OLIVER WITT

TOPOLOGICAL AND ONLINE ANALYSIS OF DYNAMIC STORAGE NETWORKS

From the Institute of Theoretical Computer Science
of the Universität zu Lübeck
Director: Prof. Dr. math. K. Rüdiger Reischuk

TOPOLOGICAL AND ONLINE ANALYSIS OF DYNAMIC STORAGE NETWORKS

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the Universität zu Lübeck

from the Department of Computer Science / Engineering

Submitted by

Oliver Witt, M. Sc.
from Winsen / Luhe

Lübeck, 2016

First referee:	Prof. Dr. math. K. Rüdiger Reischuk
Second referee:	Prof. Dr. rer. nat. Andreas Brandstädt
Date of oral examination:	December 6, 2016
Approved for printing:	December 8, 2016

»Everything flows.«

— Heraklit (c. 535 — c. 475 BC)

PREFACE

During my studies at the Universität zu Lübeck, I discovered my passion for the mathematical and theoretical aspects of computer science. Lectures of the Institute of Theoretical Computer Science were among my favorite ones. I also enjoyed teaching students as a tutor. Still, I did not expect to become a PhD student at this institute until I almost finished my master's thesis. But then, it seemed like the most natural thing to do.

In the summer of 2012, Prof. Reischuk and I started to model supply and demand scenarios based on flow networks with a strong focus on problems arising together with the possibility to store flow at specific nodes. Our original motivation is a quite practical one. At times of growing shares of renewable energy, the amount of electricity supplied to the power grid becomes more and more unpredictable. We modeled this aspect as an online problem for a certain kind of network that we defined in this context.

In order to make quite general statements about these networks and online problems, I soon immersed in the world of *mimicking networks*. A central insight from this realm carried over to our online problems is that in many cases it suffices to analyze quite small networks. Moreover, the complexity of the technique we developed for the analysis of our networks depends only on the number of nodes that cannot only forward flow, but also produce, store, or consume it.

These mimicking networks, in turn, are heavily intertwined with the set of all minimum s-t cuts of an s-t flow network. As this can be considered the basis of all results in this thesis and since many questions in this area are still open, I dedicated an entire chapter just to this topic.

This text is intended to be read front to back, but each chapter is as self-contained as possible. Concepts and notions that are frequently used are presented in an own chapter, while other concepts are introduced right where they are needed.

I thank Rüdiger for giving me the freedom to explore whatever I liked. He always had an open ear for my problems and inspired me with new ways to approach them. I would also like to thank in alphabetical ordering Christoph Stockhusen, Till Tantau, Claudia Witt, Jakob Witt, Maja Witt as well as my brother Ingo Witt and my parents Regina Witt and Wolfgang Witt.

Oliver Witt
Lübeck, fall of 2016

ABSTRACT

Storage sites are a key component of future power grids that have a high share of renewable energy [4, 18, 73]. Using dynamic flow networks and techniques from online problems [3, 9, 28], we model and analyze the question of how to utilize these storage sites most efficiently. Our model consists of a multi-terminal flow network where each terminal is a supply, storage, or demand node. For sequences of supplies and demands that are revealed step by step, we answer the two questions at which storage nodes to store flow if any is oversupplied, and from which storage nodes to take flow if too little is supplied. The limited capacities of a network coupled with the unknown future supplies and demands may produce scenarios in which flow cannot be stored in a way that an optimal use of it can be guaranteed.

Using external flow patterns known from the context of mimicking networks [40], we develop a technique to obtain optimal online algorithms for such problems. We apply it to show that in every network with a single producer and a single consumer node, there is always an online algorithm that can guarantee that approximately 83% of the supplied energy can be consumed in comparison to the offline setting where all future supplies and demands are known ahead. We also show that there are networks with a single producer and a single consumer node where no better performance than this is possible.

Two multi-terminal networks are mimicking networks if they share identical external flow patterns, i. e., for every flow that evokes certain net flows at the terminals of the one network, there is a flow evoking the same net flows at the terminals of the other network. For these mimicking networks, we show that the common assumption that for every terminal bipartition there is a unique min-cut separating the terminals according to this bipartition [52, 55] is a quite strong restriction and yields wrong expectations concerning the complexity of related problems. We show that the techniques commonly used to enforce this assumption may yield networks whose smallest contraction-based mimicking network is exponentially larger than of the original one. We examine problems in this setting *without* this unique min-cuts assumption and give close upper and lower bounds on the complexity of MCBMN, i. e., of deciding whether the number of nodes of the smallest contraction-based mimicking network of a given network is at most a given number. We show that MCBMN is in Σ_2^P and that MCBMN is coNP-hard. The analysis of this problem parameterized over the number of terminals is difficult without the unique min-cuts assumption. We pinpoint few cases that suggest that this problem is not fixed-parameter tractable anymore when the assumption is dropped. So far, it was only known that the problem is in FPT under the unique min-cuts assumption [52].

The lifted unique min-cuts assumption gives rise to up to exponentially many min-cuts for each terminal bipartition. By representing min-cuts either by all nodes on the source side or by all edges that are cut, different types of set systems are obtained. These set systems are poorly understood and, especially in the multi-terminal case, the root for many unsolved problems in this and related areas. For the classical 2-terminal case, we find a short characterization for the set systems that represent min-cuts as node sets. For the edge set representation, we develop a new framework that replaces

the set systems of well-known NP-complete problems by the edge set system implicitly encoded in an s-t flow network. The two opposing effects of succinct representation and restrictions on the sets that can be encoded are shown to outweigh each other depending on whether the emerging problem involves counting all min-cuts. If no counting is involved, the problems are P-complete. Otherwise, the problems are conjectured to be PP-complete since the counting versions are #P-complete [71].

For the analysis of these problems, the notion of the max-flow DAG is developed that succinctly represents the min-cut set systems. It allows simple algorithmic solutions and helps us to discover several dualities among the new problems.

ZUSAMMENFASSUNG

Energiespeicher sind eine Schlüsselkomponente für zukünftige Stromnetze, insbesondere für solche mit hohem Anteil erneuerbarer Energien [4, 18, 73]. Wir modellieren und analysieren die Frage wie man diese Energiespeicher möglichst effizient nutzt mit dynamischen Flussnetzwerken und Techniken aus der Welt der Online-Probleme [3, 9, 28]. Unser Modell besteht aus einem Multi-Terminal-Netzwerk, dem jedem Terminal eine Rolle als Angebots-, Speicher- oder Nachfrageknoten zugeordnet ist. Wird eine zunächst unbekannte Folge von Angebots- und Nachfragemengen Schritt für Schritt bekannt, so stellen sich zu jedem Zeitpunkt die Fragen bei welchen Speicherknoten man Fluss speichern sollte, falls mehr Fluss angeboten als nachgefragt wird, und von welchen Speicherknoten man Speicher entnehmen sollte, falls es umgekehrt ist. Die beschränkten Kapazitäten des Netzwerks zusammen mit den dezentralen Energiespeichern können nämlich wegen der unbekannten Zukunft zu Szenarien führen, wo eine optimale Nutzung des zur Verfügung gestellten Flusses nicht garantiert werden kann.

Zur Beantwortung dieser Fragen benutzen wir *external flow patterns*, die aus der Welt der *mimicking networks* bekannt sind [40]. Wir entwickeln eine Technik für die Analyse unseres Modells, welche zu optimalen Online-Algorithmen führt. Ferner zeigen wir, dass es für die Klasse von Netzwerken, die genau einen Angebots- und einen Nachfrageknoten besitzen, immer einen Algorithmus gibt, der sicherstellt, dass annähernd 83% der angebotenen Flussmenge zum Nachfrageknoten geschickt werden kann im Vergleich zu einem Algorithmus, der die gesamten Angebots- und Nachfragemengen vorab kennt. Des Weiteren zeigen wir, dass es solche Netzwerke gibt, bei denen es keinen besseren Online-Algorithmus gibt.

Zwei Multi-Terminal-Netzwerke sind *mimicking networks*, wenn sie die gleichen *external flow patterns* besitzen, d. h., wenn es für jeden Fluss, der in einem Netzwerk bestimmte Nettoflüsse an den Terminalen hervorruft, einen Fluss für das andere Netzwerk gibt, der dort an den Terminalen die gleichen Nettoflüsse hervorruft [40]. In der Literatur zu diesen mimicking networks gibt es die gängige Annahme, dass es zu jeder Terminal-Zweiteilung genau einen minimalen Schnitt gibt, der die Terminale entsprechend dieser Zweiteilung trennt [52, 55]. Wir nennen diese Annahme die *unique min-cuts assumption* und zeigen, dass diese Annahme eine starke Einschränkung an die Netzwerke ist und dass sie falsche Erwartungen an die Komplexität verwandter Probleme weckt. Darüber hinaus zeigen wir, dass die Methoden, die sicherstellen sollen, dass die unique min-cuts assumption zutrifft, dazu führen können, dass das kleinste *contraction-based mimicking network* exponentiell in der Anzahl der Terminale größer wird. Des Weiteren zeigen wir für die Komplexität des Problems zu entscheiden, ob ein gegebenes Netzwerk ein contraction-based mimicking network einer gegebenen Maximalgröße zulässt, eine untere Schranke von coNP und eine obere Schranke von Σ_2^P. Die Analyse dieses Problems parametrisiert über die Anzahl der Terminale ist ohne die unique min-cuts assumption komplex, wir können allerdings einige wenige Fälle identifizieren, die die Vermutung stützen, dass das Problem nicht in FPT ist. Bisher ist nur bekannt, dass das Problem mit der unique min-cuts assumption in FPT ist [52].

Ohne die unique min-cuts assumption kann es für jede Terminal-Zweiteilung bis zu exponentiell in der Anzahl der Terminale viele zugehörige minimiale Schnitte geben. Welchen Einschränkungen diese Mengen von minimalen Schnitten unterliegen und wie sich diese Mengen gegenseitig bedingen ist noch nicht vollständig verstanden. Wir zeigen für den Fall mit 2 Terminalen eine kurze Charakterisierung für die Menge, die sich ergibt, wenn man minimale Schnitte durch die Knoten auf der Quellenseite repräsentiert. Die Einschränkungen für die Mengen definiert über die geschnittenen Kanten der minimalen Schnitte lassen sich schwieriger charakterisieren, sodass wir eine neue Methode entwickeln. Dafür ersetzen wir die Mengensysteme bekannter NP-vollständiger Probleme durch das Kanten-Mengensystem implizit codiert in einem Flussnetzwerk, das nun Teil der Eingabe ist. Für die Komplexität des neuen Problems sind zwei gegensätzliche Effekte zu beobachten: Während die kompakte Codierung das Problem potenziell schwieriger macht, führen die Einschränkungen dieser Mengensysteme eher zu einer Vereinfachung. Wir zeigen, dass das neue Problem P-vollständig ist, falls es nicht das Zählen aller minimalen Schnitte beinhaltet. Im anderen Fall vermuten wir, dass das Problem PP-vollständig ist, da das entsprechende Zählproblem #P-vollständig ist [71].

Für die Analyse dieser Probleme entwickeln wir das Konzept des *max-flow DAGs*, der nicht nur die Menge aller minimalen Schnitte kompakt repräsentiert, sondern auch einfache algortihmische Lösungen für die neuen Probleme zulässt.

CONTENTS

LIST OF NOTATIONS

This list contains abbreviations and notations that are often used in this thesis. The entries are sorted alphabetically with Greek symbols inserted according to their English transliteration. Special symbols are put at the front.

◉	A terminal.
•	A non-terminal.
$\mathcal{A}$	An online algorithm (Def. 14, p. 106).
α	Storage vector (Def. 22, p. 113).
$\mathcal{B}_N$	Set of all terminal bipartitions for N (Def. 1, p. 20).
BANF	BINARY ACYCLIC NETWORK FLOW (Problem 7, p. 51).
c	Capacity function or competitiveness (pp. 17ff. and Def. 15, p. 106).
$c_{N,S}$	Minimum capacity over all S-cuts in N.
CBMN	Contraction-based mimicking network (Def. 7, p. 71).
$\mathrm{comp}(\cdot)$	Function assigning competitiveness to argument (p. 106).
$\mathcal{E}(N)$	Edge set system of N (Def. 2, p. 30).
f	A flow (p. 18 and p. 105).
$\lvert f\rvert$	Value of f (Eq. 5, p. 18).
$\vec{f}$	External flow of f (p. 18).
F_M	External flow pattern of M (p. 105).
F_N	External flow pattern of N (p. 18).
M	A dynamic storage network (Def. 9, p. 103).
$\mathcal{M} = (M, \rho)$	A dynamic storage system (Def. 13, p. 106).
MCBMN	MINIMUM CONTRACTION-BASED MIMICKING NETWORK (Problem 28, p. 88).
MCHS	Minimal min-cut hitting set (p. 61).
μ	Pp. 33ff.
N	A multi-terminal network (p. 17).
N_d	Max-flow DAG of N (Def. 4, p. 36).
N_e	Edge cut network of N (Def. 3, p. 34).
$N_f = (V_f, E_f)$	Residual network of N with respect to f (p. 20).
NCA	NETWORK CAPACITY AUGMENTATION (Problem 15, p. 60).
$\mathrm{opt}(\cdot)$	Profit of an optimal offline algorithm (Eq. (127), p. 106).
p-MCBMN	MCBMN parameterized over the number of terminals (p. 91).
ρ	Set of request sequences.
σ	A request.
$\bar{\sigma}$	A request sequence (Def. 10, p. 104).
$\sigma_{M,\text{fill}}$	Filling request of M (Def. 20, p. 112).
$\sigma(M)$	Request space of M.

$\tau(\sigma)$	Duration of σ (Def. 10, p. 104).
$\mathcal{T}$	Set of all terminals in a network.
$\mathcal{V}(N)$	Node set system of N (Def. 2, p. 30).

1 INTRODUCTION

Electric power grids are the most fundamental infrastructure of modern societies. Their structures evolved without clear awareness of the implications for future demands. These networks are among the most complex networks man-made and not surprisingly, they are poorly understood in many ways. There have been power outages whose cause remains unknown, for example the 2003 outage in Northeastern U. S. [4]. It turns out that this structure is obsolete in many aspects and is due to a change, in particular because of the much discussed and needed transition from fossil to renewable energy sources.

For this, many aspects of the current grids, their changes, and the envisioned power grid of the future are actively being researched. These aspects include security, reliability, efficiency, cost-effectiveness, and many more. They range from technical topics over economical contemplations to artificial intelligence.

The objective of very high reliability and quality of service is difficult to achieve in such a system. The quite young realms of non-linear dynamical networks and in particular complex adaptive systems try to capture and analyze the arising complexity and its problems. Understanding when such a system leaves a stable state may be key to avoiding future blackouts [61]. But not only blackouts, but also *brownouts*, often unintentional drops in voltage, need to be understood and prevented.

The grid of the future is supposed to be smart, making all components aware of each other's state. This forms the basis for the implementation of many algorithmic strategies that are devised.

A key part of the future grids is the capability to *store electricity*. Its necessity is unquestioned and verified by many simulations and projections [4, 18, 73]. While the amount of consumed energy always underlies unanticipated fluctuations, the amount of produced energy used to be projectable and thus known. This changes in the presence of wind, hydro, and solar power. The possibility to store electric energy at appropriate sites if put to use wisely, attenuates the associated effects of this uncertainty. Therefore, it gains significance together with the share of renewable energy and evolves to an essential ingredient for future power grids.

The load-balancing and flexibility offered by storage sites can partly be achieved at home by smart appliances that are aware of the current demand and supply, and by batteries of electric cars. The latter are expected to generate a huge storage capacity and quick response times, but are costly and only applicable for short-term storage. In the presence of seasonal effects and economical considerations, large storage facilities, like *pumped storage hydro power stations*, are indispensable [43].

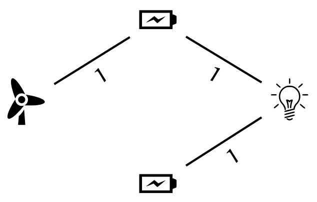
While some parts of the envisioned power grid are only beginning to evolve, storage facilities are already in use and technologically mature. They have proven to be economical and their importance will grow with the share of renewable energy [18].

Of the broad range of topics, we study a very specific part: the problem of how to use the storage capacity of a network. That is, where should flow be stored if it can be stored at several storage sites? Our contribution is a better understanding of just this question.

In this thesis, we introduce *dynamic storage networks* modeling the problem of deciding how to store flow in a flow network at times where the supply exceeds the demand, and which storage facility to tap at times of flow shortage, while future supplies and demands are unknown. We put to use the techniques of online problems – a field dealing with problems whose input is presented to a solver only step by step such that it has to cope with incomplete information – in the realm of dynamic flow networks. Our model fades out many technical issues, as for example lossy transmission, and concentrates on the core difficulty of storage strategies. Then, we derive a general technique for the online analysis of this problem. Moreover, worst case networks for certain classes are presented.

Despite this quite specific motivation, our dynamic storage networks can be used to model any kind of supply-demand scenario with storage capabilities and unknown future requests on a network topology. As our model uses zero-transit times along the edges, this particularly applies where the transit times can be neglected. Our model, for example, might apply to caching problems occurring in networks.

We would like to illustrate the problems arising together with the possibility to store flow with an example. These problems seem to be quite luxurious ones at first sight: At times of overproduction, which storage site should be used to store the excess, and at times of underproduction, from which storage site should energy be taken? The decentralized storage sites together with the limited capacities of the power lines can cause situations in which the stored flow might be condemned to stay unused at its storage site. So why has the power been stored at that storage site at all? In particular in the presence of unknown future supplies and demands, the electricity needs to be stored in a way that best anticipates all possible future scenarios. This may lead to a situation as the described one.

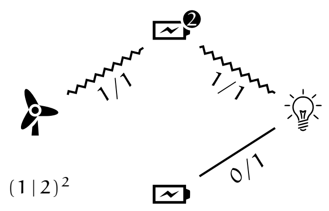


To demonstrate the model and the problems that we analyze, consider the network depicted on the right. It contains a single energy producer, represented by a wind turbine, and a single energy consumer, represented by a light bulb. Moreover, there are two batteries, that symbolize storage sites. The lines with the numbers stand for power lines with associated capacities. We consider discrete time steps, and in each time step, a line can be passed by as many energy units as its number indicates. The term $(x \mid y)^z$ abbreviates the situation in which for a total of z time units, x energy units are produced, while y energy units are demanded.

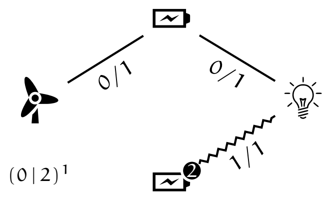
Now suppose that the wind turbine is generating 1 energy unit, and the light bulb does not consume any energy. Hence, this energy unit can be stored at the storage sites. We assume that there is no loss of energy when it is transferred or stored. The just described situation now continues for a total of two time steps, i. e., $(1 \mid 0)^2$.

The main question associated with this model now is: How would you store these two energy units at the batteries not knowing the supplies and demands of the future time steps?

Let us assume you choose to store both units at the upper battery. In the picture on the right, this is indicated by a circled number attached to the according battery. Zigzag lines indicate active power lines. If the wind now continues to blow such that the turbine produces one unit, and the consumer now demands two units, then a total of one unit needs to be released from batteries in order to satisfy the demand. However, all power stored at the upper battery is blocked by the energy that is transferred from the turbine to the light bulb. This scenario repeats for two time steps and is depicted on the right. While a total of four units is produced by the turbine, only two are able to reach the light bulb. It would have been much better if all power had been stored at the lower battery instead. In that case, all four units could have been sent to the consumer. So what if we do store all energy at the lower battery?

