text{true}$ and select $x_i$ if $X_i = \text{false}$.

(ii) For every $j \in [m]$, there exists some $X_i$ that satisfies $C_j$, i.e. $X_i = \text{true}$ and $x_i$ appears in $C_i$, or $X_i = \text{false}$ and $\bar{x}_i$ appears in $C_i$. If there are more than one such $X_i$, we choose an arbitrary one and write $i = \alpha_j$. By construction, for every $j \in [m]$, there is at most one incoming edge $e$ from $\{x_i, \bar{x}_i\}$ to $\{c_j^1, c_j^2, c_j^3\}$. Let $t$ be such that $e = (x_i, c_j^t)$ or $e = (\bar{x}_i, c_j^t)$. Then we select $c_j^t$ for gadget number $j$.

Clearly, $w(P) = 3(m + n) - 1$. Assume towards a contradiction that $P$ is not an induced path, i.e. that there is an edge $e \in E(D)$ that connects two vertices of $P$, but $e \notin E(P)$. From the construction it is clear that $e$ goes from a variable gadget to a clause gadget. W.l.o.g. we can assume that $e = (x_i, c_j^t)$ for some $i \in [n]$, $j \in [m]$ and $t \in [3]$. As $c_j^t \in V(P)$, we have $\alpha_j = i$, so $X_i = \text{true}$. It follows that we selected $\bar{x}_i$ and not $x_i$, so e does not connect vertices of $P$, a contradiction.

For the opposite direction, let $P$ be an induced path of total weight $w(P) = 3(m + n) - 1$ from $c_i^1$ to $x_j^2$. Then in each variable gadget number $i$ exactly one of the vertices $x_i$ and $\bar{x}_i$ is in $P$. We define a variable assignment $\sigma$ by $\sigma(X_i) = \text{true}$ if $x_i \in V(P)$ and $\sigma(X_i) = \text{false}$ if $x_i \in V(P)$. Let us show that $\sigma$ satisfies $C$. Otherwise there is a not satisfied clause $C_j$. Exactly one of $c_j^1$, $c_j^2$ and $c_j^3$ is in $P$, say $c_j^t \in V(P)$. There is a variable $X_i$ which appears in $C_j$ at the first position. W.o.l.g. assume that $X_i$ appears in $P$ positively. Hence by the construction of $D$, it contains the edge $(x_i, c_j^t)$. As $C_j$ is not satisfied, $\sigma(X_i) = \text{false}$, therefore by the definition of $\sigma$, $x_i \in V(P)$ and thus $\bar{x}_i \notin V(P)$. Summing up, both $x_i$ and $c_j^t$ are in $P$, but $D_{\sigma}$ contains the edge $(x_i, c_j^t) \notin E(P)$, so $P$ is not an induced path, a contradiction. This completes our proof.

From above lemmas surprisingly we have the following theorem.

**Theorem 5.2.5** ((s,t)-Induced Path Problem (ISTP) is Hard). Finding an induced $(s,t)$-path in class of directed graphs with feedback vertex set of size 1 is $NP$-complete.

**Proof.** We can easily modify the construction in the proof of Lemma 5.2.4 to show that finding an induced path in (unweighted) graphs with feedback vertex set 1 is $NP$-complete. To do this in the proof of Lemma 5.2.4, we first reverse the direction of the edges with weight $(m + n)^2$ and then forget the weights. Every cycle in the resulting graph has a vertex in some variable gadget and a vertex some clause gadget. But every path from any variable gadget to any clause gadget goes through the vertex $c_i^m$, thus the feedback vertex set of the
5. On Disjoint Paths Problem in Digraphs

graph is \{c^n_i\} of size 1. Similar line of arguments as in the proof of Lemma 5.2.4 provides a polynomial time reduction from 3-SAT problem which completes the claim of theorem.

To show the narrow border between tractability and intractability we also prove the following simple lemma.

**Lemma 5.7.1** (ISTP in P on DAGs). *Finding an induced \((s, t)\)-path in class of directed acyclic graphs is in P.*

**Proof.** Let \(G\) be a DAG. To find an \((s, t)\)-shortest induced path, we just have to find a shortest unweighted path \(P\) between \(s\) and \(t\). There cannot be an edge \(e \in E(G[P]) - E(P)\) as otherwise \(P\) was not the shortest \((s, t)\)-path.

In the following, we show that SISTP cannot be efficiently approximated.

**Theorem 5.7.1.** Unless \(P = NP\), there is no polynomial time algorithm computing a polynomial approximation for Shortest \((s, t)\)-Induced Path Problem, even if all the weights and \(k\) are given in unary and the graph is acyclic.

**Proof.** Assume towards a contradiction that there is a polynomial \(p\) (which maps natural numbers to positive numbers) and a polynomial time algorithm \(A\) that, given an instance \((D_\phi, s, t, w)\) of SISTP, either correctly says that there is no induced path from \(s\) to \(t\) or finds an induced path \(P\) such that if \(P'\) is an optimal solution, then \(w(P) \leq p(w(P')) \cdot w(P')\).

We show how we can use \(A\) to solve 3-SAT in polynomial time. Let \(\phi\) be a propositional formula in 3-CNF. We construct an instance \(D_\phi = (D_\phi, x_{i_1}^{in}, c_{m_1}^{out}, w)\) of SISTP as in the proof of Lemma 5.2.4, but setting the weights of edges \((x_i, c_j)\) to \(w(x_i, c_j) = 9(m + n)^2p(m + n)\).

Then we run \(A\) on \(D_\phi\). As \(D_\phi\) always has an induced path from \(x_{i_1}^{in}\) to \(c_{m_1}^{out}\) (because it is a DAG and there is some path from \(x_{i_1}^{in}\) to \(c_{m_1}^{out}\), there is also an induced path), \(A\) computes an induced path \(P\). If no edge of the form \((x_i, c_j)\) is used, \(w(P) = 3(n + m) - 1\), otherwise \(w(P) \geq 9(m + n)^2p(m + n)\). In the first case we output that \(\phi\) is satisfiable, in the second case that it is not.

It is clear that this algorithm runs in polynomial time even if we write the weights and \(k\) in unary notation. Let us show that the algorithm is correct. If \(\phi\) is satisfiable, then there is an induced path in \(D_\phi\) that never uses edges of the form \((x_i, c_j)\), thus an optimal induced path has total weight \(3(m + n) - 1\) and \(A\) finds a solution \(P'\) with \(w(P') \leq p(3(m + n) - 1) \cdot (3(m + n) - 1)\). However, no path \(P''\) from \(x_{i_1}^{in}\) to \(c_{m_1}^{out}\) has weight \(w(P'')\) where \(3(m + n) - 1 < w(P'') < 9(m + n)^2p(m + n)\), so \(w(P') = 3(m + n) - 1\). As shown in the proof of Lemma 5.2.4, it follows that \(\phi\) is satisfiable.

If \(\phi\) is not satisfiable, all solutions have total weight at least \(w(P) \geq 9(m + n)^2p(m + n)\) and are not found by our algorithm. Thus it answers that \(\phi\) is not satisfiable. \(\square\)
5. On Disjoint Paths Problem in Digraphs

5.8. Conclusion and Future Work

We have studied two main problems: UpPlan-VDPP and \((k, c)\)-Congestion Routing problem on acyclic digraphs and we provide efficient algorithm and hardness results for both of them.

Our algorithm for UpPlan-VDPP had a running time of \(2^{O(k \log k)}\). It is an interesting question to investigate whether the \(k!\) factor in the running time of our algorithm can be improved. Another direction of research is to extend our result to more general but still restricted graph classes, such as digraphs embedded on a torus such that all edges are monotonically increasing in the \(z\)-direction or to acyclic planar graphs. On the other hand, our FPT algorithm requires the upward planar drawing of the graph as input. So another interesting question raises here is the following. If a planar acyclic graph \(G\) has only \(k\) sink vertices and only \(k\) source vertices, for some fixed \(k\), is it possible to provide an upward planar drawing of \(G\) in linear time? These questions are interesting as the known break through FPT algorithm for directed planar graphs [47], is not just double exponential, but it is hard to implement. Also the constant factor in the exponent of the algorithm in [47] make it impractical already for 2 paths. Hence, discovering efficient algorithms in the more restricted class of sparse digraphs is interesting.

Then we considered the \((k, c)\)-Congestion Routing problem. It is easy to see that the \(n^{O(k)}\) algorithm in [65] for solving the disjoint paths problem on acyclic digraphs can be extended to an \(n^{O(k)}\) algorithm for \((k, c)\)-Congestion Routing. It is interesting to see whether our result can be extended to larger classes of digraphs. In particular classes of digraphs of bounded directed treewidth would be a natural target. On such classes, the \(k\)-disjoint paths problem can be solved in time \(n^{O(k+w)}\), where \(w\) is the directed treewidth of the input digraph (see [86]). It is conceivable that our results extend to bounded directed treewidth classes and we leave this for future research.

We also provided some hardness results in regard to induced path problem. While the major problem in this case in undirected graphs is related to finding an induced cycle or induced disjoint paths, we showed, already finding an induced path is hard in digraphs and even for the optimization version of the problem, there is no polynomial time approximation with any approximation guarantee even in acyclic graphs. But still one can parametrize the problem and hope for efficient algorithms. Hence, the following is an interesting open question in this area.

Is it possible to find an induced path of length at most \(\ell\) between two vertices in a directed graph \(G\), in time \(2^{O(\ell)} |G|\)? Clearly providing an algorithm which runs in \(|G|^{O(\ell)}\) is trivial. Here maybe color coding technique can help us to provide an FPT algorithm [6].

It is also interesting to consider those problems in distributed settings. E.g. as we explained before there is a parallel algorithm which computes 2-shortest disjoint paths problem [91]. It is quite interesting to extend such algorithms into distributed settings.
6. Rerouting Problem

6.1. Introduction

In this chapter, we consider the following flow rerouting problem. There is a set of multi-commodity single path flows and the task is to reroute them. However, we cannot reroute them all at a glance due to the asynchronous behavior of the whole system. During the rerouting procedure, for each flow, at each moment there have to be a single path that routes it. Any feasible solution is suitable. We develop several techniques and provide different algorithmic approaches to tackle the problem in acyclic graphs when the number of flows is bounded. On the other hand, we show its hardness when the number of flows is part of the input. As we already introduced the problem in the first chapter of this thesis, here in this chapter, we only focus on formal definitions and technical results, we refer the reader to the first chapter for more backgrounds and previous works.

In the next section, we provide the required definitions. Compare to the other chapters, we have a bit more definitions. All of the following definitions are natural and straightforward definitions. They encapsulate the model and the problem. We note that the reader does not need to memorize the following definitions. Many definitions in the next section are intermediate definitions to provide the “natural” definition of the problem.

6.2. Model and Definitions

We describe the problem in terms of edge capacitated directed graphs. For ease of presentation and without loss of generality, we consider directed graphs with only one source vertex (where flows will originate) and one terminal vertex (the flows sink)\(^1\). We call this graph a flow network.

**Definition 6.2.1 (Flow Network).** A **flow network** is a directed capacitated graph \(G = (V, E, s, t, c)\), where \(s\) is the source, \(t\) the terminal, \(V\) is the set of vertices with \(s, t \in V\), \(E \subseteq V \times V\) is a set of ordered pairs known as edges, and \(c: E \to \mathbb{N}\) a capacity function assigning a capacity \(c(e)\) to every edge \(e \in E\).

The forwarding rules that define the paths considered in our problem, are best seen as flows in a network. We will be interested in rerouting flows such that natural notions of consistency are preserved, such as loop-freedom and congestion-freedom. In particular, we will say that a set of flows is valid if the edge capacities of the underlying network are respected.

**Definition 6.2.2 (Flow, Valid Flow Sets).** An \((s, t)\)-flow \(F\) of capacity \(d \in \mathbb{N}\) is a directed path from \(s\) to \(t\) in a flow network such that \(d \leq c(e)\) for all \(e \in E(F)\). Given a family \(\mathcal{F}\) of

\(^1\)Once the definitions are complete, it is clear that in reality we consider multi-commodity flow not a single flow.
6. Rerouting Problem

(s, t)-flows $F_1, \ldots, F_k$ with demands $d_1, \ldots, d_k$ respectively, we call $F$ a valid flow set, or simply valid, if $c(e) \geq \sum_{i \in E(F_i)} d_i$.

Having defined our notion of flows, we next introduce the notion of flow pairs. Recall that we consider the problem of how to reroute a current (old) flow to a new flow, and hence we will consider such flows in “update pairs”:

**Definition 6.2.3 (Update Flow Pair).** An update flow pair $P = (F^o, F^u)$ consists of two (s, t)-flows $F^o$, the old flow, and $F^u$, the update (or new) flow, each of demand $d$.

Now we can define the update flow network: the update flow network is a flow network (the underlying edge capacitated graph) together with a family of flow pairs. For an illustration, recall the initial network in Figure 6.2: The old flow is presented as the directed path made of solid edges and the new one is represented by the dashed edges.

**Definition 6.2.4 (Update Flow Network).** A graph $G = (V, E, P, s, t, c)$ is a flow network, and $P = \{P_1, \ldots, P_k\}$ with $P_i = (F^o_i, F^u_i)$, a family of update flow pairs of demand $d_i$, $V = \bigcup_{i \in [k]} V(F^o_i \cup F^u_i)$ and $E = \bigcup_{i \in [k]} E(F^o_i \cup F^u_i)$, is called update flow network. The two families $P^o = \{F^o_1, \ldots, F^o_k\}$ and $P^u = \{F^u_1, \ldots, F^u_k\}$ have to be valid.

Now, with all the graphs in place, we can consider the actual rerouting problem. A flow can be rerouted by updating the outgoing edges of the vertices along its path (the forwarding rules), i.e., by blocking the outgoing edge that routes the old flow and by allowing traffic to go along the outgoing edge of the new flow (if either of them exists). When these two edges coincide, no change takes place.

In order to ensure transient consistency, the updates of these outgoing edges need to be scheduled over time: this results in a sequence which can be partitioned into update rounds. For example, in the context of Software-Defined Networks, a controller partitions the update commands into multiple rounds, and issues the next subset of updates only once it has received confirmation of the successful installation of the updates of the previous round (for details, see e.g. [109]).

Now recall that each update command is a pair. Consider the first update in Figure 6.2. Here there is only one flow pair $P = (F^o, F^u)$ and the update of vertex $v_1$ (1) renders its outgoing solid edge dashed, indicating that is not to be used for the routing anymore, and (2) renders its dashed outgoing edge solid: from now on, the new route will follow this edge. Succinctly:

**Definition 6.2.5 (Update).** Given an update flow network $G = (V, E, P, s, t, c)$, an update is a tuple $\mu = (v, P) \in V \times P$.

In order to resolve (“implement”) updates, we need to distinguish between active and inactive edges for a given update flow pair. To be more precise, we need a tool that tells us, for all possible edge subsets, which edges we are allowed to use in the current state of our migration, and which we cannot use. We call this tool activation label. In simple words, if the activation label of an edge (or link) $e$ is true (or states active) for the flow $i$, we can expect that a flow $i$ routes along $e$.

**Definition 6.2.6 (Resolving Updates).** Given $G = (V, E, P, s, t, c)$ and an update flow pair $P = (F^o, F^u) \in P$ of demand $d$, we consider the activation label $\alpha_P : E(F^o \cup F^u) \times 2^{V \times P} \rightarrow \{0, 1\}$.
{active, inactive}. For an edge \((u,v) \in E(F^0 \cup F^u)\) and a set of updates \(U \subseteq V \times \mathcal{P}\), \(\alpha_P\) is defined as follows:

\[
\alpha_P((u,v), U) = \begin{cases} 
  \text{active}, & \text{if } (u, P) \notin U \text{ and } (u,v) \in E(F^0), \\
  \text{active}, & \text{if } (u, P) \in U \text{ and } (u,v) \in E(F^u), \\
  \text{inactive}, & \text{otherwise}.
\end{cases}
\]

The graph \(\alpha(U, G) = (V, \{ e \in E | \text{there is some } i \in [k] \text{ s.t. } \alpha_P(e, U) = \text{active} \})\) is called the \(U\)-state of \(G\) and we call any update in \(U\) resolved.

As an illustration, after the second update in Figure 6.2, one of the original solid edges is still not deactivated. However, already two of the new edges have become solid (i.e., active). So in the picture of the second update, the set \(U = \{(v_1, P), (v_2, P)\}\) has been resolved.

Let \((u, P)\) be some update. When we say that we want to resolve \((u, P)\), we mean that we target a state of \(G\) in which \((u, P)\) is resolved. In most cases this will mean to add \((u, P)\) to the set of already resolved updates. Let \(G = (V,E, \mathcal{P}, s,t,c)\) be an update flow network and \(P = (F^0, F^u) \in \mathcal{P}\) of demand \(d\) an update flow pair, as well as \(U \subseteq V \times \mathcal{P}\). With a slight abuse of notation, we will denote the graph \(\alpha(U, G)[V(F^0 \cup F^u)]\) by \(\alpha_P(U, G)\).

We are now able to determine, for a given set of updates, which edges we edges can and which edges we cannot use for our flows. In the end, we want to describe a process of migration steps, starting from the empty, or initial state, in which no update has been resolved, and finishing in a state, where the only active edges are exactly those of the new flows of every update flow pair. This sequence of updates should be chosen in such a way that every update is resolved exactly once. Therefore, we have to extend the definition of resolving updates by the following:

**Definition 6.2.7 (Update Sequence).** An update sequence \(\mathcal{R} = (r_1, \ldots, r_l)\) is an ordered partition of \(V \times \mathcal{P}\). For every such \(i\) we define \(U_i = \bigcup_{j=1}^{i} r_i\) and consider the activation label \(\alpha_P^i(e) = \alpha_P(e, U_i)\) for every update flow pair \(P = (F^0, F^u) \in \mathcal{P}\) of demand \(d\) and edge \(e \in E(F^0 \cup F^u)\).

Recall that we required the flows to go along a single path. This is due to the unsplittable nature of our flows. Up until now, the notion of active and inactive edges was just an abstract label without real meaning. The following will clarify how active edges are to be used. We formally define this based on the definition of an update flow network \(G\).

**Definition 6.2.8 (Transient Flow).** The flow pair \(P\) is called transient for some set of updates \(U \subseteq V \times \mathcal{P}\), if \(\alpha_P(U,G)\) contains a unique valid \((s,t)\)-flow \(T_{P,U}\).

In short, the transient flows look like a path of active edges for flow \(F\), which starts at the source vertex and ends at the terminal vertex. Note that there may be some active edges connected to this path, but they cannot be used to route the flow since \(T_{P,U}\) is unique after resolving \(U\).

We again refer to Figure 6.2. In each of the different states, the transient flow is depicted as the light blue line connecting \(s\) to \(t\) and covering only solid (i.e., active) edges.

Now it is possible to define the transient family: The intermediate flow structure which will be obtained after a sequence of updates. In fact, the collection of the transient flows corresponding to the transient family is a snapshot of a valid updating scenario. Just recall that whenever we say a path \(p\) "routes" a flow \(F\), we mean that all edges of path \(p\) are active for flow \(F\).
6. Rerouting Problem

**Definition 6.2.9** (Transient Family). If there is a family $\mathcal{P} = \{P_1, \ldots, P_k\}$ of update flow pairs with demands $d_1, \ldots, d_k$ respectively, we call $\mathcal{P}$ a **transient family** for a set of updates $U \subseteq V \times \mathcal{P}$, if and only if every $P \in \mathcal{P}$ is transient for $U$. The family of transient flows in the $U$-state of $G$ is denoted by $\mathcal{T}_{\mathcal{P}, U} = \{T_{P_1, U}, \ldots, T_{P_k, U}\}$.

**Round:** In the above definition the $r_i$ for $i \in [\ell]$ is a round. Given an update sequence $\mathcal{R}$, the round in which some update $(v, P) \in V \times \mathcal{P}$ is resolved is denoted by $\mathcal{R}(v, P)$. Since the rounds are totally ordered, we will slightly abuse the notation and refer to rounds by their indices. We define the initial round $r_0 = \emptyset$.

We are now ready to define consistency rules. Recall that in a valid update, there should always be a transient flow for any flow and the flow set (for any set of transient flows) should be valid. In particular, they should not violate congestion. So more precisely, in each round $r_i$, any subset of updates of $r_i$ resolved without considering the remaining updates or $r_i$ should allow a transient flow for every flow pair. This models the asynchronous nature of the implementation of the update commands in each round. In the following, we rephrase this to provide a concrete model.

**Definition 6.2.10** (Consistency Rule). Let $\mathcal{R} = (r_1, \ldots, r_\ell)$ be an update sequence and $i \in [\ell]$. We require that for any $S \subseteq r_i, \mathcal{U}_i^S := S \cup \bigcup_{j=1}^{i-1} r_j$, there is a family of transient flow pairs $\mathcal{T}_{\mathcal{P}, \mathcal{U}_i^S}$.

Figure 6.1 gives an example which, beside other things, illustrates that rounds cannot be chosen trivially in general. The set $r_1 := \{(v_2, P)\} \subseteq V \times \{P\}$, as seen here, does not obey the consistency rule.

Now we have a natural definition of valid updates.

**Definition 6.2.11** (Valid Update). An update sequence $\mathcal{R}$ is **valid**, or **feasible**, if every round $r_i \in \mathcal{R}$ obeys the consistency rule.

In Figure 6.2, we give an example of a feasible update sequence. The set of updates here is $\{(s, P), (v_1, P), (v_2, P), (t, P)\}$ and the update sequence is as follows: $\mathcal{R} = (\{(s, P), (t, P)\}, \{(v_2, P), \{(s, P)\})$. In each round, including the initial state, the transient flow is depicted as a light blue line. Note that resolving $(t, P)$ does not change the label of any edge; hence it can be resolved in any round.

Note that while succinct, this consistency rule models and consolidates the fundamental properties usually studied in the literature, such as congestion-freedom [32] and loop-freedom [109]. Yet, the case of loop-freedom is somewhat special. Given an update $(v, P) \in V \times \mathcal{P}$ for some update flow network, the vertex $v$ will update all of its outgoing edges belonging to the flows in the update flow pair $P$. Because of this, which is more a property of the model than a property of the consistency rule, no valid update can produce a loop. We formalize this in the following lemma.

**Lemma 6.2.1.** Let $G = (V, E; \mathcal{P}, s, t, c)$ be an update flow network, $P = (F^o, F^u) \in \mathcal{P}$ of demand $d$ and $\mathcal{R} = (r_1, \ldots, r_\ell)$ an update sequence. If $\mathcal{R}$ is valid, for all $i \in \{1, \ldots, \ell\}$ and all $S \subseteq r_i$, there is no loop in $\alpha_P(\mathcal{U}_i^S, G)$ containing vertices of $\mathcal{T}_{\mathcal{P}, \mathcal{U}_i^S}$.

**Proof.** Let $i \in \{1, \ldots, \ell\}$ be arbitrary, as well as $S \subseteq r_i$. In $\alpha_P(\mathcal{U}_i^S, G)$, every vertex has out degree at most 1. Suppose there is some vertex $v \in V$ with out degree at least 2. Thus, there
6. Rerouting Problem

are edges \((v, x)\) and \((v, y)\) with \(x \neq y\). Then w.l.o.g. \((v, x) \in E(F^u)\) and \((v, y) \in E(F^o)\). To see this, consider the update \((u, P)\). If this is already resolved, all outgoing edges of \(v\) that belong to \(F^o\) are deactivated, and otherwise all outgoing edges of \(F^u\) are still inactive.

Now \(T_{P; \mathcal{H}^\delta}\) is a transient (\(s, t\))-flow and thus, by our definition of such flows, it is a path. The terminal \(t\) in particular has out degree 0 in \(\alpha_P(P)\). Thus any vertex \(u \in V(T_{P; \mathcal{H}^\delta}) \setminus \{t\}\) has exactly one outgoing edge with the direct successor of \(u\) along \(T_{P; \mathcal{H}^\delta}\) as the other endpoint. Hence none of the vertices in \(V(T_{P; \mathcal{H}^\delta})\) can be contained in a cycle.