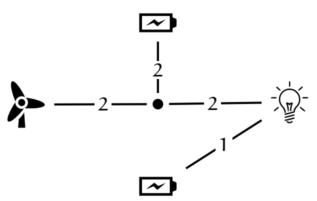


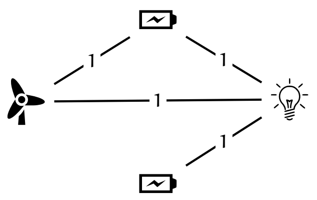
This might not be the best idea either. If the turbine now stops producing energy, and the consumer still demands two units, then we again run into a problem. In case of $(0\,|\,2)$, we can only send one unit to the consumer. Of a total of two produced energy units, only one can be sent to the consumer. Again, the consumer was only able to consume a mere 50% of the energy that has been produced. Quite unacceptable from an economical and ecological point of view. Even though it is better to split the energy and store some at the upper battery and some at the lower battery, it is still not possible to guarantee that 100% of the produced energy will end up being consumed. The best we can do is to find the storing strategy that guarantees the highest percentage.



The fraction corresponding to this percentage is a quite characteristic number for a network and its reciprocal is called *competitive ratio*. Among all networks with one producer and one consumer, we identify those networks with highest competitive ratio, which is shown to be $(1+\sqrt{2})/2$. Such a network is a *hard network* of that class. Moreover, we develop a technique to derive an optimal storing strategy for arbitrary networks. It is based on a vector space in which each point represents the net flows of storage nodes in response to a given request. The best ratio of storing flow at the storage nodes is then found by identifying a center point with respect to a scaled L_1 distance measure. In its algorithmic form, this technique leads to a runtime that is dependent solely on the number of terminals of the network.

This is one of the central insights carried over from the realm of *mimicking networks*. These are multi-terminal networks that allow exactly the same set of external flows as another multi-terminal network, where an external flow is characterized by the net flows that a flow evokes at the terminals. These networks are of interest wherever not the flows itself, but rather the net flows evoked at certain nodes are of importance. Such mimicking networks may be much smaller than the original network by using fewer non-terminals. Below you can see two networks similar to the ones before. The upper one has a new kind of node, a non-terminal, denoted by a •, in contrast to the other nodes which are all terminals.





In fact, these two networks are mimicking networks of each other. This is due to the fact that every flow that is feasible in one network evokes net flows at the terminals such that there is another flow in the other network evoking the same net flows at the corresponding terminals. Try it out.

The literature concerning mimicking networks deals with various topics, but has focused on bounding the size of the smallest mimicking network for a given network [10, 12, 40, 52, 55].

A prominent problem from this research area is to find a smallest mimicking network for a given one by contracting nodes. An assumption commonly made in the literature for this and other problems is that for every bipartition of the terminal set, there is exactly one minimum cut separating the terminals according to this bipartition [52, 55]. This will be referred to as *unique min-cuts assumption* in this thesis. We show that the results obtained with respect to this assumption cannot be carried over to the general case. Our proof demonstrates that the techniques used to enforce the unique min-cuts assumption by slightly altering the original network possibly cause the smallest contraction-based mimicking network to be exponentially larger in the number of terminals than the smallest contraction-based mimicking network of the original network.

Furthermore, we model many questions in this context as decision problems and analyze their complexity. This yields lower bounds for the complexity of related problems where so far only upper bounds have been known [52].

Dropping the unique min-cuts assumption unleashes the true complexity of finding a smallest contraction-based mimicking network that is directly linked to the set of all minimum cuts in a flow network. To close in on still unsettled problems, these set systems are carefully studied. A minimum cut can be represented by the nodes on the source side or by all edges that are cut. Each representation leads to a set system when considering all minimum cuts of a network. We study these for both classical flow networks and multi-terminal networks.

For the set systems representing minimum cuts with nodes, we present a short characterization. Moreover, we show how to efficiently construct a network whose set of all min-cuts corresponds exactly to the given one if one exists. While the node set systems can be quite well understood, the set systems emerging from the edge representation cause problems. As an indirect approach to study the characteristics of such set systems, a new framework is developed. For this, the input of well-known NP-complete decision problems that contain set systems is altered such that the set system is replaced by a flow network. The problems' questions remain the same but now assume the set system to be the set of all minimum cuts of the given network represented as edge sets. The new problems are called min-cut variants. HITTING SET is an example of such a problem. Its min-cut variant asks whether k edges can be found in a network such that every minimum cut cuts at least one of these k edges. This problem is closely related to the very natural problem of augmenting the capacity of a network by increasing the capacity of existing edges by a given amount.

The complexity of these min-cut variants is influenced by two opposing effects in comparison to the original problem. Since networks may have up to exponentially many minimum cuts, set systems can be encoded quite succinctly in networks this way. This may increase the complexity. At the same time, not every set system can be encoded in a network. This may decrease the complexity. We show that the true complexity depends on a single aspect: Does the problem involve counting all min-cuts? If this is not the case, then the problem is P-complete and thus has gotten easier than the original one. If it does involve counting, then its complexity is equivalent to deciding whether a flow network has at least k minimum s-t cuts, where k is part of the input. This is expected to be PP-completeness, since the counting version of the problem is #P-complete [71]. For example, the min-cut variant of HITTING SET does not involve counting all min-cuts and thus is P-complete while the min-cut variant of SET BASIS does involve the counting of all min-cuts and thus is conjectured to be PP-complete.

1.1 ORGANIZATION

Chapter 2 introduces concepts and notions that are repeatedly used throughout this thesis and familiarize the reader with our notation. Many very basic results concerning flow networks are presented. Many other concepts that are used just once are introduced right before they first appear.

The main part of this thesis is composed of three chapters, Chapter 3, Chapter 4, and Chapter 5. Even though these chapters are quite self-contained, each one of them relies on some results of the earlier chapters. Each chapter introduces the considered problems, presents the results, and discusses them.

Chapter 3 studies the set of all minimum s-t cuts of a flow network. As cuts can be represented by nodes or by edges, we study both of these aspects trying to characterize these sets.

Chapter 4 deals with mimicking networks and shows that an assumption commonly made in the literature in this problem setting imposes a strong restriction and yields results that do not apply to the general case. Some questions that have been believed to be settled hit new problems without this erring assumption. We link problems that we are unable to resolve to problems of Chapter 3 that also have been left unresolved.

Chapter 5 moves the focus away from the set of all min-cuts to an online problem in the realm of dynamic flow networks dealing with storage strategies. In particular, we develop techniques to find an optimal strategy for any given such network.

The thesis concludes with reviewing the presented results in Chapter 6. Additionally, future research directions related to the presented results are discussed.

2 CONCEPTS AND NOTATION

This chapter introduces the reader to the concepts and techniques commonly used in the realm of network flows. We also make the reader familiar with our notation used throughout this thesis. We expect the reader to be familiar with graph theory. Renowned textbooks dedicated to this topic are *Graph Theory* by Diestel [19], or *Modern Graph Theory* by Bollobás [8].

A good part of this work consists of analyzing the complexity of problems. We assume the reader to be acquainted with the concepts of complexity theory. Classic textbooks covering many of the notions used in this thesis include *Computational Complexity* by Papadimitriou [67], *Computational Complexity: A Modern Approach* by Arora and Barak [7], and *Introduction to Automata Theory, Languages, and Computation* by Hopcroft, Motwani, and Ullman [44]. Typical results of this thesis include the P- or NP-completeness of problems. Other parts of this thesis will deal with *parameterized complexity* which is a discipline trying to understand from which aspects of the input (the parameter) the hardness of problems originates. For an introduction to this field of computational complexity, we refer the reader to the textbooks *Fundamentals of Parameterized Complexity* by Downey and Fellows [24] and *Parameterized Complexity Theory* by Flum and Grohe [31]. We will mention the complexity class FPT, which contains all parameterized problems solvable within a time that may depend arbitrarily on the parameter, but only polynomially on the input size. For *fixed parameters*, the problem becomes *tractable* – hence the name FPT (fixed parameter tractable).

We encode all problems in binary and assume some reasonable way of representing the considered objects. A network's underlying graph, for example, can be represented by an adjacency matrix or adjacency list. The capacities of our networks will be defined over real numbers. Whenever we encode networks, we assume that all real numbers are encoded with fixed precision, as computers would do.

The theory of flow networks roots in the 1950s [75] and has ever since been a field of active research. Flow networks are used to model a great variety of real world applications. Flows can pertain to people, goods, electricity, among many other things. Classical flow networks with a single source and a single sink are a well-known and well-understood concept. Multi-terminal networks generalize these flow networks by allowing more terminal nodes than two (a source and a sink) and not assigning a fixed role to a terminal as source or sink. Formally, a *multi-terminal network* $N = (V, E, \mathcal{T}, c)$ (often only referred to as *network*) consists of a directed graph (V, E), a set of terminals $\mathcal{T} \subseteq V$, and a capacity function $c : V \times V \to \mathbb{R}_0^+ \cup \{\infty\}$. For all $(u, v) \in V \times V$ with $(u, v) \notin E$, we have $c(u, v) = 0$. The nodes $V - \mathcal{T}$ are the *non-terminals*. In case of $|\mathcal{T}| = 2$, we say $\mathcal{T} = \{s, t\}$ (or sometimes $\mathcal{T} = \{t_1, t_2\}$) and obtain classical s-t flow networks. We assume that an arbitrary but fixed total order is imposed over the terminals such that $\mathcal{T} = \{t_1, t_2, \dots, t_k\}$ for $|\mathcal{T}| > 2$. The size of a network $|N|$ is the number of its nodes $|V|$. In the illustrations throughout this thesis, ◉ denotes a terminal and • a non-terminal, see for example Figure 1 on page 20.

A *flow* in N is a function $f : V \times V \to \mathbb{R}$ that satisfies

$$\begin{aligned} &\text{the } \textit{capacity constraint} && f(u,v) \leqslant c(u,v) \text{ for all } u,v \in V, && (1)\\ &\text{the } \textit{skew symmetry} && f(u,v) = -f(v,u) \text{ for all } u,v \in V, \text{ and} && (2)\\ &\text{the } \textit{flow conservation} && \sum_{u \in V} f(u,v) = 0 \text{ for all } v \in V - \mathcal{T}. && (3)\end{aligned}$$

An edge $e \in E$ is *saturated* by a flow f if $f(e) = c(e)$, i. e., if the capacity constraint (1) is met with equality. The skew symmetry (2) is not an original constraint by Ford and Fulkerson, but has been used by Cormen et al. [16]. It simplifies the definitions of a node's *net flow* and the *residual network*, which will come up soon. The net flow

$$f(v) = \sum_{u \in V} f(u,v) \tag{4}$$

of a node v with respect to a flow f is the sum of all flow units reaching v minus the sum of all flow units leaving v. It is 0 for all nodes $v \in V - \mathcal{T}$. The terminals are exempt from the flow conservation (3). The net flow of terminals is positive (negative) if, and only if, more flow reaches (leaves) than leaves (reaches) it. The flow that appears or disappears at these nodes can be seen as connection to the outside. This is why the vector $\vec{f} = \big(f(t_1), f(t_2), \dots, f(t_{|\mathcal{T}|})\big)$ of net flows is called the *external flow* of f as introduced by Hagerup et al. [41]. Further, we call $F_N = \{\, \vec{f} \mid f \text{ is a flow in } N \,\}$ the *external flow pattern* of N.

For classical flow networks, Ford and Fulkerson make the assumption that every edge incident to s is directed away from s and every edge incident to t is directed towards t. We do not make this assumption which allows a flow from the sink to the source. To make clear in what direction flow is supposed to flow, we define a flow to be a T'*-flow* for a non-trivial $T' \subset \mathcal{T}$ if it satisfies $f(v) \leqslant 0$ for all $v \in T'$ and $f(v) \geqslant 0$ for all $v \in \mathcal{T} - T'$. This definition gives rise to another ambiguity: A $T' \cup \{v\}$-flow with $f(v) = 0$ for some $v \in \mathcal{T}$ is also a T'-flow. In particular, every cycle flow (where all nodes have a net flow of 0) and the 0-flow f_0 with $f_0(e) = 0$ for all $e \in E$ are T'-flows for any non-trivial $T' \subset \mathcal{T}$.

The *flow value*

$$|f| = \sum_{t \in T'} f(t) \tag{5}$$

of a T'-flow is the sum of the net flows of all nodes in T'. Notice that due to the definition of a T'-flow, we are summing non-negative values only. A T'-max-flow is a T'-flow of maximum value. There may be several distinct T'-max-flows in a network. Often, we will only talk about a flow or a max-flow if the direction of flow is clear from the context.

A *path* p in N is a set $\{\,(x_1,x_2),(x_2,x_3),\dots,(x_{n-1},x_n)\,\} \subseteq E$ of directed edges where all nodes $x_i \in V$ are distinct. A u-v path is a path with $x_1 = u$ and $x_n = v$ for $u,v \in V$. The length of a path denoted by $|p|$ is the number of its edges. The *width of a path* is the minimum capacity among all its edges. A *path flow* f_p is a flow in N with $f_p(e) = 0$ if $e \notin p$ and p is a path. A path flow always connects terminals. A *cycle* o in N is a set of directed edges $\{\,(x_1,x_2),(x_2,x_3),\dots,(x_{n-1},x_n),(x_n,x_1)\,\} \subseteq E$ where all nodes $x_i \in V$ are distinct. The *width of a cycle* is defined analogously to the width of a path as the minimum capacity among all its edges. For a cycle o, f_o is a *cycle flow* if f_o is a flow in N with $f_o(e) = 0$ if $e \notin o$. The value of every cycle flow f_o is 0 since $f_o(v) = 0$ for all $v \in V$.

We use the capacity function in different contexts. Even though these are formally different functions, all of them are denoted by c. It becomes clear from the context which function applies. We already introduced the edge capacity function. We can also assign a capacity to two arbitrary but disjoint node sets $A, B \subseteq V$ as

$$c(A,B) = \sum_{(u,v)\in(A\times B)\cap E} c(u,v)\,. \tag{6}$$

Extending this notion, the *capacity of a non-empty node set* $A \subset V$ is defined as

$$c(A) = c(A, V-A)\,. \tag{7}$$

For a non-trivial $V' \subset V$, a partition (S,T) of V into two non-empty sets is a V'*-cut* if $S \cap \mathcal{T} = V' \cap \mathcal{T}$. For a singleton set like $\{v\}$, we say v-cut instead of $\{v\}$-cut. If V' is clear from the context we will sometimes simply say cut. In particular, a cut in an s-t flow network usually refers to an s-cut. Note that we define cuts in a way that might contain all terminals on one side and hence do not separate any terminals. This will be useful in a few situations, but in most cases cuts do separate terminals. S will be referred to as *source side* and T as *sink side* (even in case S or T do not contain any terminals). Sometimes, a cut will be denoted just by its source side. The capacity of a cut (S,T) is $c(S,T)$ or just $c(S)$. A cut (S,T) is a V'*-min-cut* if (S,T) is a V'-cut of minimum capacity. The V'-min-cuts with smallest and largest number of nodes on the source side are *extremal* V'*-min-cuts*. These differ if, and only if, there are multiple min-cuts. The *edge cut set* of a cut (S,T) is

$$E(S,T) = (S \times T) \cap E\,, \tag{8}$$

the set of all edges directed from S to T. If for an edge (u,v) we have $|S \cap \{u,v\}| = 1$, then (S,T) *separates* u *and* v. If, and only if, additionally $(u,v) \in E(S,T)$, then (S,T) *cuts* (u,v). Any edge directed from T to S is not cut by (S,T). An edge cut by some min-cut will also be referred to as *min-cut edge*. Since $T = V - S$, we may abbreviate the function with $E(S)$. The flow across (S,T) with respect to a flow f is

$$f(S,T) = \sum_{e\in S\times T} f(e)\,. \tag{9}$$

Recall the skew symmetry (2) and note that $f(S,T)$ is the sum of all flow units going from S to T minus the sum of all flow units going in the opposite direction.

Two min-cuts are *crossing min-cuts* if their source sides S_1, S_2 have the property that $S_1 - S_2 \neq \emptyset$ and $S_2 - S_1 \neq \emptyset$.

We will often make the assumption that a network is undirected. In this case, the edge set shall not contain unordered pairs of nodes, but rather is a symmetric relation containing ordered pairs of nodes. Additionally, the capacity function is also symmetric, i. e., $c(u,v) = c(v,u)$ for all $(u,v) \in V \times V$.

Moreover, we will often consider the various bipartitions of the terminals into two non-empty sets. Such a bipartition can be understood as an assignment of roles to the terminals – the ones belonging to the one set are sources and the ones belonging to the other set are sinks. If the network is undirected, it does not matter which set of nodes is assigned which role. From this perspective, (S,T) and (T,S) describe the same terminal bipartition. It hence suffices to consider only those terminal bipartitions (S,T) with

$t_1 \in S$. Moreover, a terminal bipartition (S, T) is fully identified by S. These considerations motivate the following definition.

▷ DEFINITION 1 (SET OF TERMINAL BIPARTITIONS): For an undirected network N, let

$$\mathcal{B}_N = \{ T' \subset \mathcal{T} \mid t_1 \in T' \} \tag{10}$$

be the *set of terminal bipartitions* of N.

If there are no ambiguities, we neglect the subscript. For every $B \in \mathcal{B}$, we use $\overline{B}$ to denote $\mathcal{T} - B$. Moreover, for each $B \in \mathcal{B}$, we consider B to be the sources and $\overline{B}$ the sinks. $\mathcal{B}$ contains $2^{|\mathcal{T}|-1} - 1$ terminal subsets for any network N. For any $B \in \mathcal{B}_N$, let $c_{N,B}$ be the capacity of a B-min-cut in N. Note, that $c_{N,B} = c_{N,\overline{B}}$ for all $B \in \mathcal{B}$ and all undirected N.

RESIDUAL NETWORK. The *residual network* $N_f = (V, E_f, \mathcal{T}, c_f)$ of N with respect to a flow f in N is a new network with

$$E_f = \{ e \in V \times V \mid c(e) - f(e) > 0 \} \text{ and} \tag{11}$$

$$c_f(e) = c(e) - f(e) \text{ for all } e \in E_f\,. \tag{12}$$

Recall that $c(e) = 0$ if $e \notin E$. It is an important technique used in many contexts. We, too, will use them for a variety of problems, including the computation of max-flows, the construction of max-flow DAGs, a concept introduced in Chapter 3, and the minimization of mimicking networks. Figure 1 shows an example.

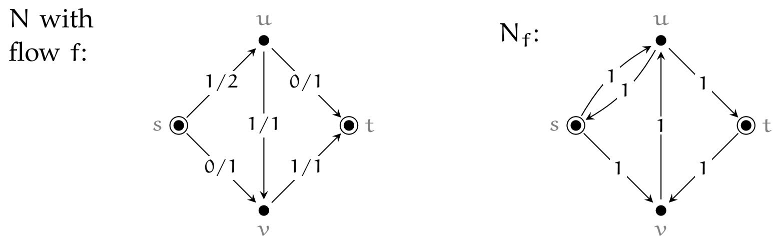


Figure 1: Network N with flow f defines the residual network $N_f = (V, E_f, \mathcal{T}, c_f)$. The edges e of N are labeled with $f(e)/c(e)$, while the edges of N_f are only labeled with the capacity c_f. According to (11), E_f contains $e = (v, u)$ since $c(e) = 0$ and $f(e) = -1$, where the latter is due to the skew symmetry (2).

MAXIMUM FLOWS. The following renowned Max-Flow Min-Cut Theorem has been found independently by Ford and Fulkerson [33] and by Elias, Feinstein, and Shannon [26] in 1956.