Note that we do not forbid edges \(e \in E(F^o \cap F^u)\) and we never activate or deactivate such an edge. Starting with an initial update flow network, these edges will be active and remain so until all updates are resolved. Hence there are vertices \(v \in V\) with either no outgoing edge for a given flow pair \(F\) at all; or it has an outgoing edge, but this edge is used by both the old and the update flow of \(F\). We will call such updates \((v, P)\) empty.

Empty updates do not have any impact on the actual problem since they never change any transient flow. Hence they can always be scheduled in the first round and thus w.l.o.g. we can ignore them in the following. In the following we define the main problem which we consider in this paper.

**Definition 6.2.12 (k-Network Flow Update Problem).** Given an update flow network \(G\) with \(k\) update flow pairs, is there a feasible update sequence \(\mathcal{R}\)? The corresponding optimization problem is: What is the minimum \(\ell\) such that there exists a valid update sequence \(\mathcal{R}\) using exactly \(\ell\) rounds?

### 6.3. Preliminaries

Let \(G = (V, E, \mathcal{P}, s, t, c)\) be an acyclic update flow network, i.e., we assume that the graph \((V, E)\) is acyclic. Let \(<\) be a topological order on the vertices \(V = \{v_1, \ldots, v_n\}\). Let \(P_i = (F^o_i, F^u_i)\) be an update flow pair of demand \(d\) and let \(v^1_i, \ldots, v^\ell_i\) be the induced topological order on the vertices of \(F^o_i\); analogously, let \(u^1_i, \ldots, u^\ell_i\) be the order on \(F^u_i\). Furthermore, let \(V(F^o_i) \cap V(F^u_i) = \{z^1_i, \ldots, z^{k_i}_i\}\) be ordered by \(<\) as well. The subgraph of \(F^o_i \cup F^u_i\) induced by the set \(\{v \in V(F^o_i \cup F^u_i) \mid z^j_i < v < z^{j+1}_i\}, j \in [k_i - 1]\), is called the \(j\)th block of the update flow pair \(F_i\), or simply the \(j\)th block. We will denote this block by \(b^j_i\).

For a block \(b\), we define \(\mathcal{I}(b)\) to be the start of the block, i.e., the smallest vertex w.r.t. \(<\); similarly, \(\mathcal{E}(b)\) is the end of the block: the largest vertex w.r.t. \(<\).

Let \(G = (V, E, \mathcal{P}, s, t, c)\) be an update flow network with \(\mathcal{P} = \{P_1, \ldots, P_k\}\) and let \(\mathcal{B}\) be the set of its blocks. We define a binary relation \(<\) between two blocks as follows. For two blocks \(b_1, b_2 \in \mathcal{B}\), where \(b_1\) is an \(i\)-block and \(b_2\) a \(j\)-block, \(i, j \in [k]\), we say \(b_1 < b_2\) (\(b_1\) is smaller than \(b_2\)) if one of the following holds:

i. \(\mathcal{I}(b_1) < \mathcal{I}(b_2)\),

ii. if \(\mathcal{I}(b_1) = \mathcal{I}(b_2)\) then \(b_1 < b_2\), if \(\mathcal{E}(b_1) < \mathcal{E}(b_2)\),

iii. if \(\mathcal{I}(b_1) = \mathcal{I}(b_2)\) and \(\mathcal{E}(b_1) = \mathcal{E}(b_2)\) then \(b_1 < b_2\), if \(i < j\).
6. Rerouting Problem

Let $b$ be an $i$-block and $P_i$ the corresponding update flow pair. For a feasible update sequence $\mathcal{R}$, we will denote the round $\mathcal{R}(\mathcal{X}(b), P_i)$ by $\mathcal{R}(b)$. We say that $i$-block $b$ is updated, if all edges in $b \cap F_i^u$ are active and all edges in $b \cap F_i^o \setminus F_i^u$ are inactive.

6.4. Congestion-Free Rerouting of Flows on DAGs

In this section and next three sections we focus on $k$-network flow update problem, alternatively we call it congestion free rerouting of flows. Figures 6.1 and 6.2 presents examples of the rerouting problem for only a single flow.

Figure 6.1: Example: We are given an initial network consisting of exactly one active flow $F^o$ (solid edges) and the inactive edges (i.e., inactive forwarding rules) of the new flow $F^u$ to which we want to reroute (dashed edges). Together we call the two flows an (update) pair $P = (F^o, F^u)$. Updating the outgoing edges of a vertex means activating all previously inactive outgoing edges of $F^u$, and deactivating all other edges of the old flow $F^o$. Initially, the blue flow is a valid (transient) $(s, t)$-flow. If all updates are sent out simultaneously, due to asynchrony, it can happen that the update of vertex $v_2$ takes effect first, introducing an invalid (not transient) flow (in pink): traffic is forwarded in a loop, hence (temporarily) destroying the path from $s$ to $t$, if no preparations are made.

6.5. NP-Hardness of 2-Flow Update in General Graphs

It is easy to see that for an update flow network with a single flow pair, feasibility is guaranteed.

Theorem 6.5.1. The 2-flow network update problem is NP-hard.

The proof is by reduction from 3-SAT. In what follows let $C$ be any 3-SAT formula with $n$ variables and $m$ clauses. We will denote the variables as $X_1, \ldots, X_n$ and the clauses as $C_1, \ldots, C_m$. The resulting update flow network will be denoted as $G(C)$. Furthermore, we will assume that the variables are ordered by their indices and their appearance in each clause respects this order. The details of the proof of Theorem 6.5.1 is available in [8].

6.6. Optimal Solution for $k = 2$ Flows

As we have seen in the previous section, even the 2-flow update problem is computationally hard on general graphs. However, we will now show that an elegant polynomial-time solution exists for the more restricted class of Directed Acyclic Graphs (DAGs). Our algorithm is
Rerouting Problem

Figure 6.2. Example: We revisit the network of Figure 6.1 and reroute from $F^o$ to $F^u$ without interrupting the connection between $s$ and $t$ along a unique (transient) path (in blue). To avoid the problem seen in Figure 6.1, we first update the vertex $v_2$ in order to establish a shorter connection from $s$ to $t$. Once this update has been confirmed, the update of $v_2$ can be performed without creating a loop. Finally, by updating $s$, we complete the rerouting. Note that this 3-round update schedule is not optimal: The last two updates can be performed in any order, and can hence be scheduled asynchronously, in the same round. The resulting shorter update schedule has 2 rounds.

Based on a dependency-graph approach, and not only finds a feasible, but also a shortest schedule (in terms of number of rounds).

In the following, let $G = (V, E, P, s, t, c)$ be an update flow network where $(V, E)$ forms a DAG and $P = \{B, R\}$ are the two update flow pairs with $B = (B^o, B^u)$ and $R = (R^o, R^u)$ of demands $d_B$ and $d_R$. As in the previous section, we identify $B$ with blue and $R$ with red.

We say that an $I$-block $b_1$ is dependent on a $J$-block $b_2$, $I, J \in \{B, R\}$, $I \neq J$, if there is an edge $e \in (E(b_1) \cap E(I^o)) \cap (E(b_2) \cap E(J^o))$, but $c(e) < d_I + d_J$. In fact, to update $b_1$, we either violate capacity constraints, or we update $b_2$ first in order to prevent congestion. In this case, we write $b_1 \rightarrow b_2$ and say that $b_1$ requires $b_2$. A block that does not depend on any other block is called free.

We say a block $b$ is a free block, if it is not dependent on any other block. A dependency graph of $G$ is a graph $D = (V_D, E_D)$ for which there exists a bijective mapping $\mu: V(D) \leftrightarrow B(G)$, and there is an edge $(v_b, v_{b'})$ in $D$ if $b \rightarrow b'$. Clearly, a block $b$ is free if and only if it corresponds to a sink in $D$.

We propose the following algorithm to check the feasibility of the flow rerouting problem.

**Algorithm 6. Feasible 2-Flow DAG Update**

**Input:** Update Flow Network $G$

1. Compute the dependency graph $D$ of $G$.
2. If there is a cycle in $D$, return *impossible to update*.
3. While $D \neq \emptyset$ repeat:
   i. Update all blocks which correspond to the sink vertices of $D$ as in Algorithm 7.
   ii. Delete all of the current sink vertices from $D$.  

---

Figure 6.2. Example: We revisit the network of Figure 6.1 and reroute from $F^o$ to $F^u$ without interrupting the connection between $s$ and $t$ along a unique (transient) path (in blue). To avoid the problem seen in Figure 6.1, we first update the vertex $v_2$ in order to establish a shorter connection from $s$ to $t$. Once this update has been confirmed, the update of $v_2$ can be performed without creating a loop. Finally, by updating $s$, we complete the rerouting. Note that this 3-round update schedule is not optimal: The last two updates can be performed in any order, and can hence be scheduled asynchronously, in the same round. The resulting shorter update schedule has 2 rounds.
Recall that empty updates can always be scheduled in the first round, even for infeasible problem instances. So for Algorithm 6 and all following algorithms we simply assume these updates to be scheduled together with the non-empty updates of round 1.

The 6.3 gives an example of an update flow network on a DAG and illustrates the block decomposition and its value to finding a feasible update sequence.

Figure 6.3.: Example for Algorithm 6. The 2 update flow pairs are red and blue, each of demand 1. The active edges of the respective colors are indicated as solid lines and the inactive edges are dashed. Each edge in the flow graph is annotated with its current load (top) and its capacity (bottom). We start by identifying the blue and red blocks. For red there is exactly one such block \( r_1 \), since \( R^o \) and \( R^u \) only coincide in \( s \) and \( t \). The blue flow pair on the other hand omits two blocks \( b_1 \) and \( b_2 \): \( B^o \) and \( B^u \) meet again at \( w \) and at \( t \). We observe that \( b_2 \) can only be updated after \( r_1 \) has been updated; similarly, \( r_1 \) can only be updated after \( b_1 \) has been updated. An update sequence respecting these dependencies can be constructed as follows. We can first prepare the blocks by updating the following two out-edges which currently do not carry any flow: \((w, red)\), \((u, blue)\), and \((v, blue)\). Subsequently, the three blocks can be updated in a congestion-free manner in the following order: Prepare the update for all blocks in the first round. Then, update \( b_1 \) in the second round, \( r_1 \) in the third round, \( b_2 \) in the fourth round.

Suppose \( \mathcal{R} \) is a feasible update sequence for \( G \). We say a c-block \( b \) w.r.t. \( \mathcal{R} = (r_1, \ldots, r_\ell) \) is updated in consecutive rounds, if the following holds: if some of the edges of \( b \) are activated/deactivated in round \( i \) and some others in round \( j \), then for every \( i < k < j \), there is an edge of \( b \) which is activated/deactivated. We update free blocks as follows:
6. Rerouting Problem

Algorithm 7. Update a Free $c$-Block $b$

1. Resolve $(v, P_i)$ for all $v \in P_v - b - \mathcal{S}(b)$.
2. Resolve $(\mathcal{S}(b), P_i)$.
3. Resolve $(v, P_i)$ for all $v \in (b - P_v)$.

Lemma 6.6.1. Let $b$ be a $c$-block. Then in a feasible update sequence $\mathcal{R}$, all vertices (resp. their outgoing $c$ flow edges) in $F_v \cap b - \mathcal{S}(b)$ are updated strictly before $\mathcal{S}(b)$. Moreover, all vertices in $b - F_v$ are updated strictly after $\mathcal{S}(b)$ is updated.

Proof. In the following, we will implicitly assume flow $c$, and will not mention it explicitly everywhere. We will write $F_v$ for $F_v \cap b$ and $F_v$ for $F_v \cap b$. For the sake of contradiction, let $U = \{v \in V(G) \mid v \in F_v - F_v - \mathcal{S}(b), \mathcal{R}(v, c) > \mathcal{R}(\mathcal{S}(b), c)\}$. Moreover, let $v$ be the vertex of $U$ which is updated the latest and $\mathcal{R}(v, c) = \max_{u \in U} \mathcal{R}(u, c)$. By our condition, the update of $v$ enables a transient flow along edges in $F_v \cap b$. Hence, there now exists an $(s, t)$-flow through $b$ using only update edges.

No vertex in $F_1 := F_v - (F_v - \mathcal{S}(b))$ could have been updated before, or simultaneously with $v$: otherwise, between the time $u$ has been updated and before the update of $v$, there would not exist a transient flow. But once we update $v$, there is a $c$-flow which traverses the vertices in $F_v - F_v$, and another $c$-flow which traverses $v \notin F_1$: a contradiction. Note that $F_1 \neq \emptyset$. The other direction is obvious: updating any vertex in $(F_v \cap b) - F_v$ inhibits any transient flow.

Lemma 6.6.2. Consider $G$ and assume a feasible update sequence $\mathcal{R}$. Then there exists a feasible update sequence $\mathcal{R}'$ which updates every block in at most 3 consecutive rounds.

Proof. Let $\mathcal{R}$ be a feasible update sequence with a minimum number of blocks that are not updated in 3 consecutive rounds. Furthermore let $b$ be such a $c$-block. Let $i$ be the round in which $\mathcal{S}(b)$ is updated. Then by Lemma 6.6.1, all other vertices of $F_v \cap b$ have been updated in the previous rounds. Moreover, since they do not carry any flow during these rounds, the edges can all be updated in round $i - 1$. By our assumption, we can update $\mathcal{S}(b)$ in round $i$, and hence now this is still possible.

As $\mathcal{S}(b)$ is updated in round $i$, the edges of $F_v \cap b$ do not carry any active $c$-flow in round $i + 1$ and thus we can deactivate all remaining such edges in this round. This is a contradiction to the choice of $\mathcal{R}$, and hence there is always a feasible sequence $\mathcal{R}$ satisfying the requirements of the lemma. In particular, Algorithm 7 is correct.

From the above two lemmas, we immediately derive a corollary regarding the optimality in terms of the number of rounds: the 3 rounds feasible update sequence.

Corollary 6.6.1. Let $b$ be any $c$-block with $|E(b \cap F_v)| \geq 2$ and $|E(b \cap F_v)| \geq 2$. Then it is not possible to update $b$ in less than 3 rounds: otherwise it is not possible to update $b$ in less than 2 rounds.

Next we show that if there is a cycle in the dependency graph, then it is impossible to update any flow.
Lemma 6.6.3. If there is a cycle in the dependency graph, then there is no feasible update sequence.

Proof. A cycle of length 2 in the dependency graph means that there is a \( c_1 \)-block \( b_1 \) and a \( c_2 \)-block \( b_2 \) whose updates mutually depend on each other: If there was a feasible update sequence, then according to Lemma 6.6.2 there would exist a feasible update sequence which either updates \( b_1 \) and then \( b_2 \) (or vice versa), or which updates them simultaneously. However, this is not possible due to the mutual dependency.

On the other hand, the dependency graph is bipartite. Therefore, every cycle of length more than two has length at least 4. So suppose that there exists a cycle \( C = b_1, b_2, ..., b_{2\ell} \) whose vertices correspond to blocks of the corresponding name: blocks of odd indices belong to the \( c \)-flow and blocks of even indices belong to the \( c' \)-flow. Suppose that \( b_1^1 \) is the smallest block of the \( c \)-flow in \( C \). There are two cases:

1. \( b_2 \leq b_{2\ell} \)
2. \( b_{2\ell} \leq b_2 \)

First suppose \( b_2 < b_{2\ell} \), and therefore \( \delta'(b_2) \leq \mathcal{I}(b_{2\ell}) \). As \( b_2 \) requires \( b_3 \), we have \( \mathcal{I}(b_3) < \delta'(b_2) \). However, as \( b_1 \) was the smallest \( c \)-block in \( C \), we have \( \delta'(b_1) \leq \mathcal{I}(b_3) < \delta'(b_2) \) or \( \delta'(b_1) < \delta'(b_2) \). On the other hand, \( b_{2\ell} \) requires \( b_1 \), and therefore \( \mathcal{I}(b_{2\ell}) < \delta'(b_1) \). But then \( \mathcal{I}(b_{2\ell}) < \delta'(b_2) \): a contradiction to our assumption.

Therefore, if there is a feasible update sequence, then \( b_{2\ell} < b_2 \). Moreover, \( b_{2\ell-1} \rightarrow b_{2\ell} \) so \( \delta'(b_1) \leq \mathcal{I}(b_{2\ell-1}) < \mathcal{I}(b_{2\ell}) \leq \mathcal{I}(b_2) \) and hence \( \delta'(b_1) < \mathcal{I}(b_2) \). But \( b_1 \rightarrow b_2 \) thus \( \mathcal{I}(b_2) < \delta'(b_1) \): a contradiction. So there cannot be a cycle of length at least 4 in the dependency graph.

We will now slightly modify Algorithm 6 to create a new algorithm which not only computes a feasible sequence \( \mathcal{R} \) for a given update flow network in polynomial time, whenever it exists, but which also ensures that \( \mathcal{R} \) is as short as possible (in terms of number of rounds). For any block \( b \), let \( c(b) \) denote its corresponding flow pair.

Algorithm 8. Optimal 2-Flow DAG Update

**Input:** Update Flow Network \( G \)

1. Compute the dependency graph \( D \) of \( G \).
2. If there is a cycle in \( D \), return impossible to update.
3. If there is any block \( b \) corresponding to a sink vertex of \( D \) with \( (b \cap F_{c(b)}^u) - \mathcal{I}(b) \neq \emptyset \) set \( i := 2 \), otherwise set \( i := 1 \).
4. While \( D \neq \emptyset \) repeat:
   i. Schedule the update of all blocks \( b \) which correspond to the sink vertices of \( D \) as in Algorithm 7 for the rounds \( i - 1, i, i + 1 \), such that \( \mathcal{I}(b) \) is updated in round \( i \).
   ii. Delete all of the current sink vertices from \( D \).
   iii. Set \( i := i + 1 \).
Theorem 6.6.1. An optimal (feasible) update sequence on acyclic update flow networks with exactly 2 update flow pairs can be found in linear time.

Proof. Let $G$ denote the given update flow network. In the following, for ease of presentation, we will slightly abuse terminology and say that “a block is updated in some round”, meaning that the block is updated in the corresponding consecutive rounds by Algorithm 7.

We proceed as follows. First, we find a block decomposition and create the dependency graph of the input instance. This takes linear time only. If there is a cycle in that graph, we output impossible (cf. Lemma 6.6.3). Otherwise, we apply Algorithm 8. As there is no cycle in the dependency graph (a property that stays invariant), in each round, either there exists a free block which is not processed yet, or everything is already updated or is in the process of being updated. Hence, if there is a feasible solution (it may not be unique), we can find one in time $O(|G|)$.

For the optimality in terms of the number of rounds, consider two feasible update sequences. Let $R_{\text{Alg}}$ be the update sequence produced by Algorithm 8 and let $R_{\text{Opt}}$ be a feasible update sequence that realizes the minimum number of rounds. According to Lemma 6.6.1, any block $b$ is processed only in round $R(b)$.

Suppose there is a block $b'$ such that $t_{\text{Opt}}(b') < t_{\text{Alg}}(b')$. Then let $b$ be the block with the smallest such $t_{\text{Opt}}(b)$. Hence, for every block $b''$ with $t_{\text{Opt}}(b'') \leq t_{\text{Opt}}(b)$, $t_{\text{Opt}}(b'') \geq t_{\text{Alg}}(b'')$ holds. Since $R(b)$ is updated in round $t_{\text{Opt}}(b)$, there are no dependencies for $b$ that are still in place in this round. Thus, according to the sequence $R_{\text{Opt}}$, $b$ is a sink vertex of the dependency graph after round $t_{\text{Opt}}(b) - 1$. Furthermore, by our previous observation, every start of some block has been updated up to this round in the optimal sequence, and hence it is also already updated in the same round in $R_{\text{Alg}}$. This means that after round $t_{\text{Opt}}(b) - 1 < t_{\text{Alg}}(b) - 1$, $b$ is a sink vertex of the dependency graph of $R_{\text{Alg}}$ as well. Thus, Algorithm 8 would have scheduled the update of block $b$ in the rounds $t_{\text{Opt}}(b) - 1$, $t_{\text{Opt}}(b)$ and $t_{\text{Opt}}(b) + 1$. Contradiction.

Thus $t_{\text{Alg}}(b) \leq t_{\text{Opt}}(b)$ for all blocks $b$. Now let $b_1, \ldots, b_{\ell}$ be the last blocks whose starts are updated the latest under $R_{\text{Alg}}$. If there is some $i \in [\ell]$ such that $|E_{b_i}^o| \geq 2$ and $|E_{b_i}^u| \geq 2$, $R_{\text{Alg}}$ uses exactly $t_{\text{Alg}}(b_i) + 1$ rounds; otherwise it is one round less, by Corollary 6.6.1. By our previous observation, none of these blocks can start later than $t_{\text{Alg}}(b_i)$ and thus $t_{\text{Opt}}$ uses at least as many rounds as Algorithm 8. Hence the algorithm is optimal in the number of rounds.

\[ \square \]

6.7. Updating $k$-Flows in DAGs is NP-complete

In this section we show that if the number of flows, $k$, is part of the input, the problem remains hard even on DAGs. In fact, we prove the following theorem.

Theorem 6.7.1. If the number of flows $k$, is part of the input, then finding a feasible update sequence for $k$-flows is NP-hard, even when the update graph $G$ is acyclic.

To prove the theorem, we provide a polynomial time reduction from the 3-SAT problem. Let $C = C_1 \land \ldots \land C_m$ be an instance of 3-SAT with $n$ variables $X_1, \ldots, X_n$, where each variable $X_i$ appears positive ($x_i$) or negative ($\bar{x}_i$) in some clause $C_j$. We construct an acyclic network update graph $G$ such that there is a feasible sequence of updates $R$ for $G$, if and only if $C$ is satisfiable by some variable assignment $\sigma$. By Lemma 6.6.2, we know that if $G$
6. Rerouting Problem

has a feasible update sequence, then there is a feasible update sequence which updates each block in consecutive rounds.

In the following, we denote the first vertex of a directed path \( p \) with \( \text{head}(p) \) and the end vertex with \( \text{tail}(p) \). Furthermore, we number the vertices of a path \( p \) with numbers \( 1, \ldots, |V(p)| \), according to their order of appearance in \( p \) (\( \text{head}(p) \) is number 1). We will write \( p(i) \) to denote the \( i \)’th vertex in \( p \).

We now describe how to construct the initial update flow network \( G \).

1. \( G \) has a start vertex \( s \) and a terminal vertex \( t \).

2. We define \( n \) variable selector flow pairs \( S_1, \ldots, S_n \), where each \( S_i = (S_i^o, S_i^u) \) is of demand 1, as follows:
   a) **Variable Selector Old Flows** are \( n \) \( s \), \( t \)-flows \( S_1^o, \ldots, S_n^o \) defined as follows:
      Each consists of a directed path of length 3, where every edge in path \( S_i^o \) (for \( i \in [n] \)) has capacity 1, except for the edge \( (S_i^o(2), S_i^o(3)) \), which has capacity 2.
   b) **Variable Selector Update Flows** are \( n \) \( s \), \( t \)-flows \( S_1^u, \ldots, S_n^u \) defined as follows:
      Each consists of a directed path of length 5, where the edge’s capacity of path \( S_i^u \) is set as follows: \( (S_i^u(2), S_i^u(3)) \) has capacity 2, \( (S_i^u(4), S_i^u(5)) \) has capacity \( m \), and the rest of its edges has capacity 1.

3. We define \( m \) clause flow pairs \( C_1, \ldots, C_m \), where each \( C_i = (C_i^o, C_i^u) \) is of demand 1, as follows:
   a) **Clauses Old Flows** are \( m \) \( s \), \( t \)-flows \( C_1^o, \ldots, C_m^o \) each of length 5, where for \( i, j \in [m] \), \( C_i^o(3) = C_j^o(3) \) and \( C_i^o(4) = C_j^o(4) \). Otherwise they are disjoint from the above defined. The edge \( (C_i^o(3), C_j^o(4)) \) (for \( i \in [m] \)) has capacity \( m \), all other edges in \( C_i^o \) have capacity 1.
   b) **Clauses Update Flows** are \( m \) \( s \), \( t \)-flows \( C_1^u, \ldots, C_m^u \) each of length 3. Every edge in those paths has capacity 3.