▷ THEOREM 1 (MAX-FLOW MIN-CUT THEOREM): In any s-t network the max-flow value equals the min-cut capacity.

It is a well-known fact that for every max-flow f and for every min-cut of a 2-terminal network, we have $f(S, T) = |f|$. Due to the definitions of the cut capacity (7) and the flow across a cut (9), a max-flow saturates all edges in S-T direction and does not allow any flow in T-S direction.

The problems of finding a max-flow and a min-cut in a network can be stated as linear programs. It can be shown that these linear programs are dual implying the Max-Flow Min-Cut Theorem as a special case of the strong duality theorem of linear programming. Thus, the value of any flow is a lower bound for the min-cut value, and the cut capacity of any cut is an upper bound for the max-flow value. There is a great variety of algorithms to directly find a max-flow. One of the simplest of them is the Ford-Fulkerson algorithm. It is based on the insight that we can augment the value of a flow f if, and only if, the residual network with respect to f contains an s-t path (an *augmenting path*). In Figure 1, for example, there is an augmenting path s-v-u-t of width 1 in N_f. In its original version, an augmenting path is chosen repeatedly to augment the flow until no more such path is found. This algorithm has the major drawback that its runtime may depend on the value of a max-flow – turning it into a pseudo-polynomial algorithm. Moreover, if the capacities are irrational, the algorithm might not even terminate. By simply choosing the shortest augmenting path in each iteration, we obtain the Edmonds-Karp algorithm presented by Edmonds and Karp [25] that overcomes these shortcomings. The Ford-Fulkerson algorithm can be used to proof the fact that a network has an integer max-flow if all edge capacities are integer, a result known as *Integrality Theorem*, already stated by Ford and Fulkerson [33].

A further enhancement is to augment a flow along many shortest paths as proposed by Dinic [20]. A whole different class of algorithms make use of so-called *preflows* that flood the network such that nodes may have excesses of flow. Such nodes are iteratively relieved from the excess by sending it towards the sink or backwards to the source, see for example [60].

Karger's algorithm [47] is an example of a randomized algorithm to compute a min-cut of a connected graph. The idea is to randomly contract edges until only two nodes are left. Repeating this process sufficiently, a min-cut is found with high probability. More precisely, after $O\left(\binom{n}{2} \log n\right)$ repetitions, the probability *not* to have found the min-cut is less than $1/n$. An extension of this idea by Karger and Stein [48] achieves an order of magnitude improvement.

For a more detailed list of approaches and the history of max-flow algorithms, see for example the textbook *Network Flows* by Ahuja, Magnanti, and Orlin [2], the book *Networks and Algorithms* by Dolan and Aldous [22], or the survey of Ahuja, Magnanti, and Orlin [1]. A more recent textbook is *Graphs, Networks and Algorithms* by Jungnickel [46]. The problem of computing a max-flow is still subject of active research with a notable and quite recent example being that of Kelner et al. [51]. For our purposes though, it suffices to know that a max-flow can be computed in strongly polynomial time.

FLOW DECOMPOSITION. Flows can be decomposed into a set of flows. For this, the *sum of two flows* $f_1 + f_2 = f_3$ in N is defined quite naturally as $f_3(e) = f_1(e) + f_2(e)$ for all $e \in E$. This function satisfies the skew symmetry and the flow conservation. Hence, f_3 is a flow if, and only if, the capacity constraint is fulfilled. Ford and Fulkerson have observed that every flow f in a network N can be decomposed into a set of path and cycle flows such that their sum is f. Here is a version of the theorem from the book *Network Flows* by Ahuja, Magnanti, and Orlin [2]. While this result originally applies to s-t networks, we can also apply it to multi-terminal networks.

▷ THEOREM 2 (FLOW DECOMPOSITION THEOREM, [2]): Every flow f can be represented as the sum of at most $|V| + |E|$ cycle and path flows with non-zero value.

Proof. Ahuja, Magnanti, and Orlin [2] give an algorithmic proof. If $f(v_0) < 0$, then v_0 is a source and there is an edge (v_0, v_1) carrying a non-zero flow. If $f(v_1) > 0$, i. e., v_1 is the sink, we stop. Otherwise, the flow conservation

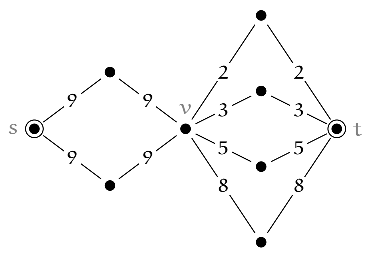


Figure 2: The problem of deciding whether a given flow f in a given network N can be decomposed into at most k path and cycle flows is NP-hard by a reduction from the well-known PARTITION problem [53], formally introduced on page 88. The reduction constructs a network with two disjoint s-v paths each of width $1/2 \sum_{a \in A} s(a)$. Additionally, there are $|A|$ disjoint paths from v to t, where the i^{th} path has width $s(a_i)$. The depicted network N is the result of the transformation of the PARTITION instance (A, s) with $A = \{a_1, a_2, a_3, a_4\}$ as well as $s(a_1) = 2$, $s(a_2) = 3$, $s(a_3) = 5$, and $s(a_4) = 8$. A max-flow in N can be decomposed into 4 path flows if, and only if, $(A, s) \in$ PARTITION. Since $(A, s) \notin$ PARTITION, a max-flow in N cannot be decomposed into 4 path flows.

constraint demands that some other edge (v_2, v_3) carries non-zero flow. We repeat this until we hit a node v_k with $f(v_k) > 0$, or we hit a node already visited. In the former case, we obtain a path flow f_p, in the latter a cycle flow f_o. We set the value of the path flow $|f_p| = \min\{-f(v_0), f(v_k), \min_{e \in p} c(e)\}$ and redefine $f(v_0) = f(v_0) + |f_p|$, $f(v_k) = f(v_k) - |f_p|$, and $c(e) = c(e) - |f_p|$ for every $e \in p$. If we obtained a cycle flow, we let $|f_o| = \min_{e \in o} c(e)$ and redefine $c(e) = c(e) - |f_o|$ for all $e \in o$.

This process is repeated, until all net flows are zero. Any remaining flow now corresponds to cycle flows. So we choose any node with outgoing flow and carry out the above procedure repeatedly until all capacities are zero. Clearly, f is the sum of the set of obtained path and cycle flows. Moreover, as every path flow reduces the net flow of some node or the capacity of some edge to zero, and every cycle flow sets the capacity of some edge to zero, we are done after at most $|V| + |E|$ path and cycle flows. □

The construction in the proof of the last theorem is remindful of the Ford-Fulkerson algorithm. A flow is constructed step by step, and each time a new network is created with adapted edge capacities. There is, however, one big difference. The Ford-Fulkerson algorithm adds back capacities for each flow, the just described procedure does not do this. While the flow along an edge may be lowered or even reversed in direction by the Ford-Fulkerson algorithm, this is not possible here. In effect, no obtained cycle or path flow can be undone.

One may ask how difficult it is to find the minimum number of path and cycle flows in which a given flow can be decomposed. It turns out that the decision version of this question is NP-complete by a reduction from PARTITION given by Kleinberg [53] – see an example in Figure 2.

FLOW NETWORKS WITH EDGE DEMANDS. The edges of an s-t network cannot only be assigned an upper bound for the flow in terms of a capacity, but also a lower bound, by so-called *edge demands*. Such a network $N = (V, E, \mathcal{T}, l, c)$ contains an additional function $l : E \to \mathbb{R}$. A flow $f : E \to \mathbb{R}$ in such a network is a function that does not only satisfy the capacity constraints, the skew symmetry, and the flow conservation, but also fulfills $f(e) \geqslant l(e)$ for all $e \in E$.

For the networks discussed so far, there was no question whether a flow existed at all. The 0-flow f_0 with $f_0(e) = 0$ for all $e \in E$ satisfies all constraints in all networks. This question, however, becomes non-trivial when we consider flow networks with edge demands. In particular, finding a *min-flow*, i. e., a flow of minimum value, is a new and interesting problem. We describe a solution to this problem similar to that by Ciurea and Ciupală [15]. They first identify a flow, if there is any, and then decrease the value of the flow until it hits the minimum.

For networks with edge demands, the capacity of an s-t cut (S, T) is defined as the sum of the lower bounds of all S-T edges minus the sum of the capacities of all T-S edges, that is

$$c(S,T) = \sum_{e \in E(S,T)} l(e) - \sum_{e \in E(T,S)} c(e)\,. \tag{13}$$

Interestingly, this leads to the following theorem, that can be proven similarly as the original Max-Flow Min-Cut Theorem.

▷ THEOREM 3 (MIN-FLOW MAX-CUT THEOREM): If there exists a flow in N, then the min-flow value in N is equal to the maximum s-t cut capacity.

Finding a flow in N can be reduced to finding a max-flow in a transformed network N' without edge demands. For this, let $N' = (V', E', \{s', t'\}, c')$ be a new network with

$$\begin{align} V' &= V \cup \{s', t'\}, \tag{14}\\ E' &\subseteq E \cup \{(t, s)\}, \tag{15}\\ c'(u, v) &= c(u, v) - l(u, v) \text{ for all } (u, v) \in E\,, \text{ and} \tag{16}\\ c'(t, s) &= \infty\,. \tag{17}\end{align}$$

E' contains additional edges. For each node $v \in V$, let the *net demand*

$$d(v) = \sum_{u \in V} l(u, v) - \sum_{w \in V} l(v, w) \tag{18}$$

be the demand of the incoming edges minus the demand of the outgoing edges. Now, we add an edge of capacity $|d(v)|$ for each node $v \in V$ if $d(v) \neq 0$ such that

$$\begin{align} e(v, t') \in E' \text{ if } d(v) < 0 \text{ and} \tag{19}\\ e(s', v) \in E' \text{ if } d(v) > 0\,. \tag{20}\end{align}$$

In the obtained network N', we compute a max-flow f'. If this flow saturates all edges leaving s', then there is a flow $f = f' + l$ in N. Due to the construction, f certainly meets the lower bound constraints. Moreover, f also satisfies the capacity constraint in N, as $f'(e) \leqslant c(e) - l(e)$. Finally, the flow conservation is given as well. Intuitively speaking, the net flow at every node in N', when removing s' and t' from the network, is of opposite sign as the net demand in N. Hence, when adding f' and l, the net flow is 0 at every node $v \in V - \{s, t\}$.

The flow f, however, may not be of minimum value. Any augmenting path algorithm for finding a max-flow can be modified to obtain a decreasing path algorithm for finding a min-flow [15]. For this, we redefine the residual network N_f in this context: If $f(u, v) > 0$, then $c_f(u, v) =$

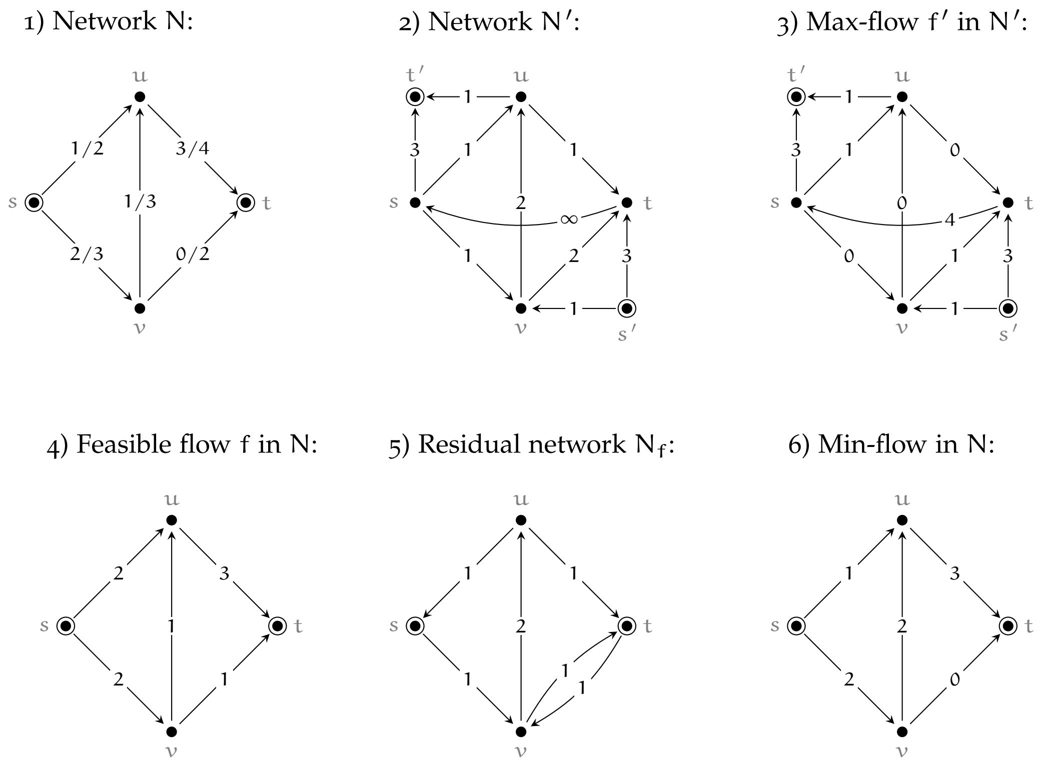


Figure 3: N is a network with edge demands. Each edge e is labeled with $l(e)/c(e)$ – its demand and its capacity. We want to find a min-flow in N, if there is one. For this, we transform N to a network N' without edge demands by setting the capacity of each edge e of N to $c(e) - l(e)$, adding two new nodes s', t', and connecting s' with each node v of positive net demand $d(v)$ and each node v with negative net demand $d(v)$ with t', each new edge with capacity $d(v)$. Moreover, we add an edge (t, s) of infinite capacity. Then, we compute a max-flow f' from s' to t' in N'. If f' saturates all edges leaving s', then there is a feasible flow in N which is $f(e) = f'(e) + l(e)$ for all edges e in N. In N_f, we identify a decreasing t-s path t-v-u-s of width 1 and decrease f by this path flow to obtain a min-flow in N.

$c(u,v) - f(u,v)$ and $c_f(v,u) = f(u,v) - l(u,v)$. Now, we may repeatedly reduce the value of a flow if there is a t-s path in the corresponding residual network. For an example, see Figure 3. The Integrality Theorem, which we have mentioned in the context of max-flows, holds for min-flows as well [15].

MULTI-TERMINAL GENERALIZATIONS. Ford and Fulkerson and others of the flow network pioneers have already considered the multi-terminal case. One straightforward generalization of s-t networks is to allow a set of source nodes and a set of sink nodes. Such a network, however, can be reduced to the single source, single sink case in the following sense. We add a new super source s', a super sink t', and edges (s', s) for every source s and (t, t') for every sink t. These new edges all have an infinite capacity. Now, f is a flow in the original network if, and only if, there is a flow f' in the altered s'-t' network that is equivalent to f at all edges existing in both networks.

Another generalization is the problem of determining the max-flow value for each pair of nodes in an undirected network. Consider an undirected network $N = (V, E, c)$ specified without any terminals. We would like to know the max-flow value $v(s,t)$ for each pair (s,t) of nodes where s and t are temporarily considered as source and sink. As N is undirected, we have $v(s,t) = v(t,s)$ for every pair of nodes. Gomory and Hu [36] demonstrated a technique to get around computing the max-flow value for each of the quadratic number of pairs. A central observation is that

$$v(x,z) \geqslant \min\{\, v(x,y), v(y,z) \,\} \text{ for all } x,y,z \in V \tag{21}$$

which leads to the fact that v may not have more than $|V| - 1$ distinct values. In fact, they show how to construct a tree for any given network that preserves the max-flow value for every pair of nodes. Preserving max-flow values between certain sets of nodes is an important aspect in this thesis.

SUBMODULARITY. A set function $f : 2^V \to \mathbb{R}$, where V is some set and 2^V its powerset, is a submodular set function if for all $X, Y \subseteq V$ it satisfies

$$f(X) + f(Y) \geqslant f(X \cup Y) + f(X \cap Y)\,. \tag{22}$$

It is well-known that the capacity function c defined by (7) is a submodular set function – with important implications. If X and Y are source sides of B-min-cuts for any $B \in \mathcal{B}$, then the sum of their capacities is at least as great as the sum of the capacities of $X \cup Y$ and $X \cap Y$. Since neither can have a capacity less than the min-cuts X and Y, (22) is satisfied with equality and hence, $X \cup Y$ as well as $X \cap Y$ are B-min-cuts, too. We may conclude that the set of source sides of all min-cuts of a network N is closed with respect to $\cap$ and $\cup$. The submodularity of the capacity function c is not bound to a fixed terminal bipartition. This will allow some more general implications for multi-terminal networks.

Another interesting fact about submodular functions is that any submodular set function that can be evaluated in polynomial time can also be minimized in polynomial time [38] using the ellipsoid method. Schrijver [74] shows how to minimize submodular functions without the use of the ellipsoid method and in strongly polynomial time.

MERGING NODES. A common technique throughout this thesis is to *merge nodes* of a network $N = (V, E, \mathcal{T}, c)$. Let $M \subseteq V$ be a non-empty set of nodes with $w \in M$ and $|M \cap \mathcal{T}| \leqslant 1$. If there is a terminal in M, then let w be this terminal. Otherwise, w is an arbitrary node of M. Further, let m be a function with

$$m(v) = \begin{cases} v & \text{if } v \in V - M \text{ and} \\ w & \text{else}\,. \end{cases} \tag{23}$$

Merging the nodes in M then yields a network $N' = (V', E', \mathcal{T}, c')$ where

$$V' = (V - M) \cup \{w\}, \tag{24}$$
$$E' = \left\{\, \big(m(x), m(y)\big) \mid (x,y) \in E \text{ and } \{x,y\} \not\subseteq M \,\right\}, \text{ and} \tag{25}$$
$$c'(x,y) = \sum_{\substack{u \in V \\ m(u)=x}} \sum_{\substack{v \in V \\ m(v)=y}} c(u,v) \text{ for all } (x,y) \in E'\,. \tag{26}$$

In case u, v are joined by an edge e, we sometimes refer to merging u, v as the *edge contraction* of e. However, merging nodes does not require any edges among them. Merging nodes may thus result in a network whose underlying graph is not a minor of the original one.

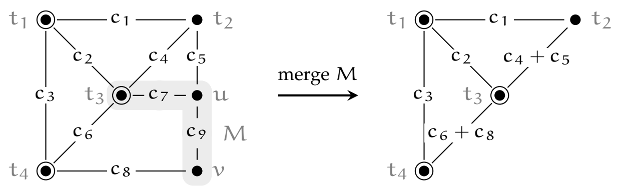


Figure 4: This example demonstrates the effect of merging the nodes $M = \{t_3, u, v\}$. The nodes u, v are removed and edge capacities that connected u or v to the outside of M are added to the edge capacities from t_3 to the outside of M. In the obtained network, terminal t_3 represents the former set M.

From a capacity perspective, merging two nodes u, v can be interpreted as setting $c(u,v) = c(v,u) = \infty$ (and previously adding the edges, if not existing). If there is a node $u \notin M$ connected to two different nodes $v, w \in M$, then merging the nodes of M also merges the edges (u, v) and (u, w). See Figure 4 for an example of nodes that are being merged.

LINEAR PROGRAMMING. We briefly introduce notation and some core concepts of linear programming. Classical textbooks devoted to this topic are *Linear Programming* by Chvátal [14] and *Combinatorial Optimization: Algorithms and Complexity* by Papadimitriou and Steiglitz [68]. Linear programming is a form of mathematical optimization used for problems whose requirements are represented by linear relationships. Such a problem looks for an optimal solution $\vec{x}$ that maximizes or minimizes an *objective function* $c(\vec{x})$ that is a linear combination of $\vec{x}$ and can thus be expressed as $\vec{c}^T\vec{x}$. Further, there are linear constraints on $\vec{x}$ that are commonly written as $A\vec{x} \leqslant \vec{b}$ where A is the *coefficients matrix*. Altogether, we will state a minimization or maximization problem like the following two linear programs (LP for short).