4. We define a Clause Validator flow pair \( V = (V^o, V^u) \) of demand \( m \), as follows.
   a) **Clause Validator Old Flow** is an \( s \), \( t \)-flow \( V^o \) whose path consists of edges \( (s, S_1^o(4)), (S_1^o(4), S_1^u(5)), (S_1^o(5), S_2^o(4)), (S_2^o(4), S_2^u(5)), (S_2^u(5), t) \) for \( i \in [n-1] \). Note that, the edge \( (S_i^o(4), S_i^u(5)) \) (for \( i \in [n] \)) also belongs to \( S_i^u \). All edges of \( V \) have capacity \( m \).
   b) **Clause Validator Update Flow** is an \( s \), \( t \)-flow \( V^u \) whose path has length 3, such that \( V^u(2) = C_i^o(3), V^u(3) = C_i^u(4) \). All new edges of \( V^u \) have capacity \( m \).

5. We define \( 2n \) literal flow pairs \( L_1, \ldots, L_{2n} \). Each \( L_i = (L_i^o, L_i^u) \) of demand 1 is defined as follows:
   a) **Literal’s Old Flows** are \( 2n \) \( s \), \( t \)-flows \( L_1^o, \ldots, L_n^o \) and \( \bar{L}_1^o, \ldots, \bar{L}_n^o \). Suppose \( x_i \) appears in clauses \( C_{i_1}, \ldots, C_{i_{2\ell}} \), then the path \( L_i^o \) is a path of length \( 2\ell + 5 \), where \( L_i^o(2j + 1) = C_{i_1}^o(2), L_i^o(2j + 2) = C_{i_j}^o(3) \) for \( j \in [\ell] \) and furthermore \( L_i^o(2\ell + 3) = S_1^u(2), L_i^o(2\ell + 4) = S_1^u(3) \). On the other hand, if \( \bar{x}_i \) appears in clauses \( C_{i_1}, \ldots, C_{i_{2\ell}} \), then \( \bar{L}_i^o \) is a path of length \( 2\ell + 5 \) where \( \bar{L}_i^o(2j + 3) = C_{i_j}^o(3), \bar{L}_i^o(2j + 4) = C_{i_j}^u(3) \)
for $j \in \{l'\}$, and furthermore $L_i^q(2l' + 3) = S_i^u(2), L_i^q(2l' + 4) = S_i^u(3)$. All new edges in $L_i^q$ (resp. $L_i^o$) have capacity 3. Note that some $L_i^q$’s may share common edges.

b) **Literal’s Update Flows** are $2n$ s,t-flows $L_1^u, \ldots, L_n^u$ and $\bar{L}_1^u, \ldots, \bar{L}_n^u$. For $i \in [n]$, $L_i^u$ and $\bar{L}_i^u$ are paths of length 5 such that $L_i^u(2) = S_i^o(2)$ and $L_i^u(3) = \bar{L}_i^u(3) = S_i^o(3)$. All new edges in those paths have capacity 3.

Note that $G$ is acyclic and every flow pair in $G$ forms a single block. Let $R$ be a feasible update sequence of $G$. We suppose in $R$, every block is updated in consecutive rounds (Lemma 6.6.1). For a single flow $F$, we write $R(F)$ for the round where the last edge of $F$ was updated.

**Lemma 6.7.1.** For $R$ and $G$, we have the following observations.

1. We either have $R(L_i^o) < R(S_i^o) < R(L_i^o)$, or $R(\bar{L}_i^o) < R(S_i^o) < R(L_i^o)$, for all $i \in [n]$.

2. $R(C_i^o) < R(V^o)$ for all $i \in [m]$.

3. $R(S_i^o) < R(V^o)$ for all $i \in [n]$.

4. For every $i \in [m]$ there is some $j \in [n]$ such that $R(C_i^o) < R(L_j^o)$ or $R(C_i^o) < R(\bar{L}_j^o)$.

5. We either have $R(L_j^o) < R(C_i^o) < R(\bar{L}_j^o)$, or $R(\bar{L}_j^o) < R(C_i^o) < R(L_j^o)$, for all $i \in [m]$ and all $j \in [n]$.

**Proof.**

i) As the capacity of the edge $e = (S_i^o(2), S_i^o(3))$ is 2, and both $L_i^u, \bar{L}_i^u$ use that edge, before updating both of them, $S_i^o$ (resp. $S_i^u$) should be updated. On the other hand, the edge $e' = (S_i^o(2), S_i^o(3))$ has capacity 2 and it is in both $L_i^o$ and $\bar{L}_i^o$. So to update $S_i^o, e'$ for one of the $L_i^o, \bar{L}_i^o$ should be updated.

ii) The edge $(V^u(2), V^u(3))$ of $V^u$ also belongs to all $C_i^o$ (for $i \in [m]$) and its capacity is $m$. Moreover, the demand of $(V^o, V^u)$ is $m$, so $V^o$ cannot be updated unless $C_i^o$ has been updated for all $i \in [m]$.

iii) Every $S_i^u$ ($i \in [n]$) requires the edge $(S_i^u(4), S_i^u(5))$, which is also used by $V^o$, until after round $R(V^o)$.

iv) This is a consequence of Observation iii and Observation ii.

v) This is a consequence of Observation iv and Observation i.

**Proof of Theorem 6.7.1.** Given a sequence of updates, we can check if it is feasible or not. The length of the update sequence is at most $k$ times the size of the graph, hence, the problem clearly is in NP.

To show that the problem is complete for NP, we use a reduction from 3-SAT. Let $C$ be as defined earlier in this section, and in polynomial time we can construct $G$. 

110
6. Rerouting Problem

Figure 6.4: Gadget Construction for Hardness in DAGs: There are 4 types of flows: Clause flows, Literal flows, Clause Validator flow and Literal Selector flows. The edge \((S_o(2), S_o(3))\) cannot route 3 different flows \(S_o, L_u, \bar{L}_u\) at the same time. On the other hand the edge \((S_i(2), S_i(3))\) cannot route the flow \(S_i\) before updating either \(L_i\) or \(\bar{L}_i\), hence by the above observation, exactly one of the \(L_i\) or \(\bar{L}_i\)'s will be updated strictly before \(S_i\) and the other will be updated strictly after \(S_i\) was updated. Only after all Clause flows are updated, the edge \((C_k(3), C_k(4))\) can route the flow \(V\) (Clause Validator flow). A Clause flow \(C_k\) can be updated only if at least one of the Literal flows which goes along \((C_u(2), C_u(3))\) is updated. So in each clause, there should be a valid literal. On the other hand the Clause validator flow can be updated only if all Clause Selector flows are updated, this is guaranteed by the edge \((S_i(4), S_i(5))\). Hence, before updating all clauses, we are allowed to update at most one of the \(L_i\) or \(\bar{L}_i\)'s, and this corresponds to a valid satisfying assignment.

By the construction of \(G\), if there is a satisfying assignment \(\sigma\) for \(C\), we obtain a sequence \(R\) to update the flows in \(G\) as follows. In the first round, if in \(\sigma\) we have \(X_i = 1\) for some \(i \in [n]\), update the literal flow \(L_i^u\); otherwise update the literal flow \(L_i^l\). Afterwards, since \(\sigma\)
satisfies $C$, for every clause $C_i$ there is some literal flow $L_j$ or $\overline{L}_j$, which is already updated. Hence, for all $i \in [m]$ the edge $(C^+_i(3), C^+_i(4))$ incurs a load of 2 while its capacity is 3. Therefore, we can update all of the clause flows and afterwards the clause validator flow $V^o$.

Next, we can update the clause selector flows and at the end, we update the remaining half of the literal flows.

On the other hand, if there is a valid update sequence $R$ for flows in $G$, by Lemma 6.7.1 observation $v$, there are exactly $n$ literal flows that have to be updated, before we can update $C^+_i$. To be more precise, for every $j \in [n]$, either $L^o_j$, or $\overline{L}^o_j$ has to be updated, but never both. If $L^o_j$ is one of those first $n$ literal flows to be updated for some $j \in [n]$, we set $X_j := 1$; otherwise $\overline{L}^o_j$ is to be updated and we set $X_j := 0$. Since these choices are guaranteed to be unique for every $j \in [n]$, this gives us an assignment $\sigma$. After these $n$ literal flows are updated, we are able to update the clause flows, since $R$ is a valid update sequence. This means in particular, that for every clause $C_i$, $i \in [m]$, there is at least one literal which is set to true. Hence $\sigma$ satisfies $C$ and therefore solving the network update problem on DAGs, is as hard as solving the 3-SAT problem.

6.8. Linear Time Algorithm for Constant Number of Flows on DAGs

We have seen that for an arbitrary number of flows, the problem is hard even on DAGs. However for $k = 2$ flows, the problem is in P. In this section, we show that if the number of flows is a constant $k$, then a solution can be computed in linear time. More precisely, we describe an algorithm to solve the network update problem on DAGs in time $2^{O(k \log k)}O(|G|)$, for arbitrary $k$. In the remainder of this section, we assume that every block has at least 3 vertices (otherwise, postponing such block updates will not affect the solution).

We say a block $b_1$ touches a block $b_2$ (denoted by $b_1 \succ b_2$) if there is a vertex $v \in b_1$ such that $\mathcal{S}(b_2) \prec v \prec \mathcal{S}(b_2)$, or there is a vertex $u \in b_2$ such that $\mathcal{S}(b_1) \prec v \prec \mathcal{S}(b_1)$. If $b_1$ does not touch $b_2$, we write $b_1 \not\succ b_2$. Clearly, the relation is symmetric, i.e., if $b_1 \succ b_2$ then $b_2 \succ b_1$.

For some intuition, consider a drawing of $G$ which orders vertices w.r.t. $\prec$ in a line. Project every edge on that line as well. Then two blocks touch each other if they have a common segment on that projection.

Algorithm and Proof Sketch: Before delving into details, we provide the main ideas behind our algorithm. We can think about the update problem on DAGs as follows. Our goal is to compute a feasible update order for the (out-)edges of the graph. There are at most $k$ flows to be updated for each edge, resulting in $k!$ possible orders and hence a brute force complexity of $O(k!|G|)$ for the entire problem. We can reduce this complexity by considering blocks instead of edges. Let $\text{TouchSeq}(b)$ contain all feasible update sequences for the blocks that touch $b$: still a (too) large number, but let us consider them for now. For two distinct blocks $b, b'$, we say that two sequences $s \in \text{TouchSeq}(b), s' \in \text{TouchSeq}(b')$ are consistent, if the order of any common pair of blocks is the same in both $s, s'$. It is clear that if for some block $b$, $\text{TouchSeq}(b) = \emptyset$, there is no feasible update sequence for $G$: $b$ cannot be updated.

We now create a graph $H$ whose vertices correspond to elements of $\text{TouchSeq}(b)$, for all $b \in B$. Connect all pairs of vertices originating from the same $\text{TouchSeq}(b)$. Connect all pairs of vertices if they correspond to inconsistent elements of different $\text{TouchSeq}(b)$. If (and
only if) we find an independent set of size $|B|$ in the resulting graph, the update orders corresponding to those vertices are mutually consistent: we can update the entire network according to those orders. In other words, the update problem can be reduced to finding an independent set in the graph $H$.

However, there are two main issues with this approach. First, $H$ can be very large. A single $\text{TouchSeq}(b)$ can have exponentially many elements. Accordingly, we observe that we can assume a slightly different perspective on our problem: we linearize the lists $\text{TouchSeq}(b)$ and define them sequentially, bounding their size by a function of $k$ (the number of flows). The second issue is that finding a maximum independent set in $H$ is hard. The problem is equivalent to finding a clique in the complement of $H$, a $|B|$-partite graph where every partition has bounded cardinality. We prove that for an $n$-partite graph where every partition has bounded cardinality, finding an $n$-clique is NP-complete. So, in order to solve the problem, we either should reduce the number of partitions in $H$ (but we cannot) or modify $H$ to some other graph, further reducing the complexity of the problem. We do the latter by trimming $H$ and removing some extra edges, turning the graph into a very simple one: a graph of bounded path width. Then, by standard dynamic programming, we find the independent set of size $|B|$ in the trimmed version of $H$: this independent set matches the independent set $I$ of size $|B|$ in $H$ (if it exists). At the end, reconstructing a correct update order sequence from $I$ needs some effort. As we have reduced the size of $\text{TouchSeq}(b)$ and while not all possible update orders of all blocks occur, we show that they suffice to cover all possible feasible solutions. We provide a way to construct a valid update order accordingly. \(\square\)

With these intuitions in mind, we now present a rigorous analysis. Let $\pi_{S_1} = (a_1, \ldots, a_{|S_1|})$ and $\pi_{S_2} = (a'_1, \ldots, a'_{|S_2|})$ be permutations of sets $S_1$ and $S_2$. We define the core of $\pi_{S_1}$ and $\pi_{S_2}$ as $\text{core}(\pi_{S_1}, \pi_{S_2}) := S_1 \cap S_2$. We say that two permutations $\pi_1$ and $\pi_2$ are consistent, $\pi_1 \approx \pi_2$, if there is a permutation $\pi$ of symbols of $\text{core}(\pi_1, \pi_2)$ such that $\pi$ is a subsequence of both $\pi_1$ and $\pi_2$.