PROBLEM A (27)

$$\begin{aligned} \min \quad & \vec{b}^T\vec{y} \\ \text{s.t.} \quad & A^T\vec{y} \geqslant \vec{c}, \\ & \vec{y} \geqslant 0 \end{aligned}$$

PROBLEM B (28)

$$\begin{aligned} \max \quad & \vec{c}^T\vec{x} \\ \text{s.t.} \quad & A\vec{x} \leqslant \vec{b}, \\ & \vec{x} \geqslant 0 \end{aligned}$$

All LPs, whether they originally are minimization or maximization problems, can be expressed in *canonical form* as a maximization problem just like PROBLEM B.

Depending on whether we allow $\vec{x}$ to be a vector over the real numbers or whole numbers, the LP is called an *RLP* or *ILP*, respectively. While ILPs in general relate to NP-hard optimization problems, an optimal solution to an RLP can always be found in polynomial time. Often we deal with LPs where every component of $\vec{x}$ is either a 0 or a 1, a so-called 0-1 *ILP*. The *relaxation* of such an LP is an RLP where $0 \leqslant x_i \leqslant 1$ for every x_i of $\vec{x}$. While many NP-hard optimization problems can be modeled as a 0-1 ILP, the relaxed version can be optimized efficiently. This implies, of course, that the optimal solutions may differ.

Every LP, referred to as *primal problem*, possesses a counterpart, the so-called *dual problem*, to which it can be converted. The dual problem of the dual problem is the primal problem. Moreover, every feasible solution for an LP gives a bound for the value of an optimal solution to its dual problem. The famous *strong duality theorem* states that the values of the optimal solutions to the primal and dual problem are equivalent.

A *covering LP* is a LP of the form of PROBLEM A, i. e., LP (27), where A, $\vec{b}$, and $\vec{c}$ are non-negative. The dual of a covering LP is a *packing LP*, which is of the form of PROBLEM B, i. e., LP (28), where again A, $\vec{b}$, and $\vec{c}$ are non-negative. The way PROBLEM A and PROBLEM B are stated, they form a covering / packing duality.

3 MIN-CUT SET SYSTEMS

This chapter pushes towards a deeper understanding of the set of all min-cuts in networks. For each fixed terminal bipartition, a multi-terminal network encodes a set system of min-cuts. Depending on whether we represent a min-cut by its source side or by the set of edges cut, we either obtain a set system over V or over E. Picard and Queyranne [69] have already shown that there is a succinct representation for the set of all min-cuts. This representation can be used to quickly identify edges that are cut by some min-cut or to even enumerate all min-cuts. Both possibilities are of great importance for applications in the areas of sensitivity analysis and parametric analysis. These disciplines are related to questions concerning slight changes of the capacity function or the existence of edges, for example.

However, from the set system point of view, many questions are still open. How can these set systems be characterized? How hard is it to tell whether a network exists that encodes a given set system as the set of all its min-cuts? And how efficiently can such a network be constructed? In what way do the set systems encoded by a multi-terminal network restrict each other? In particular the last question turns out to be the core of problems arising in the domain of mimicking networks discussed in the next chapter.

For the 2-terminal case, we are able to fully settle these questions for set systems over V. We can quickly tell whether a given set system can be encoded in a network such that the source sides of all min-cuts are exactly the given set system. Moreover, the appropriate network can be constructed in polynomial time. When considering multi-terminal networks with more than 2 terminals, each terminal bipartition encodes a set system of its own. In this case, new effects arise. Two set systems, each of which can be encoded in a 2-terminal network, may not be able to be encoded in a single multi-terminal network – they may be incompatible in a certain sense. We present necessary conditions for a collection of set systems over V to be able to be encoded in a multi-terminal network.

We can ask the same set of questions when considering min-cuts that are represented by the edges they cut. In this case, however, the question regarding 2-terminal networks is already considerably harder to answer. We are unable to give a short characterization of these set systems as it was in the previous case. Due to this, we use a new technique to explore the restrictions of these set systems. We define new decision problems based on well-known problems whose input contains a set system. Instead of a set system, the new problems' input contains a 2-terminal network encoding the set system as the set of all min-cuts represented by the edges they cut. We settle the question of the complexity of most of these problems giving us more insight into the set of all min-cuts edge sets.

Interestingly, the newly defined problems defined by this framework find immediate and natural applications. HITTING SET, for example, leads to our problem called MIN-CUT HITTING SET which asks for a smallest set of edges such that each min-cut cuts an edge of this set. Having found such a set is of particular value when trying to increase a network's capacity by increasing the capacities of existing edges.

This chapter is broken down into four sections. Section 3.1 introduces the formal definitions for the mentioned set systems. A technique useful throughout this chapter is the *max-flow DAG* that is introduced in Section 3.2.

Section 3.3 deals with the node set systems that are defined by the source sides of all min-cuts. As already described on page 25, it is a well-known fact that these are closed with respect to $\cup$ and $\cap$ for every classical s-t flow network N due to the submodularity of the capacity function – and so is the arising node set system for every fixed $B \in \mathcal{B}$ for every multi-terminal network N. We establish a link in the opposite direction. In the multi-terminal case, we describe some necessary conditions for a set system to allow a corresponding network. Moreover, we define a problem whose complexity is linked to the question of how restrictive these node set systems are. This problem will be revisited in Chapter 4.

In Section 3.4, we focus on the edge set systems that arise when a min-cut is represented by the set of edges it cuts. We show necessary conditions for set systems to be the edge set system of a network. To further examine the edge set systems, we replace the set systems in the input of well-known NP-complete problems like HITTING SET and SET PACKING with a flow network and analyze the complexity of the obtained problems. While not all set systems can be encoded as the edge set system of a flow network, some large set systems can be encoded quite succinctly. We analyze the impact of these two opposing effects on the complexity of the obtained problems.

3.1 THE SETS OF ALL MIN-CUTS

In the 2-terminal case, we allow directed and undirected networks, but still only consider flow in s-t direction (i. e., from t_1 to t_2). For networks with more than 2 terminals, we consider undirected networks (except where explicitly stated differently). Such a network encodes $|\mathcal{B}|$ set systems.

▷ DEFINITION 2 (NODE SET SYSTEM AND EDGE SET SYSTEM): For a multi-terminal network $N = (V, E, \mathcal{T}, c)$ with $|\mathcal{T}| \geqslant 3$ and any non-trivial $V' \subset V$, we define its V' *node set system* and its V' *edge set system* as

$$\mathcal{V}_{V'}(N) = \{ S \mid (S,T) \text{ is a } V'\text{-min-cut in } N \} \text{ and} \tag{29}$$
$$\mathcal{E}_{V'}(N) = \{ E(S,T) \mid (S,T) \text{ is a } V'\text{-min-cut in } N \}, \tag{30}$$

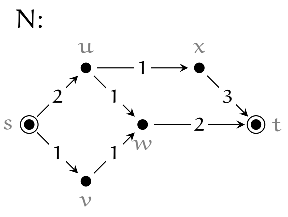
respectively. The *node set system* and *edge set system* of N are defined as

$$\mathcal{V}(N) = \{ \mathcal{V}_B \mid B \in \mathcal{B} \} \text{ and} \tag{31}$$
$$\mathcal{E}(N) = \{ \mathcal{E}_B \mid B \in \mathcal{B} \}, \tag{32}$$

respectively. For directed 2-terminal networks, the hierarchy is flattened, and the *node set system* and *edge set system* are defined as

$$\mathcal{V}(N) = \mathcal{V}_{\{s\}}(N) \text{ and} \tag{33}$$
$$\mathcal{E}(N) = \mathcal{E}_{\{s\}}(N). \tag{34}$$

We define these set systems not only for undirected networks, but also for directed ones even though the definition of $\mathcal{B}$ makes use of the symmetry present in undirected networks only. We do this in order to directly compare these set systems for directed and undirected networks. In particular, we would like to know whether there are set systems to which there is a directed, but no undirected network.



$$\mathcal{V}(N) = \big\{ \{ s \}, \{ s, u \}, \{ s, v \}, \{ s, u, v \}, \{ s, u, v, w \} \big\}$$

$$\mathcal{E}(N) = \big\{ \{ (s,u), (s,v) \}, \{ (u,x), (u,w), (s,v) \}, \{ (s,u), (v,w) \}, \{ (u,x), (u,w), (v,w) \}, \{ (u,x), (w,t) \} \big\}$$

Figure 5: The 2-terminal network N with a total of five min-cuts defines the two set systems $\mathcal{V}(N)$ and $\mathcal{E}(N)$. The edge (x,t) is the only edge not cut by any min-cut.

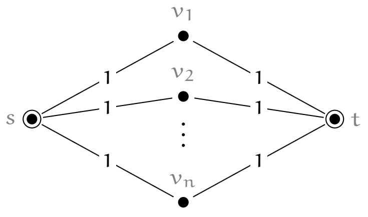


Figure 6: The depicted network is a network with the maximum number of min-cuts, which is $2^{|V-\mathcal{T}|}$. For each set $V' \subseteq (V - \mathcal{T})$ of non-terminals, $\{ s \} \cup V'$ is the source side of a min-cut.

See Figure 5 for an example of set systems defined by a directed 2-terminal network. Note that a V' set system is not only defined for $V' \subset \mathcal{T}$, but for all non-trivial $V' \subset V$. This generalization is handy at some time, but in the majority of the cases, V' is a set of terminals.

Undirected multi-terminal networks give rise to some symmetry. When B is considered to be the set of sources and $\mathcal{T} - B$ the sinks, then switching these roles does not change much – any flow that works in one situation can be reversed to work in the other situation. This, however, switches the direction of flow and switches source and sink sides. This impacts the corresponding node and edge set systems such that $\mathcal{E}_B(N) = \{ (v,u) \mid (u,v) \in \mathcal{E}_{\overline{B}}(N) \}$ and $\mathcal{V}_B(N) = \{ T \mid (S,T) \in \mathcal{V}_{\overline{B}}(N) \}$. Even though these pairs are different sets, they do not add any new information. On the contrary, $\mathcal{E}_{\overline{B}}$ is fully determined by $\mathcal{E}_B$.

For any given non-trivial $B \in \mathcal{B}$, finding a B-max-flow can be achieved via the multi-source multi-sink reduction described in Chapter 2 on page 24. Moreover, everything that we know or will prove for classical s-t networks also applies to each fixed B in this way.

3.2 THE MAX-FLOW DAG OF A NETWORK

In this section, we will focus on the 2-terminal case. Despite this restriction, we arrive at insights that also apply for each fixed terminal bipartition of a multi-terminal network.

An s-t network may have up to $2^{|V|-2}$ min-cuts (Figure 6) rendering it impossible to list all of them in polynomial time. It is, however, possible to find a short representation for all of them in polynomial time. In order to gain more insight into the structure of the set of all min-cuts in a network, consider the relation $R \subseteq V \times V$ with

$$(u,v) \in R \iff f(u,v) < c(u,v) \tag{35}$$

for a network N with respect to a max-flow f in N as defined by Picard and Queyranne [69]. Due to the skew symmetry (2), this definition is somewhat shorter than its original. A set $V' \subseteq V$ is a *closure* for R if for all nodes $u, v \in V$ we have that $v \in V'$ and $(u, v) \in R$ imply $u \in V'$.

▷ THEOREM 4 ([69]): An s-t cut (S, T) is a min-cut if, and only if, S is a closure for R.

Recall that the submodularity of the capacity function implies that the source sides of min-cuts are closed with respect to $\cap$ and $\cup$. Interestingly, the last theorem implies the same closedness since the intersection and the union of closures are again closures.

The relation R is actually nothing else but the edge relation in the residual network $N_f = (V, E_f, \{s, t\}, c_f)$ for any max-flow f. To see this, compare the definition of the relation R in (35) with the definition of the edge set of the residual graph in (11) on page 20. Hence, $R = E_f$ for all max-flows f.

Translating the closure property of R to E_f, we see that a node set S with $s \in S$ and $t \notin S$ is the source side of a min-cut if, and only if, there are no edges in E_f leaving S. This implies two well-known facts: As we have seen earlier, a min-cut exclusively cuts edges that are saturated by *every max-flow*. Additionally, there is no flow in f from T to S. If there was, then the residual network would contain an edge in S-T direction.

The residual network may thus not contain any edges crossing the min-cut from S to T. We may hence reformulate Theorem 4 such that an s-t cut (S, T) is a min-cut if, and only if, there are no edges leaving S in the residual graph N_f with respect to any max-flow f. This, in turn, leads directly to the statement that an s-t cut is a min-cut if, and only if, its capacity is 0 in the residual network with respect to any max-flow, as the following well-known lemma states.

▷ LEMMA 5: An s-t cut (S, T) is a min-cut if, and only if, $c_f(S, T) = 0$, where f is any max-flow.

In the remainder of this chapter, we will implicitly assume that N is connected in the sense that for every node v, there is a positive flow from s to t via v. Moreover, we assume s and t to be different nodes. In particular, this implies that each network has a positive capacity. Note that the connectedness assumption is not very strong. A node v that is not used by any flow can be removed including all incident edges. This neither changes any flow nor $\mathcal{E}(N)$. It may, however, change $\mathcal{V}(N)$. If this node v is connected to other nodes, then no min-cut will ever separate v from its neighbors as this unnecessarily increases the cut capacity.

▷ LEMMA 6: In any s-t network N, two nodes u, v belong to the same strongly connected component (SCC) in N_f with respect to any max-flow f if, and only if, there is no min-cut separating u and v.

Proof. Suppose u, v belong to the same SCC in N_f and let (S, T) be any min-cut separating u and v. Without loss of generality, we assume that $u \in S$ and $v \in T$. As u, v do belong to the same SCC in N_f, there is a u-v path in N_f such that (S, T) cuts an edge from S to T in N_f – a contradiction to Lemma 5.

For the opposite direction, we make use of our connectedness assumption. Due to this assumption, the network has a truly positive capacity. We will first show that for all nodes $v \in V$, there is a v-s path as well as a t-v path in N_f.

If t can be reached from v in N_f, then there is a v-s path in N_f since N has a positive capacity. So assume there is no v-t path in N_f. Let u be a node that can be reached from v in N_f (possibly $u = v$) but from which t cannot be reached. This is due to the fact that flow blocks all paths from u to t. This, however, implies that there is a v-u-t path in N_f.

Similarly, if there is flow from v to t, then there is a t-v path in N_f. If there is no flow from v to t, then because every s-v path or every v-t path is already blocked by flow. Now let u be a node from which v can be reached in N_f and from which there is flow to t (possibly $u = s$). Then, there is a t-u-v path in N_f.

If u, v do not belong to the same SCC, then without loss of generality, there is no u-v path in N_f. This implies that there is no u-t path in N_f as otherwise, there would be a u-t-v path – but u and v are not in the same SCC. Now, let S be the closure of $\{s, u\}$. As there is no s-t path and no u-t path, we have $s \in S$ and $t \notin S$. Hence, S is the source side of a min-cut. □

Picard and Queyranne [69] have proposed a similar partition of V, but without the connectedness assumption, there may be closures not composed of a set of strongly connected components. We will make use of this later. A nice corollary of the last lemma is that reachability between two nodes in N_f is independent of the choice of a max-flow f. Due to this, we can assign a function $\mu_N : V \to 2^V$ to each s-t network N such that $\mu_N(v)$ is the set of nodes of the SCC in N_f that contains v. Further, we may define the function $\mu_N : E \to 2^E$ as $\mu_N(u, v) = \big(\mu_N(u) \times \mu_N(v)\big) \cap E$. It maps an edge e to the set of edges that connect the same SCCs as e does. We will neglect the subscripts in μ_N when the appropriate network is clear from the context. In the case of multi-terminal networks, μ_B denotes the described functions for a fixed terminal bipartition $B \in \mathcal{B}$. Interestingly, μ does not only describe a partition of the nodes, but also of the min-cut edges.

▷ LEMMA 7: For any network N, $\mu : E \to 2^E$ partitions E. Further, for any two min-cut edges e_1, e_2 we have $e_2 \in \mu(e_1)$ if, and only if, every min-cut cuts e_1 if, and only if, it also cuts e_2.

Proof. Clearly, every edge e is part of some set as $e \in \mu(e)$. Let $\mu(u, v)$ and $\mu(w, x)$ be two distinct sets. Then

$$\mu(u, v) \cap \mu(w, x) = \big(\mu(u) \times \mu(v)\big) \cap \big(\mu(w) \times \mu(x)\big) \cap E = \emptyset$$

since $\mu : V \to 2^V$ is a partition of the nodes. We infer that $\mu : E \to 2^E$ partitions E.

Let $(u, v), (w, x)$ be two min-cut edges in N. If $(u, v) \in \mu(w, x)$, then $u, w \in \mu(u)$ and $v, x \in \mu(v)$. As a min-cut therefore separates u from v if, and only if, it separates w from x, the statement is true. □

Moreover, it now becomes clear that an edge (u, v) in N is a min-cut edge if, and only if, its reversed counterpart (v, u) connects two different SCCs in N_f. In order to show that every max-flow saturates an edge e if, and only if, e is cut by some min-cut, we first need the following lemma.

▷ LEMMA 8: Let f_1, f_2 be two flows of equal value in an arbitrary s-t network N. Then, there is a set of cycle flows O in N_{f_1} such that $f_1 + \sum_{f_o \in O} f_o = f_2$.

Proof. Let $f_3 : V \times V \to \mathbb{R}$ be a function with $f_1(e) + f_3(e) = f_2(e)$ for all $e \in E$. We show that f_3 is a flow in the residual network N_{f_1}. The capacities in N_{f_1} are $c_{f_1}(e) = c(e) - f_1(e)$ for all $e \in E_{f_1}$. For each edge $e \in E_{f_1}$, we have $f_3(e) = f_2(e) - f_1(e)$. Since $f_2(e) \leqslant c(e)$, we get $f_3(e) \leqslant c(e) - f_1(e)$ and the capacity constraint is fulfilled. Function f_3 is also skew symmetric since $f_3(u, v) = f_2(u, v) - f_1(u, v) = -\big(f_1(v, u) - f_2(v, u)\big) = -f_3(v, u)$. Finally, the flow conservation is satisfied at all nodes $v \in V - \mathcal{T}$ as the net flow $f_3(v)$ is the difference of the net flows $f_2(v)$ and $f_1(v)$.

Since f_3 is a flow in N_{f_1} of value 0, we may infer from the Flow Decomposition Theorem (Theorem 2 on page 21) that f_3 can be decomposed into a set of cycle flows O. This set is exactly the desired one. □

▷ LEMMA 9: For every s-t network N, an edge e is cut by some min-cut if, and only if, every max-flow saturates e.

Proof. Let $e = (u, v)$ be an edge in N. By Lemma 8, every max-flow f saturates e if, and only if, N_f does not contain any cycle using e. By Lemma 6, N_f contains a cycle using (u, v) if, and only if, u, v belong to the same SCC. If they do belong to the same SCC, then no min-cut cuts e. If they do not belong to the same SCC, then some min-cut cuts (u, v). □

In the following, we introduce the *edge cut network* as well as the *max-flow DAG* that both will prove to be useful techniques. While the edge cut network is defined for multi-terminal networks, the max-flow DAG is defined for the 2-terminal case only (or a fixed terminal bipartition).

▷ DEFINITION 3 (EDGE CUT NETWORK): The *edge cut network* $N_e = (V_e, E_e, \mathcal{T}, c_e)$ of a multi-terminal network $N = (V, E, \mathcal{T}, c)$ is the network obtained from N by merging all pairs of nodes not separated by any min-cut.

By the way we defined how to merge nodes on page 25, it follows that $V_e \subseteq V$ and $E_e \subseteq E$. Figure 8 on page 37 shows an example of an edge cut network. For every pair of terminals, there is a min-cut separating them and hence, the terminal set of N_e is the same as of N. It is not immediately clear, however, whether the edge cut network is unique. What happens if we vary the order in which mergeable nodes are merged? Do we still obtain the same edge cut network? Could merging two nodes force two edges to be identified with each other where one of them is contractible and the other is not?

Two nodes u, v can be merged for the edge cut network, if there is no terminal bipartition that contains a min-cut separating u, v. Let μ_B be the discussed partitioning functions for a specified terminal bipartition $B \in \mathcal{B}$. Nodes u, v may be merged, if $\mu_B(u) = \mu_B(v)$ for all $B \in \mathcal{B}$. The set

$$\mu(v) = \bigcap_{B \in \mathcal{B}} \mu_B(v) \tag{36}$$

is the intersection of all SCCs containing v. This is a generalization of function μ from the 2-terminal case to the multi-terminal case. The set $\{\, \mu(v) \mid v \in V \}$ hence is a partition of V and nodes u, v can be merged if, and only if, $\mu(u) = \mu(v)$, i. e., we merge all nodes in $\mu(v)$ for all $v \in V$. The yielded network N_e is unique regardless of the order in which we merge as μ is a partitioning of V.

Let $\mu(u), \mu(v)$ be two distinct sets in a multi-terminal network N and $e_1, e_2 \in E$ two distinct edges connecting nodes in $\mu(u)$ with nodes in $\mu(v)$ in N. Then these edges will be represented by a single edge in N_e. Analogously to the 2-terminal case, we define a function $\mu : E_e \to 2^E$ as $\mu(u, v) = \big(\mu(u) \times \mu(v)\big) \cap E$ to keep track of these represented edges.

The previous considerations lead straightforward to an algorithm to find the edge cut network.

▷ THEOREM 10: The edge cut network N_e of a given network N can be computed in time $O\big(2^{|\mathcal{T}|} \cdot p(n)\big)$, where n is the input length and p some fixed polynomial.

Proof. We construct N_e by computing a T'-max-flow $f_{T'}$ for every non-trivial $T' \subset \mathcal{T}$ and marking all edges connecting nodes of different SCCs in $f_{T'}$ as incontractible. Finally, we contract all edges not marked as incontractible. If an edge is cut by some T'-min-cut, then the algorithm will mark this edge as incontractible when it reaches T'. If an edge is not cut by any min-cut, then it always connects nodes within an SCC and hence is not marked. Concluding, the algorithm contracts an edge if, and only if, it is not cut by any min-cut.

This algorithm loops over all non-trivial elements of $2^{\mathcal{T}}$ and performs a polynomial time computation for each iteration. We thus have the claimed runtime. □

The algorithm of the proof is an FPT algorithm when $|\mathcal{T}|$ is seen as parameter. Hence, for classical s-t flow networks, the edge cut network can be computed in polynomial time.

Some edges are quite obviously never cut by any min-cut. If an edge is a min-cut edge, it is saturated by every max-flow. For some edges, however, we can locally verify that it is not saturated by any flow implying that it is not cut by any min-cut. Based on this observation, we say that an edge (u,v) of a directed or undirected network *dominates* u if its capacity is truly greater than all capacities reaching u, and it *dominates* v if $c(u,v)$ is truly greater than all capacities leaving v. Formally, (u,v) dominates u, if $c(u,v) > \sum_{(w,u)\in E,\, w\neq v} c(w,u)$, and (u,v) dominates v, if $c(u,v) > \sum_{(v,w)\in E,\, w\neq u} c(v,w)$. No min-cut will ever cut a dominating edge if it dominates a non-terminal as its capacity can be lowered simply by moving the dominated node to the other side. Hence, all such dominating edges can be contracted. This notion can even be generalized. Algorithm 1 is a heuristic and involves the computation of at most $|V|$ many min-cuts and is thus a polynomial time algorithm. It guarantees to contract all dominating edges among possibly more edges that are no min-cut edges. It can be used as a preprocessing that might already suffice before running the FPT algorithm described in the proof of Theorem 10 to find the edge cut network. Additionally, a modified version of this algorithm will be useful in the context of mimicking networks in Chapter 4.

Algorithm 1: Heuristic to find the edge cut network of an undirected network.

```
INPUT: An undirected network $N = (V, E, \mathcal{T}, c)$.
OUTPUT: A minor $N'$ obtained by contracting edges not cut by any min-cut.
1: $N' \leftarrow N$
2: FOR every $v \in V$ DO
3:     Compute the $v$-min-cut $(S_v, T_v)$ with minimum $|S_v|$ and $v \in S_v$.
4:     Merge all nodes in $S_v$.
5: END FOR
6: RETURN $N'$
```

▷ LEMMA 11: Let N be an undirected network, $B \in \mathcal{B}$, and $v \in V$. If $\emptyset \neq S_B \cap S_v$ and $v \in T_B$, then $S_B - S_v$ is a B-min-cut, and $S_v - S_B$ is a v-min-cut.

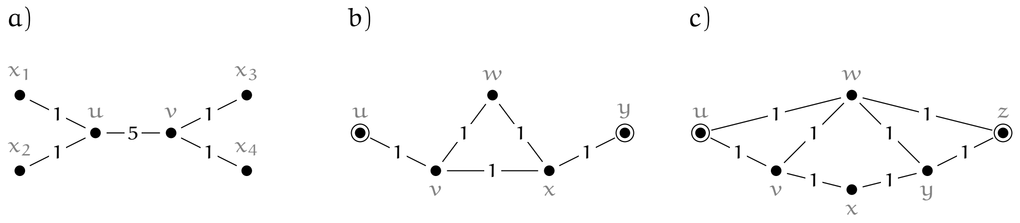
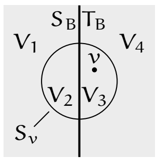


Figure 7: a) Edge (u, v) with $u, v \notin \mathcal{T}$ is a dominating edge and hence not cut by any min-cut. It is contracted by Algorithm 1. b) The nodes v, w, x can be merged to obtain the edge cut network. Algorithm 1 accomplishes this as the v-min-cut (S_v, T_v) with minimum $|S_v|$ and $v \in S_v$ is $S_v = \{v, w, x\}$. Note, that this network does not contain any dominating edges showing that Algorithm 1 generalizes the notion of dominating edges. c) In this example, v, w, x, y can be merged to obtain the edge cut network, but Algorithm 1 fails to do so.

Proof. Let $V_1 = S_B - S_v$, $V_2 = S_v - T_B$, $V_3 = S_v - S_B$, and $V_4 = T_B - S_v$ as depicted. Note that by the definition of a v-min-cut, $S_v \cap S_B$ does not contain any terminals no matter whether v is a terminal or not. Since S_B is a B-min-cut, we know that $c(V_1) \geqslant c(V_1 \cup V_2)$ and hence $c(V_1, V_2) \geqslant c(V_2, V_3) + c(V_2, V_4)$. Moreover, S_v is a v-min-cut and therefore $c(V_3, V_2) \geqslant c(V_2, V_4) + c(V_2, V_1)$. Since N is undirected, we have $c(V_1, V_2) = c(V_2, V_1)$ and $c(V_2, V_3) = c(V_3, V_2)$. But then, the two inequalities are only satisfied if $c(V_1, V_2) = c(V_2, V_3)$ and $c(V_2, V_4) = 0$, which in turn implies that the two inequalities are satisfied with equality. The claim follows immediately. □



▷ LEMMA 12: Algorithm 1 finds a minor N_m for a given undirected network N by contracting only edges that are not cut by any min-cut.

Proof. Let $v \in V$ be some node and S_v the v-min-cut with $v \in S_v$ and minimum S_v. For the sake of contradiction, we assume that there is a B-min-cut (S_B, T_B) for some $B \in \mathcal{B}$ such that $\emptyset \neq S_B \cap S_v \neq S_v$. Without loss of generality, we assume that $v \in S_v - S_B$. By Lemma 11, this implies that there is a true subset of S_v that is a v-min-cut also. This contradicts the assumption, that S_v is a v-min-cut with minimum $|S_v|$. Furthermore, the resulting network is a minor of N, as every v-min-cut induces a connected component. □

That Algorithm 1 is just a heuristic to find the edge cut network and that there are networks where the output of the algorithm is not the edge cut network can be seen in Figure 7.

In the classical case of $|\mathcal{T}| = 2$, we go one step further and orient all edges of the edge cut network to obtain the *max-flow DAG*.

▷ DEFINITION 4: The *max-flow DAG* $N_d = (V_d, E_d, \{s, t\}, c_d)$ of an s-t network $N = (V, E, \{s, t\}, c)$ is a network obtained from N by orienting all edges of the edge cut network N_e of N in the flow direction of f for an arbitrary max-flow f. The capacity function c_d is equal to c_e, but 0 for edges not in E_d.

Again, we have $V_d \subseteq V$ and $E_d \subseteq E$. From a technical perspective, orienting the edges simply refers to removing either (u, v) or (v, u) if both exist. See Figure 8 for an example of a max-flow DAG.

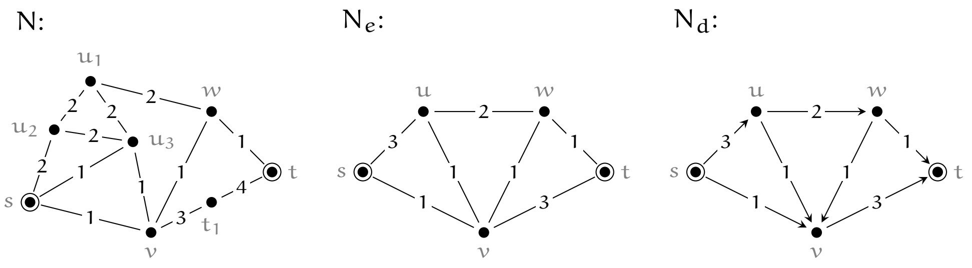


Figure 8: The s-t network N contains several edges that are not cut by any min-cut. Contracting these edges merges $\{u_1, u_2, u_3\}$ and $\{t_1, t\}$, and yields the edge cut network N_e of N. By directing all edges of N_e in the direction of the max-flow, we obtain the max-flow DAG N_d of N. We now have $\mu(u) = \{u_1, u_2, u_3\}$ and $\mu(s, u) = \{(s, u_2), (s, u_3)\}$.

▷ LEMMA 13: For every s-t network N, the max-flow DAG N_d is a unique DAG and can be constructed in polynomial time.

Proof. As the edge cut network N_e is unique, it remains to be shown that all max-flows use all edges of N_e in the same direction. We know that every min-cut edge is saturated by every max-flow. A max-flow that saturates (u, v) and (v, u) can easily be altered to be a max-flow that uses neither edge. This implies that neither one belongs to N_e.

Furthermore, a max-flow in an edge cut network does not contain any cycles. If it did, then the corresponding residual network would contain an SCC consisting of the nodes of that cycle – but each SCC has been merged to a single node to obtain the edge cut network. Hence, we have arrived at a directed, acyclic flow network. Finding a max-flow and directing all edges in the direction of its flow can be done in polynomial time. □

The two functions $\mu : V \to 2^V$ and $\mu : E \to 2^E$ tell us for a node or an edge in N_d which nodes or edges of N have been merged to obtain it. Refer to Figure 8 for an example. We will now introduce two more functions that link min-cuts in N and N_d, represented as node sets and as edge sets. They are fully based on the already discussed function μ.

▷ DEFINITION 5: We define $\tilde{\mu} : \mathcal{E}(N_d) \to \mathcal{E}(N)$ as

$$\tilde{\mu}(E') = \bigcup_{e \in E'} \mu(e) \tag{37}$$

and $\tilde{\mu} : \mathcal{V}(N_d) \to \mathcal{V}(N)$ as

$$\tilde{\mu}(S') = \bigcup_{v \in S'} \mu(v)\,. \tag{38}$$

▷ LEMMA 14: The functions $\tilde{\mu} : \mathcal{E}(N_d) \to \mathcal{E}(N)$ and $\tilde{\mu} : \mathcal{V}(N_d) \to \mathcal{V}(N)$ are bijective.

Proof. We claim for both functions that there are inverse functions defined for every element of $\mathcal{E}(N)$ and $\mathcal{V}(N)$, respectively. These are

$$\tilde{\mu}^{-1}(E') = E_d \cap E' \text{ for all } E' \subseteq E \text{ and} \tag{39}$$

$$\tilde{\mu}^{-1}(S') = V_d \cap S' \text{ for all } S' \subseteq V\,, \tag{40}$$

and hence simply the restrictions of the edge and node sets to those edges and nodes also present in the corresponding max-flow DAG. To see that this suffices, let $V' \in \mathcal{V}(N)$ be any min-cut in N. Then, we have

$$\tilde{\mu}\big(\tilde{\mu}^{-1}(V')\big) = \tilde{\mu}\big(V_d \cap V'\big) = \bigcup_{\nu \in V_d \cap V'} \mu(\nu) = V', \tag{41}$$

since $V_d \cap V'$ restricts V' to those in V_d – these are exactly those nodes representing the nodes of V' in N_d. Function μ re-expands these nodes to obtain V' again.

Knowing this, it is immediate to see how the according functions for the edges work. For any min-cut $E' \in \mathcal{E}(N)$ in N, $\tilde{\mu}^{-1}(E')$ is the restriction of its edges to the representing edges in E_d. Function $\tilde{\mu}$ then re-expands these edges to E'. □

This lemma implies that $E' \subseteq E_d$ ($S' \subseteq V_d$) is a min-cut in N_d if, and only if, $\tilde{\mu}(E')$ ($\tilde{\mu}(S')$) is a min-cut in N. For the same arguments, the statement also holds true for N and its edge cut network N_e. The max-flow DAG of a network N in combination with function $\mu : V \to 2^V$ contains the same information concerning the min-cuts of a network as N itself. As the max-flow DAG commonly is obtained from N, we also have μ and hence a short representation for the set of all min-cuts. In other words, $\mathcal{V}(N) = \{\, \mu(S) \mid S \in \mathcal{V}(N_d)\,\}$.

The following lemmata show some more useful properties of max-flow DAGs.

▷ LEMMA 15: Let N be an arbitrary network and N_d the corresponding max-flow DAG. Then, every s-t path in N_d is cut exactly once by every min-cut in N_d.

Proof. Every min-cut cuts every s-t path in N_d at least once, as it is an s-t cut. Suppose there is a min-cut (S, T) that cuts some s-t path p twice. If so, then we can find a flow decomposition of a max-flow following the proof of Theorem 2 (page 21) that contains a path flow f_p. Hence, the value of f_p is counted twice for the capacity of (S, T) – a contradiction. □

In general, it is well possible that a network allows an s-t path p that is cut more than once by some min-cut. In that case, however, no max-flow allows a flow decomposition containing a path flow along p. See Figure 14 on page 60 for such a network.

▷ LEMMA 16: Every set $S \subset V_d$ with $s \in S$ and $t \notin S$ is a min-cut in N_d if, and only if, every edge with exactly one endpoint in S leaves S.

Proof. Let $S \subset V_d$ with $s \in S$ and $t \notin S$ be a set of nodes such that every edge with exactly one endpoint in S leaves S. Let f be any max-flow in N_d. This flow saturates all edges and hence, $c_f(S, T) = 0$. By Lemma 5, (S, T) is a min-cut in N_d.

Now let (S, T) be a min-cut in N_d. This implies that there is a max-flow f with $c_f(S, T) = 0$. Hence, all edges with exactly one endpoint in S leave S. □

A DAG has the advantage that not only shortest, but also longest paths can be computed efficiently. Starting a breadth-first search at s, we have found the shortest path when we hit t for the first time, and the longest path when we hit t for the last time, as all paths starting at s end at t. This fact will later be exploited.

3.3 NODE SET SYSTEMS

Definition 2 on page 30, that defines node set systems, raises some questions. Given a set system $\mathcal{V}$, is there a network N with $\mathcal{V}(N) = \mathcal{V}$? What properties does $\mathcal{V}(N)$ have? We will first discuss the 2-terminal case where these questions are answered. The multi-terminal case complicates the situation. We present necessary conditions for a node set system $\mathcal{V}$ to allow a network N with $\mathcal{V}(N) = \mathcal{V}$.

3.3.1 *The 2-Terminal Case*

As described on page 25, the submodularity of the capacity function implies that both the intersection and the union of the source sides of any two min-cuts yield the source side of another min-cut. The opposite direction obviously cannot be true. As soon as a set system that is closed with respect to $\cap$ and $\cup$ contains two disjoint sets, it also contains the empty set. The empty set, however, cannot be the source side of a min-cut in any network, as it always has to contain at least the source. With the following theorem, we still establish a link in the opposite direction by understanding the sets as source sides that explicitly leave out the source. The idea of the proof is to first construct a network N' that can be viewed as the residual network of some network with respect to a corresponding max-flow f. As we know from Lemma 6, such a network can be partitioned into SCCs. As such a network does not fulfill the connectedness assumption we extend the idea to construct a connected network N with the property that $N_f = N'$.

▷ THEOREM 17: For every set system $\mathcal{V}$ over a finite universe U, a flow network N with nodes $V = U \cup \{s, t\}$ $(s, t \notin U)$ and $\mathcal{V}(N) = \{V' \cup \{s\} \mid V' \in \mathcal{V}\}$ exists if, and only if, $\mathcal{V}$ is closed with respect to $\cap$ and $\cup$. If such network N exists, then another such network with positive capacity can be constructed in polynomial time.

Proof. We already know that every node set system is closed with respect to $\cap$ and $\cup$. This still holds true even if the source node is removed from every element of that set system.

For the opposite direction, we will first show how to construct a network N' with $c(N') = 0$ in the desired time and proof that $\mathcal{V}(N') = \{V' \cup \{s\} \mid V' \in \mathcal{V}\}$. Then, we show how this network can be augmented to obtain a network N with positive capacity such that N' equals the residual network N_f for some max-flow f in N.

Let $L = \bigcap_{S \in \mathcal{V}} S$, $M = \left(\bigcup_{S \in \mathcal{V}} S\right) - L$, and $R = U - (L \cup M)$. L contains the nodes present in the source side of every min-cut, while R contains all nodes that are in no source side. Construct a flow network $N' = \big(U \cup \{s, t\}, E', \{s, t\}, c'\big)$ with edges

- (s, v) and (v, s) if $v \in L$,
- (v, t) and (t, v) if $v \in R$, and
- (x, y) if $x, y \in M$, and $x \in S$ implies $y \in S$ for all $S \in \mathcal{V}$

of positive capacity.

As there is no s-t path, the only valid flow is the 0-flow. Let $\mu(v)$ be the SCCs that have emerged so far and v a node in $\mu(v)$ that represents $\mu(v)$.

More edges are added now depending on the *indegree* in and *outdegree* out of a strongly connected component defined as

$$\begin{aligned} \mathrm{in}(v) &= |\{\,(u,v) \mid u \notin \mu(v),\ v \in \mu(v)\,\}| \text{ and} && (42)\\ \mathrm{out}(v) &= |\{\,(v,u) \mid u \notin \mu(v),\ v \in \mu(v)\,\}|\,. && (43)\end{aligned}$$

For every SCC $\mu(v)$ we add the edges

- (t,v) if $\mathrm{in}\big(\mu(v)\big) = 0$ and
- (v,s) if $\mathrm{out}\big(\mu(v)\big) = 0$.