The Weak Dependency Graph, simply called dependency graph in the following, of a set of permutations is a labelled graph defined recursively as follows. The dependency graph of a single permutation $\pi = (a_1, \ldots, a_t)$, denoted by $G_{\pi}$, is a directed path $v_1, \ldots, v_t$, and the label of the vertex $v_i \in V(G_{\pi})$ is the element $a$ with $\pi(a) = i$. We denote by $\text{Labels}(G_{\pi})$ the set of all labels of $G_{\pi}$.

Let $G_{II}$ be a dependency graph of the set of permutations $\Pi$ and $G_{IV}$ the dependency graph of the set $\Pi'$. Then, their union (by identifying the same vertices) forms the dependency graph $G_{\Pi \cup \Pi'}$ of the set $\Pi \cup \Pi'$. Note that such a dependency graph is not necessarily acyclic.

We call a permutation $\pi$ of blocks of a subset $\mathcal{B}' \subseteq \mathcal{B}$ congestion free, if the following holds: it is possible to update the blocks in $\pi$ in the graph $G_{\mathcal{B}}$ (the graph on the union of blocks in $\mathcal{B}$), in order of their appearance in $\pi$, without violating any edge capacities in $G_{\mathcal{B}}$. Note that we do not respect all conditions of our Consistency Rule (definition 6.2.10) here.

**Lemma 6.8.1.** Let $\pi$ be a permutation of the set blockset $1 \subseteq \mathcal{B}$. Whether $\pi$ is congestion free can be determined in time $O(|\mathcal{B}_1| \cdot |G|)$.

**Proof.** In the order of $\pi$, perform Algorithm 7. If it fails, i.e., if it violates congestion freedom for some edges, $\pi$ is not a congestion free permutation. \(\square\)

The smaller relation defines a total order on all blocks in $G$. Let $\mathcal{B} = \{b_1, \ldots, b_{|\mathcal{B}|}\}$ and suppose the order is $b_1 < \ldots < b_{|\mathcal{B}|}$.
6. Rerouting Problem

\[ G_{\{\pi_{\text{blue}}, \pi_{\text{green}}, \pi_{\text{red}}\}} \]

\[ \pi_{\text{blue}} = (v_7, c, a, v_2) \]
\[ \pi_{\text{green}} = (v_6, b, c, v_1) \]
\[ \pi_{\text{red}} = (v_3, v_4, a, b, v_5) \]

**Figure 6.5.** Example: The weak dependency graph of three pairwise consistent permutations \( \pi_{\text{blue}}, \pi_{\text{green}} \) and \( \pi_{\text{red}} \). Each pair of those permutation has exactly one vertex in common and with this the cycle \((a, b, c)\) is created. With such cycles being possible a weak dependency graph does not necessarily contain sink vertices. To get rid of them, we certainly need some more refinements.

We define an auxiliary graph \( H \) which will help us find a suitable dependency graph for our network. We first provide some high-level definitions relevant to the construction of the graph \( H \) only. Exact definitions will follow in the construction of \( H \), and will be used throughout the rest of this paper.

Recall that \( \mathcal{B} \) is the set of all blocks in \( G \). We define another set of blocks \( \mathcal{B}' \) which is initialized as \( \mathcal{B} \); the construction of \( H \) is iterative, and in each iteration, we eliminate a block from \( \mathcal{B}' \). At the end of the construction of \( H \), \( \mathcal{B}' \) is empty. For every block \( b \in \mathcal{B}' \), we also define the set \( \text{TouchingBlocks}(b) \) of blocks which touch the block \( b \). Another set which is defined for every block \( b \) is the set \( \text{PermutList}(b) \); this set actually corresponds to a set of vertices, each of which corresponds to a valid congestion free permutation of blocks in \( \text{TouchingBlocks}(b) \). Clearly if \( \text{TouchingBlocks}(b) \) does not contain any congestion-free permutation, then \( \text{PermutList}(b) \) is an empty set. As we already mentioned, every vertex \( v \in \text{PermutList}(b) \) comes with a **label** which corresponds to some congestion-free permutation of elements of \( \text{TouchingBlocks}(b) \). We denote that permutation with \( \text{Label}(v) \).

**Construction of \( H \):** We recursively construct a labelled graph \( H \) from the blocks of \( G \) as follows.

i) Set \( H := \emptyset \), \( \mathcal{B}' := \mathcal{B} \), \( \text{PermutList} := \emptyset \).

ii) For \( i := 1, \ldots, |\mathcal{B}| \) do

1. Let \( b := b_{|\mathcal{B}|+i-1} \).
2. Let \( \text{TouchingBlocks}(b) := \{b_1, \ldots, b'_\ell\} \) be the set of blocks in \( \mathcal{B}' \) touched by \( b \).
3. Let \( \pi := \{\pi_1, \ldots, \pi_\ell\} \) be the set of congestion free permutations of \( \text{TouchingBlocks}(b) \).
4. Set \( \text{PermutList}(b) := \emptyset \).
5. For \( i \in [\ell] \) create a vertex \( v_{\pi_i} \) with \( \text{Label}(v_{\pi_i}) = \pi_i \) and set \( \text{PermutList}(b) := \text{PermutList}(b) \cup v_{\pi_i} \).
6. Set \( H := H \cup \text{PermutList}(b) \).
7. Add edges between all pairs of vertices in \( H[\text{PermutList}(b)] \).
8. Add an edge between every pair of vertices \( v \in H[\text{PermutList}(b)] \) and \( u \in V(H) - \text{PermutList}(b) \) if the labels of \( v \) and \( u \) are inconsistent.
9 Set $B' := B' - b$.

Figure 6.6: Example: The graph $H$ consists of vertex sets PermutList($b_i$), $i \in [|B|]$, where each such partition contains all congestion free sequences of the at most $k$ iteratively chosen touching blocks. In the whole graph, we then create edges between the vertices of two such partitions if and only if the corresponding sequences are inconsistent with each other, as seen in the three highlighted sequences. Later we will distinguish between such edges connecting vertices of neighbouring partitions (w.r.t. the topological order of their corresponding blocks), PermutList($b_i$) and PermutList($b_{i+1}$), and partitions that are further away, PermutList($b_i$) and PermutList($b_j$). Edges of the latter type, depicted as red in the figure, are called long edges and will be deleted in the trimming process of $H$.

**Lemma 6.8.2.** For Item (ii) of the construction of $H$, $t \leq k$ holds.

**Proof.** Suppose for the sake of contradiction that $t$ is bigger than $k$. So there are $c$-blocks $b, b'$ (where $b_{|B|-i+1}$ corresponds to a flow pair different from $c$) that touch $b_{|B|-i+1}$. But then one of $\mathcal{I}(b)$ or $\mathcal{I}(b')$ is strictly larger than $\mathcal{I}(b_{|B|-i+1})$. This contradicts our choice of $b_{|B|-i+1}$ in that we deleted larger blocks from $B'$ in Item (ii9).

**Lemma 6.8.3** (touching lemma). Let $b_{j_1}, b_{j_2}, b_{j_3}$ be three blocks (w.r.t. $<$) where $j_1 < j_2 < j_3$. Let $b_z$ be another block such that $z \notin \{j_1, j_2, j_3\}$. If in the process of constructing $H$, $b_z$ is in the touch list of both $b_{j_1}$ and $b_{j_3}$, then it is also in the touch list of $b_{j_2}$.

**Proof.** Let us suppose that $\mathcal{I}(b_{j_1}) \neq \mathcal{I}(b_{j_2}) \neq \mathcal{I}(b_{j_3})$. We know that $\mathcal{I}(b_z) < \mathcal{I}(b_{j_1})$ as otherwise, in the process of creating $H$, we eliminate $b_z$ before we process $b_{j_1}$; it would hence not appear in the touch list of $b_{j_1}$. As $b_z > b_{j_3}$, there is a vertex $v \in b_z$ where $\mathcal{I}(b_{j_3}) < v$. But by our choice of elimination order: $\mathcal{I}(b_z) < \mathcal{I}(b_{j_3}) < v < \mathcal{I}(b_z)$, and on the other hand: $\mathcal{I}(b_z) < \mathcal{I}(b_{j_1}) < \mathcal{I}(b_{j_3})$. Thus, $\mathcal{I}(b_z) < \mathcal{I}(b_{j_2}) < \mathcal{I}(b_{j_3})$, and therefore $b_z$ touches $b_{j_2}$. If some of the start vertices are the same, a similar case distinction applies.
6. Rerouting Problem

Figure 6.7: Example: Select one of the permutations of length at most k from every PermutList(b). These permutations obey the touching lemma. Taking the three permutations from the example in Figure 6.5, we can see that the touching lemma forces a to be in the green permutation as well. Assuming consistency, this would mean a to come before b and after c. Hence a <π green b and b <π green a, a contradiction. So if our permutations are derived from H and are consistent, we will show that cycles cannot occur in their weak dependency graph.

For an illustration of the property described in the touching lemma, see Figure 6.7: it refers to the weak dependency graph of Figure 6.5. This example also points out the problem with directed cycles in the weak dependency graph and the property of touching lemma.

We prove some lemmas in regard to the dependency graph of elements of H, to establish the base of the inductive proof for the Lemma 6.8.5.

We begin with a simple observation on the fact that a permutation π induces a total order on the elements of S.

Observation 6.8.1. Let π be a permutation of a set S. Then the dependency graph Gπ does not contain a cycle.

Lemma 6.8.4. Let π1, π2 be permutations of sets S1, S2 such that π1, π2 are consistent. Then the dependency graph Gπ1∪π2 is acyclic.

Proof. For the sake of contradiction suppose there is a cycle C in Gπ1∪π2. By Observation 6.8.1 this cycle must contain vertices corresponding to elements of both S1 and S2. Let a be the least element of S1 with respect to π1 such that va ∈ V(C). As C is a cycle there is a vertex vb with b ∈ S1 ∪ S2 such that the edge (vb, va) is an edge of C. By our choice of a, b is not contained in S1. Hence, since the edge (vb, va) exists, a ∈ S1 ∩ S2. Similarly we can consider the least element c ∈ S2 in C and its predecessor d ∈ S1 \ S2 along the cycle. Again the edge (vd, vc) exists and thus c ∈ S1 ∩ S2. Now we have d < a in π2, but a < d in π1 contradicting the consistency of π1 and π2.

In the next lemma, we need a closure of the dependency graph of permutations which we define as follows.

Definition 6.8.1 (Permutation Graph Closure). The Permutation Graph Closure, or simply closure, of a permutation π is the graph Gπ+ obtained from taking the transitive closure of Gπ, i.e. its vertices and labels are the same as Gπ and there is an edge (u, v) in Gπ+ if there is a path starting at u and ending at v in Gπ. Similarly the Permutation Graph Closure of
6. Rerouting Problem

A set of permutations $\Pi = \{\pi_1, \ldots, \pi_n\}$ is the graph obtained by taking the union of $G^+_{\pi_i}$’s (for $i \in [n]$) by identifying vertices of the same label.

In the above definition note that if $\Pi$ is a set of permutations then $G_\Pi \subseteq G^+_\Pi$.

The following lemma generalizes Lemma 6.8.4 and observation 6.8.1 and uses them as the base of its inductive proof.

**Lemma 6.8.5.** Let $I = \{v_{\pi_1}, \ldots, v_{\pi_\ell}\}$ be an independent set in $H$. Then the dependency graph $G_\Pi$, for $\Pi = \{\pi_1, \ldots, \pi_\ell\}$, is acyclic.