Note that edges from t to a node v or edges from a node v to s do not affect any min-cut. The capacity of each edge is set to an arbitrary, but positive value. This concludes the construction of N'. Since t is not reachable from s, the maximal flow is 0 and hence, there are no edges crossing any min-cut in S-T direction. In the following, we show that $\mathcal{V}(N') = \{\,V' \cup \{\,s\,\} \mid V' \in \mathcal{V}\}$.

First, we show that any set $S \in \mathcal{V}$ is a source side of an s-t min-cut. By definition $L \subset S$ and $R \cap S = \emptyset$, so edges from s and edges to t do not cross any min-cut. Edges within M do not cross a cut by definition, since for any such edge (x,y), if x is in S, then y must also be in S.

Now we show that if Q is a source side of an s-t min-cut, Q must be in $\mathcal{V}$. As there are no edges crossing a minimal cut, $L \subset Q$ and $R \cap Q = \emptyset$. If $Q = L$, the proof is done since $L \in \mathcal{V}$. Otherwise, let

$$Q' = \bigcup_{x \in Q} \bigcap_{\substack{S \in \mathcal{V} \\ x \in S}} S\,. \tag{44}$$

By definition, $Q \subseteq Q'$. Suppose there is an element $y \in Q'$ that is not in Q. Then $y \in M$ and there is an element $x \in Q$ such that for all $S \in \mathcal{V}$, $y \in S$ if $x \in S$. Then, $x \in M$, since otherwise $x \in L$ and $L \in \mathcal{V}$ is a counterexample. Hence, (x,y) crosses a min-cut – a contradiction. We conclude that $Q = Q'$ can be expressed as union and intersection of sets in $\mathcal{V}$ and hence $Q \in \mathcal{V}$.

Since all these construction steps can be achieved in polynomial time, the first part of the proof is done. Next, we alter N' to obtain a connected network $N = (V, E, \{\,s,t\,\}, c)$ with $\mathcal{V}(N) = \{\,V' \cup \{\,s\,\} \mid V' \in \mathcal{V}\}$ and $c(N) > 0$. For this, we first construct network $N'' = (V, E, \{\,s,t\,\}, l, c'')$ with lower bounds. Its edges

$$E = \big\{\,(u,v) \mid (v,u) \in E'\,\big\} \tag{45}$$

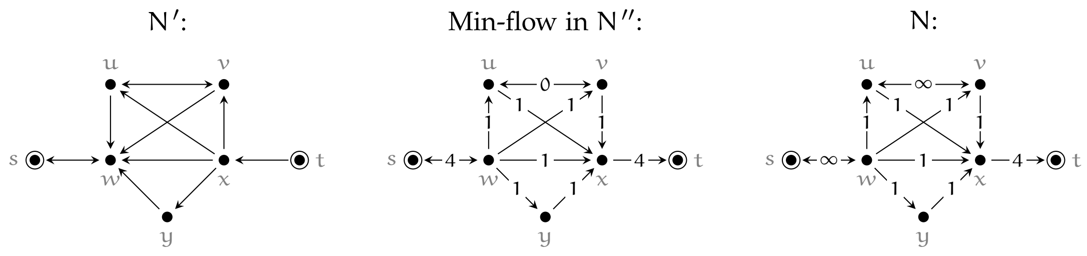
are the same as in N', but reversed. The lower bounds are

$$l(u,v) = \begin{cases} 0 & u,v \text{ belong to the same SCC in } N''; \\ 1 & \text{otherwise;} \end{cases} \tag{46}$$

and the capacity is $c(u,v) = \infty$ for all $(u,v) \in E$. Then, let f'' be a min-flow in N''. This flow is now used to construct network $N = (V, E, \{\,s,t\,\}, c)$ by setting

$$c(u,v) = \begin{cases} \infty & u,v \text{ belong to the same SCC;} \\ \max\{\,0, f''(u,v)\,\} & \text{otherwise}\,. \end{cases} \tag{47}$$

This concludes the construction of network N. Due to the construction of N' and the reversal of the edges, for every node v in N there is an s-t flow via v. Moreover, since the min-flow in N'' has a value greater 0, we have



$\mathcal{V} = \{\{s,w\},\{s,u,v,w\},\{s,w,y\},\{s,u,v,w,y\},\{s,u,v,w,x,y\}\}$
$L = \{s,w\}$, $M = \{u,v,x,y\}$, $R = \{t\}$

Figure 9: Given $\mathcal{V}$, the task is to construct a connected network N with $\mathcal{V}(N) = \mathcal{V}$. Following the proof of Theorem 17, we first construct network N' with arbitrary positive capacities on each edge, which already fulfills $\mathcal{V}(N) = \mathcal{V}$. In a second step, all edge directions are reversed, we set $c(e) = \infty$ for every edge, and a lower bound $l(e) = 0$ for every edge within an SCC, and $l(e) = 1$ for every other edge. We call the resulting network N''. Concluding, we obtain the directed network N from N'' and a min-flow f'' in N'' by setting the capacity of each edge e to ∞, if it belongs to a strongly connected component to N', and to $\max\{0, f''(e)\}$ otherwise. The source sides of all min-cuts in the connected network N is now exactly $\mathcal{V}$ as N_f is topologically identical to N' for any max-flow f in N.

$c(N) > 0$. Clearly, all construction steps can be accomplished in polynomial time.

It remains to be shown that for any max-flow f in N, we have $N_f = N'$ (with according positive capacities). This suffices, since we have already shown that $\mathcal{V}(N') = \{V' \cup \{s\} \mid V' \in \mathcal{V}\}$. As we have set the capacities of all edges e connecting two strongly connected components to $f''(e)$, this flow f'' is now a max-flow in N. Clearly, it saturates all edges connecting two strongly connected components and no edges within strongly connected components. Hence, N' with an appropriately chosen capacity function is the residual network for N with respect to the max-flow f''. □

Of course the same construction works without the addition of a new source, if there is a node that is present in the source side of every min-cut. See Figure 9 for an example of the construction just described.

While a flow network N uniquely defines $\mathcal{V}(N)$, there may be many flow networks that correspond to a given node set system in the sense of Theorem 17. The proof of Theorem 17 constructs two, a third and undirected network can easily be derived from the constructed network N by adding an edge (u,v) for every edge (v,u) and setting $c(u,v) = c(v,u)$. This neither alters the max-flow nor the residual network. We may thus infer the following corollary.

▷ COROLLARY 18: The set of all node set systems of directed 2-terminal networks and the set of all node set systems of undirected 2-terminal networks are equivalent.

3.3.2 *The Multi-Terminal Case*

We now generalize the setting from two terminals to more than two terminals. Suppose we are given a set $\mathcal{V}$ of set systems and the question is whether there is a multi-terminal network N with $\mathcal{V}(N) = N$ where $|\mathcal{T}| > 2$.

Such a set $\mathcal{V}$ contains several of those set systems just described in the 2-terminal case. The question is whether these are compatible in a certain sense. In the following, we present necessary conditions for a set $\mathcal{V}$ such that there is a network N with $\mathcal{V}(N) = \mathcal{V}$.

If $\mathcal{V}$ is a set system such that $\mathcal{V}(N) = \mathcal{V}$ for some network N, then there is a bijection $\pi_{\mathcal{V}} : \mathcal{B} \to \mathcal{V}$ with $\pi(B) = S$ if, and only if, $S \cap \mathcal{T} = B$. This bijection simply tells us which set system of $\mathcal{V}$ belongs to which terminal bipartition B and vice versa. We will neglect the subscript of π if it is clear from the context.

▷ LEMMA 19: Let $\mathcal{V}$ be a node set system such that there is a network N with $\mathcal{V}(N) = \mathcal{V}$. Further, let $B', B'' \in \mathcal{B}$ be two terminal bipartitions with $B' \subset B''$. Now, if $S_1 \in \pi(B')$, $S_2 \in \pi(B'')$, then $S_1 \cap S_2 \in \pi(B')$ and $S_1 \cup S_2 \in \pi(B'')$.

Proof. Due to $B' \subset B''$, we know that every $B' \cap B''$-cut is also a B'-cut, and every $B' \cup B''$-cut is also a B''-cut. Recall the submodularity of the cut capacity function, which in this context implies

$$c(S_1) + c(S_2) \geqslant c(S_1 \cap S_2) + c(S_1 \cup S_2)\,. \tag{48}$$

Since S_1 and S_2 are min-cuts for their corresponding terminal bipartitions, we know that $S_1 \cap S_2$ and $S_1 \cup S_2$ are min-cuts as well. □

Some set systems in $\mathcal{V}$ may force the existence of certain edges while some other set systems in $\mathcal{V}$ may rule out some edges. If this applies to the same edge, then these elements are not compatible in such a way, that there is no corresponding network.

▷ LEMMA 20: Let N be a network and $B \in \mathcal{B}$. Further, let $X_1, X_2, X_3 \in V - \mathcal{T}$ be non-empty and disjoint non-terminal sets such that

- $S_1 = B \cup X_1 \in \mathcal{V}_B(N)$,
- $S_2 = B \cup X_1 \cup X_2 \in \mathcal{V}_B(N)$, and
- $S_3 = B \cup X_1 \cup X_2 \cup X_3 \in \mathcal{V}_B(N)$.

Then, $c(X_2, X_3) > 0$ if, and only if, $B \cup X_2 \notin \mathcal{V}_B(N)$.

Proof. Let $R = V - B - X_1 - X_2 - X_3$. Since $S_1, S_2 \in \mathcal{V}_B(N)$, we have

$$c(S_1) = c(S_2) \iff c(B \cup X_1, X_2) = c(X_2, X_3 \cup R)\,. \tag{49}$$

Due to $S_2, S_3 \in \mathcal{V}_B(N)$, we get

$$c(S_2) = c(S_3) \iff c(B \cup X_1 \cup X_2, X_3) = c(X_3, R)\,. \tag{50}$$

Now we compare the capacities of the min-cuts $B \cup X_1 \cup X_2$ and $B \cup X_1 \cup X_3$.

$$\begin{aligned} &c(B \cup X_1 \cup X_2) - c(B \cup X_1 \cup X_3) \\ &= c(B \cup X_1, X_2) + c(X_3, X_2 \cup R) - c(X_2, X_3 \cup R) - c(B \cup X_1, X_3) \\ &= c(B \cup X_1, X_2) + c(X_3, X_2) + c(B \cup X_1 \cup X_2, X_3) \\ &\quad - c(B \cup X_1, X_2) - c(B \cup X_1, X_3) \\ &= 2 \cdot c(X_2, X_3)\,. \end{aligned} \tag{51}$$

The last equality follows from (49) and (50). Now we see that $B \cup X_1 \cup X_3 \in \mathcal{V}_B(N)$ if, and only if, $c(X_2, X_3) = 0$. □

The last lemma shows us when min-cuts rule out or force the existence of certain edges. If a set system of a node set system $\mathcal{V}$ contains two crossing min-cuts S_1 and S_2, then the last lemma implies that $c(S_1 - S_2, S_2 - S_1) = 0$ (and, symmetrically, $c(S_2 - S_1, S_1 - S_2) = 0$). Hence, there must be no edge joining nodes from X_2 and X_3 in any network N with $\mathcal{V}(N) = \mathcal{V}$. As a consequence, a network with maximum number of min-cuts as in Figure 6 on page 31 cannot have any edges connecting non-terminals.

On the other hand, if $\mathcal{V}$ contains a set system with three min-cuts S_1, S_2, S_3 containing each other as described in Lemma 20 without a min-cut crossing S_2, then there needs to be an edge with positive capacity joining nodes from X_2 and X_3.

If a node set system contains set systems where this happens for the same pair of nodes at once, then clearly, there can be no network – see the following Example 1.

▷ EXAMPLE 1: In this example, Lemma 19 does not help for $\mathcal{V}_1$, but Lemma 20 tells us that the t_1-min-cuts demand an edge between u and v. At the same time, it tells us that the $\{t_1, t_2\}$-min-cuts can only exist if there is no edge between u and v. Hence, there is no network N with $\mathcal{V}(N) = \mathcal{V}_1$ even though there is an s-t network for each single set system of $\mathcal{V}_1$. Symmetrically, Lemma 20 does not apply to $\mathcal{V}_2$, but Lemma 19 rules out the existence of a network N with $\mathcal{V}(N) = \mathcal{V}_2$ since it demands the existence of the min-cuts $\{t_1\}$, $\{t_1, t_2, u, v\}$, and $\{t_1, t_3, u\}$.

$$\begin{aligned}\mathcal{V}_1 = \Big\{ & \underbrace{\big\{\{t_1\}\big\}}_{t_1\text{-min-cuts}}, \\ & \underbrace{\big\{\{t_1,t_2\},\{t_1,t_2,u\},\{t_1,t_2,v\},\{t_1,t_2,u,v\}\big\}}_{\{t_1,t_2\}\text{-min-cuts}}, \\ & \underbrace{\big\{\{t_1,t_3\},\{t_1,t_3,u\},\{t_1,t_3,u,v\}\big\}}_{\{t_1,t_3\}\text{-min-cuts}} \Big\}. \end{aligned} \tag{52}$$

$$\mathcal{V}_2 = \Big\{ \underbrace{\big\{\{t_1,u\}\big\}}_{t_1\text{-min-cuts}}, \underbrace{\big\{\{t_1,t_2,v\}\big\}}_{\{t_1,t_2\}\text{-min-cuts}}, \underbrace{\big\{\{t_1,t_3\}\big\}}_{\{t_1,t_3\}\text{-min-cuts}} \Big\}. \tag{53}$$

We now introduce a problem that we call MIN-CUT MINIMUM TYPE SELECTION. In the context of mimicking networks in Chapter 4 we will see that it is equivalent to another problem that will play a prominent role in that chapter. The lack of understanding this problem was part of the reason for the research involving min-cut set systems.

For this problem, we represent min-cuts by bit vectors for which we need a total order imposed over the set of all nodes of a network. This order may be arbitrary, it could, for example, be the order in which the nodes appear in the encoding of the network. Once an order is fixed, the *min-cut vector* of a min-cut (S, T) is a bit vector $\{0, 1\}^{|V|}$ where the i^{th} bit represents the i^{th} node of the network. A node is represented by a `0` if, and only if, it is on the same side as t_1.

▷ PROBLEM 1 (MIN-CUT MINIMUM TYPE SELECTION):

Input: A tuple (N, l) of an undirected multi-terminal network N and a number l.

Question: Is there a choice of min-cuts, one for each $B \in \mathcal{B}$, such that the matrix composed row-wise of the corresponding min-cut vectors does not contain more than l distinct columns?

We will revisit the complexity of this problem when we talk about mimicking networks in Chapter 4. Right now, we want to focus on a different aspect of it. Consider the problem

▷ PROBLEM 2 (MINIMUM TYPE SELECTION):

Input: A tuple $(\mathcal{S}, l)$ of a family $\mathcal{S}$ containing sets of bit vectors of identical length and a number $l \geqslant 0$.

Question: Is there a choice of exactly one vector per set such that a matrix composed row-wise of these vectors contains at most l distinct column vectors?

These two problems are quite similar. To solve either one, we need to choose exactly one bit vector from each of the specified sets of bit vectors such that the matrix composed row-wise of these chosen vectors does not contain more than a specified number of distinct columns. While MINIMUM TYPE SELECTION presents these sets to choose from in a quite explicit way, MIN-CUT MINIMUM TYPE SELECTION implicitly encodes these sets in the min-cuts of a network.

These encoded sets are quite restricted, they cannot be arbitrary sets of vectors. In particular, these sets are closed with respect to the logical `and` and `or` operator, see Theorem 17. The main reason for this is the fact that the capacity function of min-cuts is a submodular function. This makes the problem potentially easier. Opposing this restriction, a network may encode exponentially in its size many min-cuts for a fixed terminal bipartition. This kind of succinctness makes the problem potentially harder. It seems not clear which of these two effects dominates. This also implies that the complexity of either problem does not bound the complexity of the other. At least not without more knowledge about these node set systems.

For now, we will only show that MINIMUM TYPE SELECTION is NP-complete by a reduction from the NP-complete PARTITION INTO HAMILTONIAN SUBGRAPHS problem [34].

▷ PROBLEM 3 (PARTITION INTO HAMILTONIAN SUBGRAPHS):

Input: A directed graph $G = (V, E)$.

Question: Is it possible to partition the vertices of G into disjoint sets $V_1, V_2, \dots, V_k$ for some k, such that each V_i contains at least three vertices and induces a subgraph of G that contains a Hamiltonian cycle?

▷ THEOREM 21: MINIMUM TYPE SELECTION is NP-complete.

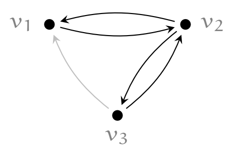
Proof. We have MINIMUM TYPE SELECTION $\in$ NP, as a solution can easily be guessed and verified. The reduction function to prove hardness creates a family $\mathcal{S}_G = \{S_1, S_2, \dots, S_n\}$ such that each vector $v_j \in S_i$ encodes an edge that is directed from v_i to some other node. The number of components of each such vector is $(n^2-n)/2 + 2n + 1$. Each vector is composed of two main blocks. The first block contains one bit for each pair of nodes. Such a bit is 1 if the vector encodes an edge between the corresponding (unordered) pair of nodes. The second block contains $2n$ more bits – two consecutive bits for each node in G. If the edge is coming from v_i, then the two bits corresponding to v_i are `01`, and `10` if it is going to v_i. Finally, a `0` is always the last bit. This reduction can easily be implemented by a logarithmic space machine. Intuitively, the second block ensures that we obtain a set of disjoint cycles that cover all nodes, and the first block ensures that every cycle contains at least three nodes.

The MINIMUM TYPE SELECTION instance is now fully defined by $(\mathcal{S}_G, n+1)$. Let G belong to PARTITION INTO HAMILTONIAN SUBGRAPHS. In this case, there is a set of disjoint cycles with at least three nodes each that cover all nodes. For every node v_i, there is an edge of a cycle that leaves v_i. For each v_i, pick the vector encoding this edge from S_i. Now, we count the distinct column vectors in the emerging matrix. Since each node is left by exactly one edge and reached by exactly one edge of the cycles, each column vector of the second block contains exactly one `1`. Hence, there are exactly n distinct column vectors in the second block. Moreover, for each pair v_i, v_j of nodes, at most one of the two edges (v_i, v_j) and (v_j, v_i) is used. This implies, that each column vector in the first block contains at most one `1` as well. Finally, the last column vector is always the `0`-vector leading to $n+1$ distinct column vectors altogether. Concluding, $(\mathcal{S}_G, n+1) \in$ MINIMUM TYPE SELECTION.

Now, assume $(\mathcal{S}_G, n+1) \in$ MINIMUM TYPE SELECTION. Note, that the only way of choosing n vectors from the sets in $\mathcal{S}$ such that there are no more than $n+1$ unique column vectors is by choosing them in such a way that each column in the second block contains exactly one `1`. This is due to the fact that each vector from S_i contains `01` at the according spot for v_i. Hence, the minimum of distinct column vectors in the second block is n. With the additional `0`-vector we have a total of $n+1$ distinct column vectors. If vectors are chosen such that a column vector contains two 1s, then the number of distinct column vectors is at least $n+2$. Hence, we are able to choose edges such that each node is left and reached exactly once each. Moreover, each cycle must contain at least three vertices. If there was a cycle containing only two nodes, then there would be a column in the first block containing two 1s which again would generate too many distinct columns. □

Refer to the following Example 2 to see how the reduction works with a concrete input.

▷ EXAMPLE 2: The reduction function described in the proof of Theorem 21 transforms the PARTITION INTO HAMILTONIAN SUBGRAPHS instance



to the MINIMUM TYPE SELECTION instance $(\mathcal{S} = \{S_1, S_2, S_3\}, 4)$ with

$S_1 = \{$`1000110000`$\}$, $S_2 = \{$`1001001000`, `0010001100`$\}$, $S_3 = \{$`0010010010`, `0101000010`$\}$.

The gray edge is represented by the gray bit vector. Without the gray edge, the graph cannot be partitioned into Hamiltonian subgraphs. In this case, the set S_3 contains just one element, and there are only the two possible matrices

$$M_1 = \begin{pmatrix} \texttt{1000110000} \\ \texttt{1001001000} \\ \texttt{0010010010} \end{pmatrix} \quad \text{and} \quad M_2 = \begin{pmatrix} \texttt{1000110000} \\ \texttt{0010001100} \\ \texttt{0010010010} \end{pmatrix}.$$

Both these matrices clearly have more than 4 distinct columns. If the gray edge does exist, then a possible choice of vectors allows the matrix

$$M_3 = \begin{pmatrix} \texttt{1000110000} \\ \texttt{0010001100} \\ \texttt{0101000010} \end{pmatrix}.$$

Each column in M_3 contains one 1. The matrix thus contains exactly 4 distinct columns.

We will revisit the complexity of the problem MIN-CUT MINIMUM TYPE SELECTION in Chapter 4. We will see similar examples of problems that are encoded in the sets of all min-cuts of a network in Subsection 3.4.2 of this chapter starting on page 49.

3.4 EDGE SET SYSTEMS

This section deals with a quite similar question as the last section: Given a set $\mathcal{E}$, is there a network N such that $\mathcal{E}(N) = \mathcal{E}$? And what properties does $\mathcal{E}(N)$ have? Recall that $\mathcal{E}(N)$ is the set of all B edge set systems

$$\mathcal{E}_B(N) = \left\{ E(S,T) \mid (S,T) \text{ is a B-min-cut in } N \right\}.$$

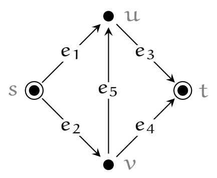
Moreover, we will apply the same kind of encoding problems in the min-cuts of networks as we have seen it at the end of the last section. Interestingly, this gives rise to a framework in which we analyze multiple problems related to flow network and their min-cuts.

When considering an edge set system $\mathcal{E}$ and the question whether a network exists with $\mathcal{E}(N) = \mathcal{E}$, we make the assumption that no pair of edges belonging to $\mathcal{E}$ shares a common node. This way, no information about the network N is revealed. This is equivalent to assigning unique labels to the edges and then defining the edge cut set of a min-cut to contain the labels of the corresponding edges.

For the node set systems, we noted in Corollary 18 that in the 2-terminal case, it does not matter whether we consider directed or undirected networks – the sets of all node set systems are equivalent. This, however, is not the case for edge set systems as there is an edge set system for which there is a directed network but no undirected network.

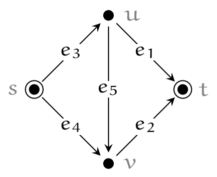
▷ LEMMA 22: The set of all edge set systems for undirected networks is a true subset of the set of all edge set systems for directed networks.

Proof. Clearly, the set of all edge set systems for undirected networks is a subset of all edge set systems for directed networks. It remains to be shown that there is an edge set system $\mathcal{E}$, such that $\mathcal{E}(N) = \mathcal{E}$ for some directed network N, but for all undirected networks N', we have $\mathcal{E}(N') \neq \mathcal{E}$. Let $\mathcal{E} = \{\{e_1, e_2\}, \{e_2, e_3\}, \{e_3, e_4\}\}$, and let N be the network



with unit capacity on every edge. We easily verify that $\mathcal{E}(N) = \mathcal{E}$.

The set of all networks N with $\mathcal{E}(N) = \mathcal{E}$ can be split into two classes – those whose max-flow DAG is the above network N and those whose max-flow DAG is the following network N':



However, there exists no undirected network N'' with $\mathcal{E}(N'') = \mathcal{E}$ such that the $N''_d = N$ or $N''_d = N'$. This implies that there is no undirected network for the given edge set system $\mathcal{E}$. □

3.4.1 *The 2-Terminal Case*

The task to construct an s-t network for a given edge set system $\mathcal{E}$ is already surprisingly challenging. We describe necessary conditions that an edge set system $\mathcal{E}$ must fulfill in order to allow a network N with $\mathcal{E}(N) = \mathcal{E}$. One quite obvious one is that an edge set system cannot contain two sets such that one is a true subset of the other, as this violates the requirement of the cut to be a min-cut. In the following, we will generalize this observation. For that, we need the symmetric difference $A\Delta B$ of two sets, i. e., $A\Delta B = (A - B) \cup (B - A)$. We say that an edge set system $\mathcal{E}$ has the *Δ-property* if there are three sets $e_1, e_2, e_3 \in \mathcal{E}$ with $e_1 \Delta e_2 \subseteq e_3$. Note that if $e_1 \subseteq e_2$, then $e_1 \Delta e_2 \subseteq e_2$. Hence, the Δ-property generalizes the subset relation in the appropriate sense.

▷ LEMMA 23: Let $\mathcal{E}$ be any set system. If $\mathcal{E}$ has the Δ-property, then there is no network N with $\mathcal{E}(N) = \mathcal{E}$.

Proof. Let us assume that there is a network N with $\mathcal{E}(N) = \mathcal{E}$, and that there are sets $e_1, e_2, e_3 \in \mathcal{E}$ with $e_1 \Delta e_2 \subseteq e_3$. If one of these sets is a subset of another, then clearly there is no corresponding network. If e_1, e_2 are disjoint, then $e_1 \Delta e_2 = e_1 \cup e_2$, and thus both e_1 and e_2 are subsets of e_3. Hence, e_1, e_2 are not disjoint and not subsets of each other.

We know the max-flow DAG in combination with μ stores the same information about $\mathcal{E}$ as N. Knowing this, we now switch to the max-flow DAG N_d and exploit the fact that every min-cut cuts every s-t path in N_d exactly once. The edges in $e_1 \cap e_2$ can be augmented to become a min-cut by either adding the edges of $e_1 - e_2$ or of $e_2 - e_1$ to it. This means that all s-t paths not cut by $e_1 \cap e_2$ are cut by the edges of $e_1 - e_2$ and are also cut by the edges of $e_2 - e_1$. This implies, that e_3 cuts at least one s-t path twice and hence cannot be a min-cut. □

From a node set system perspective, the source sides of two min-cuts either contain each other or they are crossing min-cuts. This can easily be detected if the min-cuts are represented as node sets. In the case of edge sets, this is not so easy. In fact, it is impossible to determine whether two min-cuts are crossing min-cuts. This even holds for three min-cuts. To any pair of crossing min-cuts belong four min-cuts. Lemma 24 allows us to detect whether four given edge sets belong to four min-cuts belonging to a pair of crossing min-cuts.

A possible approach is to keep thinking in terms of node set systems even though an edge set system is given. This edge set systems has to consist of edges that connect nodes in a way that we understand and which has been described in the previous section.

If two min-cuts S_1, S_2 are crossing min-cuts, i.e., $S_1 - S_2 \neq \emptyset$ and $S_2 - S_1 \neq \emptyset$, then they induce two more min-cuts $S_1 \cap S_2$ and $S_1 \cup S_2$ by the submodularity property. The resulting four min-cuts associated with such a crossing can be identified from an edge set system as described in the following lemma.

▷ LEMMA 24 (CROSSING MIN-CUT EDGE SETS): Four min-cut edge sets E_0, E_1, E_2, E_3 with $I = E_0 \cap E_1 \cap E_2 \cap E_3$ correspond to the four min-cuts induced by two crossing min-cuts if, and only if,

- the intersection graph of $\{E_0 - I, E_1 - I, E_2 - I, E_3 - I\}$ is the C_4 and
- no edge set contains an element that is not contained in any other edge set.

Proof. Suppose we have two crossing min-cuts $S \cup X_1$ and $S \cup X_2$ with disjoint X_1, X_2 and let R be $V - S - X_1 - X_2$. By the submodularity, we also have the min-cuts S and $S \cup X_1 \cup X_2$. By Lemma 20, there are no edges connecting X_1 and X_2. These four min-cuts represented by the cut edges are

$$\begin{aligned} E(S) &= E(S, X_1) \cup E(S, X_2) \cup E(S, R)\,, && (54)\\ E(S \cup X_1) &= E(S, X_2) \cup E(X_1, R) \cup E(S, R)\,, && (55)\\ E(S \cup X_2) &= E(S, X_1) \cup E(X_2, R) \cup E(S, R)\,, \text{ and} && (56)\\ E(S \cup X_1 \cup X_2) &= E(X_1, R) \cup E(X_2, R) \cup E(S, R)\,. && (57) \end{aligned}$$

With $I = E(S, R)$ it can easily be verified, that the intersection graph of $\{E_0 - I, E_1 - I, E_2 - I, E_3 - I\}$ is the C_4. Moreover, no edge set contains any edges that are not contained in any of the other three sets.

Conversely, suppose we are given a set of four edge sets E_0,E_1,E_2,E_3 with $I = E_0 \cap E_1 \cap E_2 \cap E_3$ such that no edge set contains an edge that is not contained in any other edge set and such that the intersection graph of $\{E_0 - I, E_1 - I, E_2 - I, E_3 - I\}$ is the C_4. Without loss of generality, we assume that $E_0 \cap E_2 - I = \emptyset$ and $E_1 \cap E_3 - I = \emptyset$. Note, that for such a set of sets, we have $E_0 - E_1 = E_3 - E_2$, $E_0 - E_3 = E_1 - E_2$, $E_2 - E_1 = E_3 - E_0$, and $E_2 - E_3 = E_1 - E_0$. We will now construct a network based on a multi-graph consisting of the nodes $V = \{s, x_1, x_2, t\}$ and the edges $E = E_0 \cup E_1 \cup E_2 \cup E_3$. For this, set

$$\begin{aligned} (s, t) &= I\,, && (58)\\ (s, x_1) &= E_0 - E_3\,, && (59)\\ (s, x_2) &= E_0 - E_1\,, && (60)\\ (x_1, t) &= E_2 - E_1\,, \text{ and} && (61)\\ (x_2, t) &= E_2 - E_3\,. && (62) \end{aligned}$$

This multi-graph can be turned into a graph by expanding each node to a clique of appropriate size and infinitely high capacities.

Now, the capacity of the edges in E_i is set to $1/|E_i|$. The capacity of the network hence is 2. The set of all min-cuts is $\{s\}, \{s, x_1\}, \{s, x_2\}, \{s, x_1, x_2\}$, and the according edge sets are

$$
\begin{aligned}
E(\{s\}) &= I \cup (E_0 - E_3) \cup (E_0 - E_1) = E_0\,, && (63)\\
E(\{s, x_1\}) &= I \cup (E_0 - E_1) \cup (E_2 - E_1) && (64)\\
&= I \cup (E_3 - E_2) \cup (E_3 - E_0) = E_3\,, && (65)\\
E(\{s, x_2\}) &= I \cup (E_0 - E_3) \cup (E_2 - E_3) && (66)\\
&= I \cup (E_1 - E_2) \cup (E_1 - E_0) = E_1\,, \text{ and} && (67)\\
E(\{s, x_1, x_2\}) &= I \cup (E_2 - E_1) \cup (E_2 - E_3) = E_2\,. && (68)
\end{aligned}
$$

□

Even if four min-cuts are identified that belong to a pair of crossing min-cuts, it is not clear which two of these four min-cuts are the crossing ones. There are two possible pairs for this and these are the two sets whose intersection is I.

For any three min-cuts E_1, E_2, E_3 corresponding to min-cuts whose source sides S_1, S_2, S_3 satisfy $S_1 \subset S_2 \subset S_3$, we know that the min-cut E_2 in between the other two has to cut every edge that E_1 as well as E_3 cut.

▷ LEMMA 25: Let N be a network with three min-cuts S_1, S_2, S_3 such that $S_1 \subset S_2 \subset S_3$. Then $E_1 \cap E_3 \subset E_2$.

Proof. Suppose $e \in S_1 \cap S_3$. This implies that one endpoint of e is in S_1 and the other endpoint of e is in S_3. Hence, $e \in S_2$ as well. Moreover, if $E_1 \cap E_3 = E_2$, then either $E_1 = E_3$ (which is a contradiction to the assumption) or E_2 is a true subset of E_1 and E_3 which contradicts the minimality of E_1 and E_3. □

This, however, does not imply the other direction. If the given edge set system $\mathcal{E}$ allows a network N with $\mathcal{E}(N) = \mathcal{E}$ and if it contains three sets E_1, E_2, E_3 with $E_1 \cap E_3 \subset E_2$, then these three sets do not necessarily correspond to three min-cuts S_1, S_2, S_3 with $S_1 \subset S_2 \subset S_3$.

Even though these considerations may lead to an efficient algorithm to construct a network from a given edge set system, this will not give us a short characterization as we obtained it for node set systems.

3.4.2 *Problems Encoded in Edge Set Systems*

As we are unable to fully settle the question of when a set system $\mathcal{E}$ allows a network N with $\mathcal{E}(N) = \mathcal{E}$, we further explore these set systems using new ideas, but keep focusing on 2-terminal networks. We consider new variants of well-known set problems by encoding them in flow networks. In particular, set problems whose instances contain a set or a set system $\mathcal{S}$ of subsets of a universe U are encoded by a directed s-t flow network N such that the universe is the set of all *min-cut edges*, i. e., edges that are cut by some min-cut, and $\mathcal{S}$ is $\mathcal{E}(N)$. The obtained new problem remains its name with a preceding MIN-CUT. HITTING SET, for example, is turned into MIN-CUT HITTING SET this way.

Clearly, not every set system can be encoded by a flow network due to the restrictions imposed by min-cuts. This makes the problem potentially easier. On the other hand, a network may have up to exponentially many min-cuts in the number of nodes. The problem of determining the number of min-cuts in a network is #P-complete [71]. This makes the problem potentially harder. In the following we will analyze which effect dominates.

This research could also be done by encoding the problem in the min-cuts of graphs instead of flow networks. However, the number of min-cuts in a graph with n nodes is at most quadratic in n [21] and the set of all min-cuts is easily computed in P. A possible approach to achieve this is by computing for every pair of nodes s, t the s-t min-cut capacity. Then, we list all min-cuts for those pairs with the minimum min-cut capacity. As these can be at most polynomial many, the total runtime is polynomial. Encoding set systems in graph min-cuts thus puts severe restrictions on the sets but does not provide any kind of succinctness. Hence, we will only consider encoding sets and set systems in flow networks.

For some problems, like PARTITION or SUBSET SUM, whose input does not contain set systems, but simply a set, we do not observe any of the two described effects if encoded in networks. Such a set cannot be encoded succinctly. Moreover, there are no restrictions – any set can be encoded in a flow network. Hence, these problems remain NP-complete. We demonstrate this with the following three problems.

▷ PROBLEM 4 (MIN-CUT PARTITION):

Input: A network N.

Question: Is there a subset E' of the min-cut edges such that the sum of their capacities equals the sum of the capacities of the min-cut edges not in E'?

▷ PROBLEM 5 (MIN-CUT SUBSET SUM):

Input: A network N and a number k.

Question: Is there a set of min-cut edges whose sum of capacities is k?

▷ PROBLEM 6 (MIN-CUT SUBSET PRODUCT):

Input: A network N and a number k.

Question: Is there a set of min-cut edges whose product of capacities is k?

▷ THEOREM 26: The problems

- MIN-CUT PARTITION,
- MIN-CUT SUBSET SUM, and
- MIN-CUT SUBSET PRODUCT

are NP-complete.

Proof. For membership, a short solution is easily guessed and verified for each of the problems. For each problem, let $A = \{a_1, a_2, \dots\}$ be the set and $w : A \to \mathbb{N}$ the weight function of the input. An instance of the original version can be reduced to the min-cut variant by creating a network with source s, sink t, nodes $s_1, s_2, \dots, s_{|A|}$, and $t_1, t_2, \dots, t_{|A|}$. The source is connected to each s_i and each t_i is connected to the sink by an edge of large capacity to ensure that none of these edges are min-cut edges. Moreover, each s_i is connected to t_i by an edge of capacity $w(a_i)$. This can be done using only logarithmic space.

This network has a unique min-cut that contains as many edges as there are elements in the input of the instance. Clearly, this instance is a member

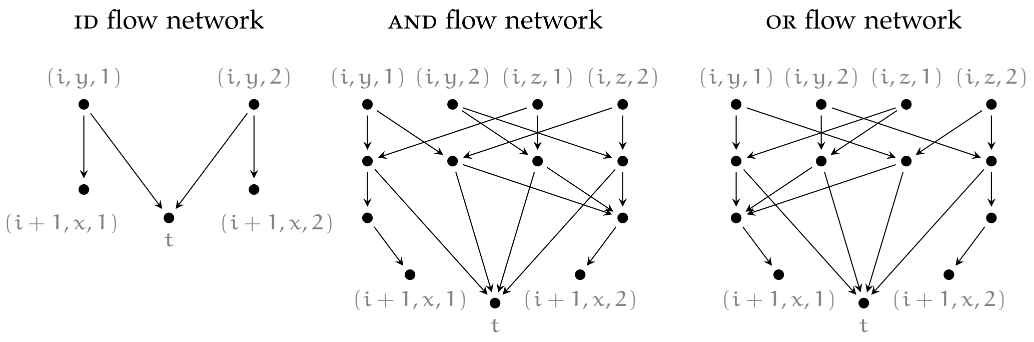


Figure 10: As part of the reduction from MLCVE2 to BANF, each edge from gate (i, y) to gate $(i + 1, x)$ is replaced by one of the depicted flow networks depending on the type of $(i + 1, x)$ where each edge has capacity 4^{k-i}.

of the min-cut variant if, and only if, the original instance is a member of the original problem. □

As already mentioned, min-cut variants are more interesting, when the input of the original problem contains a set system rather than just a set. Those problems change their complexity and do not stay NP-complete. Whether they are easier or harder than before depends on whether the min-cut variant involves counting the number of min-cuts. If it does not, the complexity is P-completeness, otherwise it is the complexity of deciding whether a given network N contains at least k min-cuts, where k is part of the input as well.

The following is a list of NP-complete problems whose instances contain a set system and are listed by Garey and Johnson [34]. Many of these problems belong to the 21 NP-complete problems listed by Karp [50]. We will define their min-cut variants and analyze the complexity of the new problems.

- MINIMUM COVER,
- HITTING SET,
- SET PACKING,
- SET SPLITTING,
- TEST SET, and
- SET BASIS.

In 1982, Goldschlager, Shaw, and Staples [35] proved that deciding if the max-flow value of an s-t network is odd is P-complete. In 1990, a result by Lengauer and Wagner [58] followed showing that deciding whether the max-flow value of a network is at least k is P-complete as well. We will often use this to show P-completeness of some MIN-CUT variants. Formally, Lengauer and Wagner [58] define the problem as

▷ PROBLEM 7 (BINARY ACYCLIC NETWORK FLOW (BANF)):
Input: A network $N = (V, E, \{s, t\}, c)$, where $G = (V, E)$ is a DAG, and a natural number m.
Question: Is the value of a max-flow in N at least m?