**Proof.** Instead of working on $G_\Pi$, we can work on its closure $G^+_\Pi$ as defined in Definition 6.8.1.

First we observe that every edge in $G_\Pi$ also appears in $G^+_\Pi$, so if there is a cycle in $G_\Pi$, the same cycle exists in $G^+_\Pi$.

We prove that there is no cycle in $G^+_\Pi$. By Lemma 6.8.4 and observation 6.8.1 there is no cycle of length at most 2 in $G^+_\Pi$, otherwise there is a cycle in $G_\Pi$ which consumes at most two consistent permutations.

For the sake of contradiction, suppose $G^+_\Pi$ has a cycle and let $C = (a_1, \ldots, a_n) \subseteq G^+_\Pi$ be the shortest cycle in $G^+_\Pi$. By Lemma 6.8.4 and observation 6.8.1 we know $n \geq 3$.

In the following, because we work on a cycle $C$, whenever we write any index $i$ we consider it w.r.t. its cyclic order on $C$, in fact $i \mod |C| + 1$. So for example $i = 0$ and $i = n$ are identified as the same indices, $i = n + 1, i = 1$ as well and so on.

Recall the construction of the dependency graph where every vertex $v \in C$ corresponds to some block $b_v$. In the remainder of this proof we do not distinguish between the vertex $v$ and the block $b_v$.

Let $\pi_v$ be the label of a given vertex $v \in I$. For each edge $e = (a_i, a_{i+1}) \in C$, there is a permutation $\pi_v$ such that $(a_i, a_{i+1})$ is a subsequence of $\pi_v$, and additionally the vertex $v_i$ is in the set $I$. So there is a block $b^i$ such that $\pi_v$ is a permutation of the set TouchingBlocks($b^i$).

The edge $e = (a_i, a_{i+1})$ is said to represent $b^i$, and we call it the representative of $\pi_v$. For each $i$ we fix one block $b^i$ which is represented by the edge $(a_i, a_{i+1})$ (note that one edge can represent many blocks, but here we fix one of them). We define the set of those blocks as $B^I = \{b^1, \ldots, b^\ell\}$ and state the following claim.

**Claim 4.** For every two distinct vertices $a_i, a_j \in C$, either there is no block $b \in B^I$ such that $a_i, a_j \in \text{TouchingBlocks}(b)$ or if $a_i, a_j \in \text{TouchingBlocks}(b)$ then $(a_i, a_j)$ or $(a_j, a_i)$ is an edge in $C$. Additionally $|B^I| = |C|$.

**Proof.** Suppose there is a block $b \in B^I$ such that $a_i, a_j \in \text{TouchingBlocks}(b)$. Then in $E(G^+_\Pi)$ there is an edge $e_1 = (a_i, a_j)$ or $e_2 = (a_j, a_i)$. If either of $e_1, e_2$ is an edge in $C$ then we are done. Otherwise if $e_1 \in E(G^+_\Pi)$ then the cycle on the vertices $a_1, \ldots, a_i, a_j, \ldots, a_n$ is shorter than $C$ and if $e_2 \in E(G^+_\Pi)$ then the cycle on the vertices $a_1, \ldots, a_j$ is shorter than $C$.

Both cases contradict the assumption that $C$ is the shortest cycle in $G^+_\Pi$. For the second part of the claim it is clear that $|B^I| \leq |C|$, on the other hand if both endpoints of an edge $e = (a_i, a_{i+1}) \in C$ appear in TouchingBlocks($b$) and TouchingBlocks($b'$) for two different blocks $b, b' \in B^I$ then, by our choice of the elements of $B^I$, at least one of them (say $b$) has a representative $e' \neq e$. But, then there is a vertex $a_j \in V(e')$ such that $a_j \neq a_i, a_j \neq a_{i+1}$. But by the first part this cannot happen, so we have $|C| \leq |B^I|$ and the second part of the claim follows.

\[\Box\]
6. Rerouting Problem

By the above claim we have $\ell = n$. W.l.o.g. suppose $b^1 < b^2 < \ldots < b^n$. There is an $i \in [n]$ such that $(a_{i-1}, a_i)$ represents $b^1$, we fix this $i$.

**Claim 5.** If $(a_{i-1}, a_i)$ represents $b^1$ then $(a_{i-2}, a_{i-1})$ represents $b^2$.

**Proof.** By claim 4 there is a block $b^t$ which is represented by $(a_{i-2}, a_{i-1})$. We also have $b^1 < b^2 \leq b^t$ hence by touching lemma, $a_{i-1}$ appears in TouchingBlocks($b^2$). But then by Claim 4 either $a_{i-1}$ is in TouchingBlocks($b^2$) or $a_{i-2} \in$ TouchingBlocks($b^2$), by the former case we have $b^1 = b^2$ which is a contradiction to the assumption that $b^1 < b^2$. In the latter case we have $t = 2$ which proves the claim. \(\square\)

Similarly we can prove the endpoints of the edges, that have $a_i$ as their head, are in $b^2$.

**Claim 6.** If $(a_{i-1}, a_i)$ represents $b^1$ then $(a_i, a_{i+1})$ represents $b^2$.

**Proof.** By claim 4 there is a block $b^t$ such that $(a_i, a_{i+1})$ represents $b^t$. We also have $b^1 < b^2 \leq b^t$ thus by the touching lemma, $a_i$ appears in TouchingBlocks($b^2$). But, then by claim 4 either $a_{i-1}$ or $a_{i+1}$ is in TouchingBlocks($b^2$). In the former case we have $b^1 = b^2$ which is a contradiction to the assumption that $b^1 < b^2$. In the latter case we have $t = 2$ which proves the claim. \(\square\)

By our claims 5 and 6 we have that both $(a_{i-2}, a_{i-1})$ and $(a_i, a_{i+1})$ represent $b^2$ hence by claim 4 they are the same edge. Thus there is a cycle on the vertices $a_{i-1}, a_i$ in $G^{\Pi}_I$ and this gives a cycle in $G^{\Pi}_I$ on at most 2 consistent permutations which is a contradiction according to Lemma 6.8.4. \(\square\)

The following lemma establishes the link between independent sets in $H$ and feasible update sequences of the corresponding update flow network $G$.

**Lemma 6.8.6.** There is a feasible sequence of updates for an update network $G$ on $k$ flow pairs, if and only if there is an independent set of size $|B|$ in $H$. Additionally if the independent set $I \subseteq V(H)$ of size $|B|$ together with its vertex labels are given, then there is an algorithm which can compute a feasible sequence of updates for $G$ in $O(k \cdot |G|)$.

**Proof.** First we prove that if there is a sequence of feasible updates $\mathcal{R}$, then there is an independent set of size $|B|$ in $H$. Suppose $\mathcal{R}$ is a feasible sequence of updates of blocks. For a block $b$, recall that $\text{TouchingBlocks}(b) = \{b'_1, \ldots, b'_l\}$ is the set of remaining (not yet processed) blocks that touch $b$. Let $\pi_b$ be the reverse order of updates of blocks in $\text{TouchingBlocks}(b)$ w.r.t. $\mathcal{R}$. In fact, if $\mathcal{R}$ updates $b'_1$ first, then $b'_2$, then $b'_3, \ldots, b'_l$, then $\pi_b = b'_l \ldots b'_1$.

For every two blocks $b, b' \in I$, we have $\pi_b \approx \pi_{b'}$. From every set of vertices $\text{PermutList}(b)$, for $b \in B$, let $v_b$ be a vertex such that $\text{Label}(v_b)$ is a subsequence of $\pi_b$. Recall that, the labels of vertices in $\text{PermutList}(b)$ are all possible congestion free permutations of blocks that touch $b$ in the remaining set of blocks $B'$ during the construction of $H$. So the vertex $v_b$ exists. Put $v_b$ in $I$. The labels of every pair of vertices in $I$ are consistent, as their super-sequences were consistent, so $I$ is an independent set and furthermore $|I| = |B|$.

118
6. Rerouting Problem

For the other direction, suppose there is an independent set of vertices $I$ of size $|B|$ in $H$. It is clear that for every block $b \in B$, there is exactly one vertex $v_b \in I \cap \text{PermutList}(b)$. Let us define the dependency graph of the set of labels (permutations) $\Pi = \{\text{Label}(v_b) \mid b \in B, v_b \in I\}$ as the dependency graph $D := G_\Pi$. $I$ is an independent set and thus every pair of labels of vertices in $I$ are consistent, hence by Lemma 6.8.5 we know that $D$ is a DAG, and thus it has a sink vertex. We update blocks which correspond to sink vertices of $D$ in parallel by applying Algorithm 7 and we remove those vertices from $D$ after they are updated. Then we proceed recursively, until there is no vertex in $D$, similarly to algorithm 6 for 2-flows. We claim that this gives a valid sequence of updates for all blocks.

Suppose there is a sink vertex whose corresponding block $b$ cannot be updated. There are two reasons preventing us from updating a block:

1. Its update stops the flow between some source and terminal. So afterwards there is no transient flow on the active edges.

2. There is an edge $e \in E(b)$ (denoting the set of edges in the block $b$) which cannot be activated because this would imply routing along it and produce congestion.

The first will never be the case since we apply Algorithm 7 to update. So suppose there is such an edge $e$. Edge $e$ cannot be updated because some other blocks are incident to $e$ and route flows: updating $b$ would violate a capacity constraint. There are may be some blocks which are incident to $e$ but are not updated yet. These blocks would not effect the rest of our reasoning and we restrict ourselves to those blocks which have been updated already by our algorithm. Otherwise, if there is no such block, the label corresponding to $b$ is an invalid congestion free label. We will denote the set of the blocks preventing the update of $e$ by $B_e$.

Suppose the blocks in $B_e$ are updated in the order $b'_1, b'_2, \ldots, b'_\ell$ by the above algorithm. Among $b, b'_1, \ldots, b'_\ell$, there is a block $b'$ which is the largest one (w.r.t. $<$). In the construction of $H$, we know that $\text{PermutList}(b') \neq \emptyset$, as otherwise $I$ was not of size $|B|$. Also suppose $v \in \text{PermutList}(b') \cap I$, then in the label of $v$, we have a subsequence $b'_1, \ldots, b''_{\ell+1}$ such that $b''_{\ell+1} \in \{b, b'_1, \ldots, b'_\ell\}$: in the iteration where we create $\text{PermutList}(b')$, $b'$ touches all those blocks. We claim that the permutations $\pi_1 = b'_1, \ldots, b''_{\ell+1}$ and $\pi_2 = b'_1, \ldots, b'_\ell, b$ are exactly the same, which would contradict our assumption that $e$ cannot be updated: $\pi_1$ is a subsequence of the congestion free permutation $\text{Label}(v)$.

Suppose $\pi_1 \neq \pi_2$, then there are two blocks $b''_1, b''_2$ with $\pi_1(b''_2) < \pi_1(b''_1)$ and $\pi_2(b''_2) < \pi_2(b''_1)$. Since both, $b''_2$ and $b''_1$, will appear in Label($v$), there is a directed path from $b''_2$ to $b''_1$ in $D$. Then our algorithm cannot choose $b''_2$ as a sink vertex before updating $b''_1$: a contradiction. At the end recall that algorithm 6 uses Algorithm 7 as a subroutine and this guarantees the existence of transient flow if we do not violate the congestion of edges. Hence, the sequence of updates we provided by deleting the sink vertices, is a valid sequence of updates if $I$ is an independent set of size $|B|$.

On the other hand, in the construction of $H$, all congestion free routings are already given and the runtime of algorithm 6 is linear in the size of the dependency graph: If $I$ is given, the number of blocks is at most $k$ times larger than the original graph or $|G_\Pi| = O(k \cdot |G|)$; therefore, we can compute the corresponding update sequence in $O(k \cdot |G|)$ as claimed. \qed

With Lemma 6.8.6, the update problem boils down to finding an independent set of size $|B|$ in $H$. However, this reduction does not suffice yet to solve our problem in polynomial time, as we will show next.
6. Rerouting Problem

Finding an independent set of size $|B|$ in $H$ is equivalent to finding a clique of size $|B|$ in its complement. The complement of $H$ is a $|B|$-partite graph where every partition has cardinality $\leq k!$. In general, it is computationally hard to find such a clique. This is shown in the following lemma. Note that the lemma is not required for the analysis of our algorithm, but constitutes an independent result and serves to round off the discussion.

Lemma 6.8.7. Finding an $m$-clique in an $m$-partite graph, where every partition has cardinality at most 3, is NP-hard.

Proof. We provide a polynomial time reduction from 3-SAT. Let $C = C_1 \land C_2 \land \ldots \land C_m$ be an instance of 3-SAT with $n$ variables $X_1, \ldots, X_n$. We denote positive appearances of $X_i$ as a literal $x_i$ and negative appearance as a literal $\bar{x}_i$ for $i \in [m]$. So we have at most 2$n$ different literals $x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n$. Create an $m$-partite graph $G$ as follows. Set $G$ to be an empty graph. Let $C_i = \{l_{i1}, l_{i2}, l_{i3}\}$ be a clause for $i \in [m]$, then add vertices $v_{l_{i1}}^i, v_{l_{i2}}^i, v_{l_{i3}}^i$ to $G$ as partition $p_i$. Note that $l_{i1} = x_t$ or $l_{i1} = \bar{x}_t$ for some $t \in [n]$. Add an edge between each pair of vertices $v_{l_{ij}}^i, v_{l_{ij}}^j$ for $i, j \in [m], i \neq j$ if $x = x_t$ for some $t \in [n]$ and $y \neq \bar{x}_t$ or if $x = \bar{x}_t$ and $y \neq x_t$. It is clear that $G$ now is an $m$-partite graph with exactly 3 vertices in each partition.

Claim 7. There is a satisfying assignment $\sigma$ for $C$ if, and only if, there is an $m$-clique in $G$.

Proof. Define a vertex set $K = \emptyset$. Let $\sigma$ be a satisfying assignment. Then from each clause $C_i$ for $i \in [m]$, there is a literal $l_{ij}$ which is set to true in $\sigma$. We take all vertices of $G$ of the form $v_{l_{ij}}^i$ and add it to $K$. The subgraph $G[K]$ forms a clique of size $m$. On the other hand suppose we have an $m$-clique $K_m$ as a subgraph of $G$. Then, clearly from each partition $p_i$, there exists exactly one vertex $v_{l_{ij}}^i$ which is in $K_m$. We set the literal $l_{ij}$ to true. This gives a valid satisfying assignment for $C$.

Now we trim $H$ to avoid the above problem. Again we will use the special properties of the touching relation of blocks. We say that some edge $e \in E(H)$ is long, if one end of $e$ is in $\text{PermutList}(b_j)$, and the other in block type $\text{PermutList}(b_j)$ where $j > i + 1$. The length of $e$ is $j - i$. Delete all long edges from $H$ to obtain the graph $R_H$. In other words we can construct $R_H$ directly, similar to $H$, without adding long edges. In the following we first prove that in linear time we can construct the graph $R_H$. Second we show that if there is an independent set $I$ of size exactly $|B|$ in $R_H$ then $I$ is also an independent set of $H$.

Lemma 6.8.8. There is an algorithm which computes $R_H$ in time $O((k \cdot k!)^2 |G|)$.

Proof. The algorithm is similar to the construction of $H$. For completeness we repeat it here and then we prove it takes time proportional to $(k \cdot k!)^2 |G|$. The algorithm is provided in the following, we present the analysis. The only difference between the following algorithm and the construction of $H$ is line ii8, where we add at most $O(k^2)$ edges to the graph. As there are at most $|B|$ steps in the algorithm, this shows that the size of $R_H$ is at most $O(|B| \cdot k!^2)$. Moreover, as there are at most $O(k |E(G)|)$ blocks in $G$, the total size of $R_H$ w.r.t. $G$ is at most $O(k \cdot k!^2 \cdot |G|)$. The computations in all other lines except for line ii3 are linear in $k$, hence we only show that the total amount of computations in line ii3 is
6. Rerouting Problem

in $O(k! \cdot |G|)$. We know that every edge appears in at most $k$ blocks, hence the algorithm provided in Lemma 6.8.1, for each edge, runs at most $k$ times and as per individual round of that algorithm, takes $O(k \cdot |G|)$. Since there are $k!$ possible permutations for each block, this yields a running time of $O(k^2 \cdot k! \cdot |G|)$. So all in all, the construction of $R_H$ takes at most $O((k \cdot k!)^2 |G|)$ operations.

Algorithm 9. Construction of $R_H$

**Input:** Update Flow Network $G$

i Set $H := \emptyset$, $B' := B$, PermutList := $\emptyset$.

ii For $i := 1, \ldots, |B|$ do

1. Let $b := b_{|B|-i+1}$.
2. Let TouchingBlocks$(b) := \{b'_1, \ldots, b'_1\}$ be the set of blocks in $B'$ which touch $b$.
3. Let $\pi := \{\pi_1, \ldots, \pi_\ell\}$ be the set of congestion free permutations of TouchingBlocks$(b)$, compute $\pi$ by the algorithm provided in 6.8.1.
4. Set PermutList$(b) := \emptyset$.
5. For $i \in [\ell]$ create a vertex $v_{\pi_i}$ with $\operatorname{Label}(v_{\pi_i}) = \pi_i$ and set PermutList$(b) := \text{PermutList}(b) \cup v_{\pi_i}$.
6. Set $H := H \cup \text{PermutList}(b)$.
7. Add edges between all pairs of vertices in $H[\text{PermutList}(b)]$.
8. Add an edge between every pair of vertices $v \in H[\text{PermutList}(b)]$ and $u \in \text{PermutList}(b_{|B|-i+2})$ if the labels of $v$ and $u$ are inconsistent and if $b_{|B|-i+2}$ exists.

In the above lemma note that we can run the algorithm in parallel. Hence using parallelization we could get much more better running time in practice.

**Lemma 6.8.9.** $H$ has an independent set $I$ of size $|B|$ if, and only if, $I$ is also an independent set of size $|B|$ in $R_H$.

**Proof.** One direction is clear: if $I$ is an independent set of size $|B|$ in $H$, then it is an independent set of size $|B|$ in $R_H$. On the other hand, suppose $I$ is an independent set of size $|B|$ in $R_H$. Then for the sake of contradiction, suppose there are vertices $u, v \in I$ and an edge $e = \{u, v\} \in E(H)$, where $e$ has the shortest length among all possible long edges in $H[I]$. Let us assume that $u \in \text{PermutList}(b_i), v \in \text{PermutList}(b_j)$ where $j > i + 1$. Suppose from each PermutList$(b_i)$ for $i \leq \ell \leq j$, we have $v_{b_\ell} \in I$, where $v_{b_i} = u, v_{b_j} = v$. Clearly as $I$ is of size $|B|$ there should be exactly one vertex from each PermutList$(b_i)$. We know core$(\operatorname{Label}(u), \operatorname{Label}(v)) \neq \emptyset$ as otherwise the edge $e = \{u, v\}$ was not in $E(H)$. On the other hand, as $e$ is the smallest long edge which connects vertices of $I$, then there is no long edge between $v_{b_i}$ and $v_{b_{j-1}}$ in $H$. That means $\operatorname{Label}(v_{b_i}) = \operatorname{Label}(v_{b_{j-1}})$ but then as $\operatorname{Label}(v_{b_i}) \neq \operatorname{Label}(v_{b_j})$ and by touching lemma we know that core$(\operatorname{Label}(u), \operatorname{Label}(v)) \subseteq \operatorname{Label}(v_{b_{j-1}})$, so $\operatorname{Label}(v_{b_i}) \neq \operatorname{Label}(v_{b_{j-1}})$. Therefore, there is an edge between $v_{b_j}$ and $v_{b_{j-1}}$: a contradiction, by our choice of $I$ in $R_h$. \qed

121
6. Rerouting Problem

$R_H$ is a much simpler graph compared to $H$, which helps us find a large independent set of size $|B|$ (if exists). We have the following lemma.

**Lemma 6.8.10.** There is an algorithm that finds an independent set $I$ of size exactly $|B|$ in $R_H$ if such an independent set exists; otherwise it outputs that there is no such an independent set. The runtime of this algorithm is $O(|R_H|)$.

**Proof.** We find an independent set of size $|B|$ (or we output there is no such set) by dynamic programming. For this purpose we define a function $f : [|B|] \times V(R_H) \rightarrow 2^{V(R_H)}$ which is presented in detail in the algorithm below. Before providing said algorithm we explain it in plain text. It is a straightforward dynamic program: start from the left most groups of vertices in $R_H$ (one extreme side of $R_H$). Consider every vertex as part of the independent set and build the independent set bottom up on those groups. We omit the proof of correctness and the exact calculation of the running time as it is clear from the algorithm.

**Algorithm 10. Finding an Independent Set of Size $|B|$ in $R_H$**

**Input:** $R_H$

a) Set $f(i, v) := \emptyset$ for all $i \in [|B|], v \in V(R_H)$.

b) Set $f(1, v) := v$ for all $v \in \text{PermutList}(b_1)$.

c) For $2 \leq i \leq [|B|]$ do

i. For all $v \in \text{PermutList}(b_i)$

A. If there is a vertex $u \in \text{PermutList}(b_{i-1})$ and $|f(i-1, u)| = i - 1$ and $\{u, v\} \notin E(R_H)$ then $f(i, v) := f(i - 1, u) \cup \{v\}$,

B. otherwise set $f(i, v) := \emptyset$

d) If $\exists v \in \text{PermutList}(b_{|B|})$ where $|f([|B|], v)| = |B|$ then output $f([|B|], v)$,

e) otherwise output there is no such independent set.

Our main theorem is now a corollary of the previous lemmas and algorithms.

**Theorem 6.8.1.** There is a linear time FPT algorithm for the network update problem on an acyclic update flow network $G$ with $k$ flows (the parameter), which finds a feasible update sequence, whenever it exists; otherwise it outputs that there is no feasible solution for the given instance. The algorithm runs in time $O(2^{O(k \log k)} |G|)$.

**Proof.** First construct $R_H$ using algorithm 9, then find the independent set $I$ of size $|B|$ in $R_H$ using Algorithm 10. If there is no such independent set $I$, then we output that there is no feasible update solution for the given network; this is a consequence of Lemma Lemmas 6.8.6 and 6.8.9. On the other hand, if there is such an independent set $I$, then one can construct the corresponding dependency graph and update all blocks, using the algorithm provided in the proof of Lemma Lemma 6.8.6. The dominant runtime term in the above algorithms is $O(k^2 \cdot k!^2 \cdot |G|)$ (from Lemma 6.8.10), which proves the claim of the theorem.

Note that the above theorem only provides a feasible solution verias the **Theorem 6.6.1** provides an optimum solution. Already for 3-flows we do not know whether there is an optimal algorithm.
6.9. Conclusion and Future Work

We studied flow rerouting in asynchronous networks. We first formalize the problem and encapsulated most of the previous works in our model. We saw that the problem is hard already for two flows in general graphs. We considered a more restricted class of acyclic graphs. There we show that the problem remains hard for general $k$ flows but for two flows it is possible to not only decide the feasibility in linear time but also we can find the minimum rerouting schedule. Eventually, we provide a fixed parameter tractable algorithm (FPT w.r.t. $k$, the number of flows) for acyclic graphs.

There are two interesting general directions of research on a restricted class of digraphs. One direction concerns about the sparsity of the graph which we did not completely address it in this chapter. It is interesting to know the complexity of the problem in classes of planar, bounded genus, excluded minor and Nowhere Crownful [97] digraphs.

Another direction of research is to attack the problem on the graph classes with a small complexity w.r.t. some width parameter. E.g. acyclic graphs, graphs of bounded feedback vertex set, graphs of bounded DAG-width[11, 21], etc. We studied the problem in this direction and we established a classification of the problem on acyclic graphs. It is interesting to know what happens in bigger graph classes such as the class of graphs of bounded DAG-width or bounded directed tree-width.

We did not provide a full classification for acyclic graphs. We do not know if it is possible to improve the running time of our algorithm. It is quite interesting to know if the problem admits $2^{O(k)} |G|$ FPT algorithm. The last but not least question is to find optimum algorithm w.r.t. the number of rounds. For $k > 2$, our algorithm provides a feasible solution in a reasonable time, but it does not give any approximation guarantee for the number of rounds. Already for 3 flow pairs we do not know what is an optimum algorithm or we do not even know what is a good approximation algorithm to minimize the number of rounds for a feasible instance.


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