The completeness proof, that will be presented shortly, makes use of *circuits*. A circuit is a tuple $C = (V, E, v_0, \mathrm{in})$ where (V, E) is a DAG, $v_0 \in V$, and in a function. The elements of V are referred to as *gates*. In- and out-degree of a gate are called *fan-in* and *fan-out*. Every gate with fan-in 0 is an *input gate* and every gate with fan-out 0 is an *output gate*. The function in maps every input gate to $\{0, 1\}$. Typically, there are the four gate types AND, OR, NOT, and ID with indegrees 2, 2, 1, 1, respectively. These gates compute the conjunction, disjunction, negation, and identity of their inputs. Gate $v_0 \in V$ is a particular output gate. The classical CIRCUIT VALUE PROBLEM (CVP) asks for the output value of v_0 for a given circuit and is P-complete [56]. For a more detailed introduction to circuits, refer to the book *Limits to Parallel Computation: P-Completeness Theory* by Greenlaw, Hoover, and Ruzzo [37] or *Introduction to Circuit Complexity: A Uniform Approach* by Vollmer [80].

The problem MLCVE2 is equal to CVP restricted to circuits that are monotone (the NOT gate is forbidden), each gate has fan-out either 0 or 2, and (V, E) is layered meaning that V can be partitioned into *layers* $\{0, 1, \dots, k\}$ such that for each edge (u, v), u belongs to layer $0 \leqslant i < k$ if, and only if, v belongs to layer $i + 1$. Despite these restrictions MLCVE2 is P-complete as well [58]. The proof of this result does not make any use of cycles making it possible to restrict the input to directed acyclic networks. We present this proof as we will later adjust it for our needs. The unrestricted version of this problem allowing cycles is P-complete, too [37, 58].

▷ THEOREM 27 (FROM [58]): BANF is P-complete.

Proof. The problem is in P, as a max-flow can be computed in polynomial time. To show hardness, a log-space reduction from MLCVE2 is given. The nodes V of the circuit C are made up of nodes (i, x_j) with $0 \leqslant i \leqslant k$ and $j \geqslant 0$, where (i, x_j) is the j^{th} node on the i^{th} layer. Let $C(i, x_j)$ be the value of the according gate (recall that the input of the circuit is fixed by the function in). Now, a flow network $N = (V', E', \{s, t\}, c)$ is constructed from C as follows:

1. A source s and a sink t are added, and every gate (i, x) in C is replaced with the two nodes $(i, x, 1)$ and $(i, x, 2)$.
2. If $\mathrm{in}(0, x) = 1$, then an edge is added from s to $(0, x, 1)$ with capacity 4^k. Otherwise, there is an edge from s to $(0, x, 2)$ with capacity 4^k.
3. Each edge from gate (i, y) to gate $(i + 1, x)$ is replaced by a small flow network. Depending on whether $(i + 1, x)$ is an ID, OR, or AND gate, the edge is replaced by the ID, OR, or AND flow network, respectively, depicted in Figure 10.
4. There is an edge of capacity 1 from all nodes $(k, x, 1)$ and $(k, x, 2)$ to t.

Let D be the number of input gates in C. Lengauer and Wagner [58] show that for every flow f in N we have $|f(s)| = f(t) \leqslant D\,4^k$. Moreover, for every max-flow f_m in N the flow reaching $(i, x, 1)$ and $(i, x, 2)$ with respect to f_m is

$$f_{m,\mathrm{in}}(i, x, 1) = 4^{k-i} \text{ if } C(i, x) = 1 \text{ and} \tag{69}$$

$$f_{m,\mathrm{in}}(i, x, 2) = 4^{k-i} \text{ if } C(i, x) = 0\,. \tag{70}$$

We exploit this fact and link the value of the max-flow to the value of the circuit by deleting the edge from $(k, x_0, 2)$ to t in N. This way, we have

$$c(N') = \begin{cases} D\,4^k & \text{if } C(k, x_0) = 1 \text{ and} \\ D\,4^k - 1 & \text{otherwise}. \end{cases} \tag{71}$$

Consequently, $C \in$ MLCVE2 if, and only if, $(N', D\,4^k) \in$ BANF. □

We will use the idea of the just presented proof to demonstrate the P-completeness of some of our min-cut variants. For this, we define the network N'_d to be the max-flow DAG N_d of N with a few alterations. We introduce a distance function $w(e) = 1$ for each edge in N_d. Then, we add the edge (v, u) for every edge (u, v) and set $w(v, u) = 0$. The following lemma might seem a little unnatural at the moment, but will prove very helpful soon.

▷ LEMMA 28: Given a network N and a number k, the following two decision problems are P-complete:

- Is the length of the longest s-t path in N_d at least k?
- Is the length of the shortest s-t path in N'_d with respect to w at most k?

Proof. The problems are in P as we can find the max-flow DAG N_d and compute the longest path in it in polynomial time, and we can compute N'_d and find the shortest path in it in polynomial time.

For hardness, we use a log-space reduction similar to the one used in the proof of Theorem 27. We utilize the fact that the value of a max-flow is linked to the value of the circuit and attach a new path from t to the new sink t' of length $3k + 3$ and width $D\,4^k$. This can easily be done using logarithmic space.

If $C(k, x_0) = 1$, then the value of a maximum s-t' flow is $D\,4^k$ and the t-t' path is saturated by every max-flow. By Lemma 9, the length of the shortest s-t' path in N_d is thus quite long – in particular longer than $3k + 2$. If $C(k, x_0) = 0$, then the network's capacity is $D\,k^4 - 1$ and the t-t' path is contracted to obtain N_d. Now, the shortest s-t' path in N_d is at most $3k + 2$. These considerations remain valid even if the length of a path is measured with respect to the described distance function w.

Similar considerations hold for the length of the longest path in N_d. The longest path is at least of length $3k + 3$ if, and only if, $C(k, x_0) = 1$. □

In the following, we take a look at some basic problems related to networks, min-cuts, and max-flows to gain some intuition for the hardness of such problems, before finally turning our attention towards the complexity of the new min-cut variants of set problems.

▷ PROBLEM 8 (MIN-CUT):

Input: A flow network N and a natural number k.
Question: Is there an s-t cut in N of capacity at most k?

As BANF is P-complete, it is no big surprise that MIN-CUT is P-complete as well due to the Max-Flow Min-Cut Theorem (Theorem 1, page 20). While checking whether a given set of edges defines a min-cut is P-complete, checking whether a given flow is a max-flow is NL-complete. This has also been noted by Papadimitriou [67].

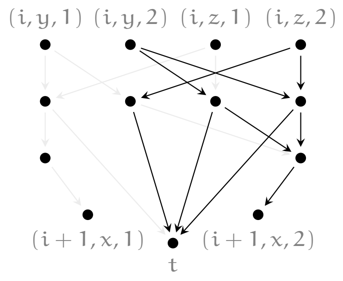
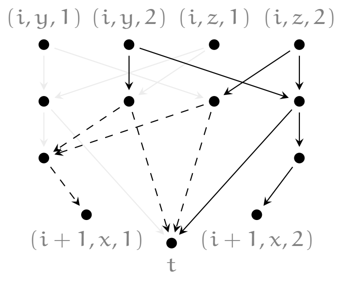
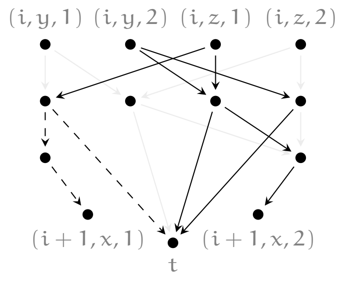
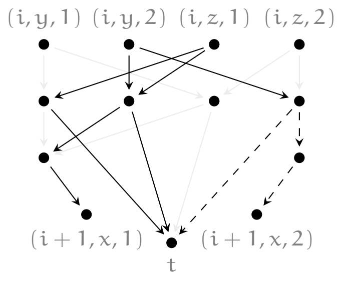
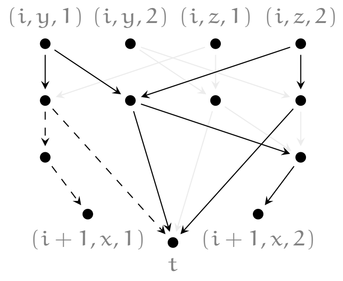
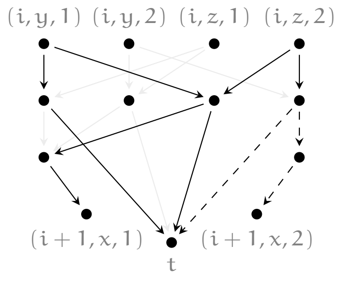
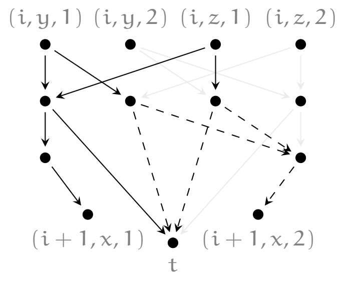
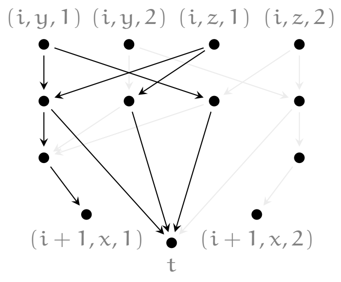
Gate input	Max-flow DAG of AND network	Max-flow DAG of OR network
0,0	 $(i,y,1)$ $(i,y,2)$ $(i,z,1)$ $(i,z,2)$ $(i+1,x,1)$ $(i+1,x,2)$ t	 $(i,y,1)$ $(i,y,2)$ $(i,z,1)$ $(i,z,2)$ $(i+1,x,1)$ $(i+1,x,2)$ t
0,1	 $(i,y,1)$ $(i,y,2)$ $(i,z,1)$ $(i,z,2)$ $(i+1,x,1)$ $(i+1,x,2)$ t	 $(i,y,1)$ $(i,y,2)$ $(i,z,1)$ $(i,z,2)$ $(i+1,x,1)$ $(i+1,x,2)$ t
1,0	 $(i,y,1)$ $(i,y,2)$ $(i,z,1)$ $(i,z,2)$ $(i+1,x,1)$ $(i+1,x,2)$ t	 $(i,y,1)$ $(i,y,2)$ $(i,z,1)$ $(i,z,2)$ $(i+1,x,1)$ $(i+1,x,2)$ t
1,1	 $(i,y,1)$ $(i,y,2)$ $(i,z,1)$ $(i,z,2)$ $(i+1,x,1)$ $(i+1,x,2)$ t	 $(i,y,1)$ $(i,y,2)$ $(i,z,1)$ $(i,z,2)$ $(i+1,x,1)$ $(i+1,x,2)$ t

Figure 11: Depending on the input to each of the local replacement networks of Figure 10, every max-flow saturates the solid edges, no max-flow saturates the grayed out edges, and some max-flows saturate the dashed edges, while other max-flows do not saturate the dashed edges. In the not depicted ID network, either the edges leaving $(i, y, 1)$ or the edges leaving $(i, y, 2)$ are saturated, depending on the input. By Lemma 9, the according max-flow DAGs consist of the solid edges, while dashed and grayed out edges are contracted.

▷ PROBLEM 9 (MIN-CUT EDGE SET):

Input: A flow network N and a set of edges E'.
Question: Is E' a min-cut in N?

▷ LEMMA 29: MIN-CUT EDGE SET is P-complete.

Proof. For membership, check if E' is a cut by deleting the edges E' in N and then checking if there is an s-t path. If there is none, check if the capacity of E' is equal to the min-cut capacity.

For hardness, we reduce BANF to MIN-CUT EDGE SET via a log-space function. The original instance (N, k) is altered to obtain network N' by adding a new sink t' and a new edge from the original sink t to t' of capacity k. Now $\big(N', \{(t, t')\}\big) \in$ MIN-CUT EDGE SET if, and only if, there is a flow of value at least k in N. □

▷ PROBLEM 10 (MAX-FLOW):

Input: A flow network N and a function $f : V \times V \to \mathbb{R}$.
Question: Does f describe a max-flow in N?

▷ LEMMA 30: MAX-FLOW is NL-complete.

Proof. To verify that f is a flow, we check for every edge whether the skew symmetry (2) holds and whether its flow is at most its capacity. Next, we need to check for every node except for source and sink if the flow conservation is fulfilled. Since ITERATED ADDITION is in TC^0 [80], the problem stays in NL. If f is a flow but not a max-flow, then there is an augmenting path in the residual network N_f of width at least 1. We non-deterministically guess successively the next node of this path and adjust the flow value accordingly, checking the flow constraints as we go. If there is such a path, then f is not a max-flow and hence we reject, otherwise we accept, which can be done due to $\mathsf{coNL} = \mathsf{NL}$ [45, 78].

For hardness, we alter a REACHABILITY instance (G, s, t) via a log-space machine by declaring G as a flow network N with unit capacity on all edges and with source s and sink t. Now $(N, f_0) \in$ MAX-FLOW where f_0 assigns a zero to every edge if, and only if, t is not reachable from s. Since $\mathsf{coNL} = \mathsf{NL}$, we are done. □

We now define a first set of min-cut variants following the pattern discussed earlier.

▷ DEFINITION 6: For a given s-t network N,

- a *min-cut hitting set* $H \subseteq E$ for N is a set of edges such that for all $E' \in \mathcal{E}(N)$ we have $H \cap E' \neq \emptyset$,
- a *min-cut set packing* $P \subseteq \mathcal{E}(N)$ for N is a set of pairwise disjoint min-cuts,
- a *min-cut cover* $C \subseteq \mathcal{E}(N)$ for N is a set of min-cuts such that every min-cut edge is cut by some min-cut in C, and
- a *min-cut edge packing* for N is a set of min-cut edges, such that no two edges of this packing are cut by the same min-cut.

▷ PROBLEM 11 (MIN-CUT HITTING SET):

Input: A directed s-t network N and a natural number k.
Question: Is there a min-cut hitting set H for N of size at most k?

▷ PROBLEM 12 (MIN-CUT SET PACKING):

Input: A directed s-t network N and a natural number k.
Question: Is there a min-cut set packing of size at least k?

▷ PROBLEM 13 (MIN-CUT MINIMUM COVER):

Input: A directed s-t network N and a natural number k.
Question: Is there a min-cut cover of size at most k?

▷ PROBLEM 14 (MIN-CUT EDGE PACKING):

Input: A directed s-t network N and a natural number k.
Question: Is there a min-cut edge packing of size at least k?

Describing some of these problems as linear programs reveals some interesting properties. Recall the cover-packing dualities mentioned on page 26. There are several well-known cover-packing dualities. One of them is the duality between HITTING SET and SET PACKING. HITTING SET can be viewed as a kind of covering problem where each set S in the given set system $\mathcal{S}$ is covered if, and only if, an element $e \in U$ is picked with $e \in S$. The corresponding ILP with variables $x_e \in \{0,1\}$ for each $e \in U$ can be formulated as ILP (72). Finding the dual to its LP relaxation and restricting it to integer solutions yields ILP (73).

HITTING SET: (72)

$$\begin{aligned} \min \quad & \sum_{e \in U} x_e \\ \text{s.t.} \quad & \sum_{e \in S} x_e \geqslant 1, \quad \forall S \in \mathcal{S} \\ & x_e \in \{0,1\}, \quad \forall e \in U\,. \end{aligned} \tag{72}$$

SET PACKING: (73)

$$\begin{aligned} \max \quad & \sum_{S \in \mathcal{S}} x_S \\ \text{s.t.} \quad & \sum_{S:\, e \in S} x_S \leqslant 1, \quad \forall e \in U \\ & x_S \in \{0,1\}, \quad \forall S \in \mathcal{S}\,. \end{aligned} \tag{73}$$

Clearly, ILP (72) is a covering ILP and ILP (73) is a packing ILP. As HITTING SET and SET PACKING are NP-complete, an optimal solution to ILP (72) does not always have the same value as an optimal solution to ILP (73). If the LP relaxations always had the same optimum value, then $\mathrm{P} = \mathrm{NP}$.

Very similar observations can be made for another pair. Vazirani [79] describes a duality between SET COVER and a certain packing problem. To formulate SET COVER as an ILP, we need for each set S of the given set system $\mathcal{S}$ a variable x_S which is allowed to be either 0 or 1. Variable x_S is supposed to be 1 if, and only if, set S is picked in the cover. The constraints need to make sure that for each element e of the universe U, there is at least one picked set that contains it. The corresponding ILP thus is ILP (74). Again, we identify the dual to its LP relaxation and restrict it to integer solutions to obtain ILP (75).

SET COVER: (74)

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{S}} x_S \\ \text{s.t.} \quad & \sum_{S:\, e \in S} x_S \geqslant 1, \quad \forall e \in U \\ & x_s \in \{0,1\}, \quad \forall S \in \mathcal{S}\,. \end{aligned} \tag{74}$$

ELEMENT PACKING: (75)

$$\begin{aligned} \max \quad & \sum_{e \in U} y_e \\ \text{s.t.} \quad & \sum_{e \in S} y_e \leqslant 1, \quad \forall S \in \mathcal{S} \\ & y_e \in \{0,1\}, \quad \forall e \in U\,. \end{aligned} \tag{75}$$

A solution to the ILP (75) can be interpreted as a packing of elements $e \in U$. Two elements may be in a packing if, and only if, there is no set that

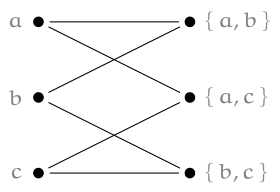


Figure 12: Consider the universe $U = \{a, b, c\}$ and the set system $\mathcal{S} = \{\{a, b\}, \{a, c\}, \{b, c\}\}$. In the above graph the elements of U are on the left, and the sets in $\mathcal{S}$ on the right. There is an edge between an element $e \in U$ and a set $S \in \mathcal{S}$ if, and only if, $e \in S$. For this instance, all smallest hitting sets are of size 2 (for example $\{a, b\}$), while all largest set packings are of size 1. Similarly, the largest element packing may pack 1 element, while the smallest set cover needs 2 sets. Hence, even though the LP-relaxations are dual to each other, the optimal integer solutions do not have the same value. This is consistent to the fact that the considered problems are all NP-complete. The min-cut variants of these problems, however, are P-complete, and the optimal values of the corresponding ILPs are equal.

contains both of them. Considering the min-cut variant of this problem, it translates to finding as many min-cut edges as possible such that no two of them are cut by the same min-cut. This is MIN-CUT EDGE PACKING.

The above formulated ILPs of the same pair do not always have the same value. For an example, see Figure 12. We will shortly see, that the min-cut variants of the just described problems restrict the instances to those where the optimal values of the corresponding ILPs are equal. This is consistent with the fact that these min-cut variants are P-complete as we will see next.

▷ LEMMA 31 (REPRESENTATION LEMMA): Let N be any network and N_d its max-flow DAG. Then,

1. $\mathcal{E}' \subseteq \mathcal{E}(N_d)$ is a maximum min-cut set packing (a minimum min-cut cover) for N_d if, and only if, $\mathcal{E}'' = \{\mu(E') \mid E' \in \mathcal{E}'\}$ is a maximum min-cut set packing (a minimum min-cut cover) for N, and
2. $E' \subseteq E_d$ is a minimum min-cut hitting set (a maximum min-cut element packing) for N_d if, and only if, it also is for N.

Proof. Let $E_1, E_2 \in \mathcal{E}(N_d)$ be two distinct min-cuts in N_d. There is an edge $(u, v) \in E_1 \cap E_2$ such that u is on the source side of both cuts and v is on the sink side of both cuts if, and only if, u is on the source side and v is on the sink side of $\mu(E_1)$ and of $\mu(E_2)$. Hence, E_1 and E_2 are disjoint if, and only if, $\mu(E_1)$ and $\mu(E_2)$ are. This implies that $\mathcal{E}'$ is a min-cut set packing for N_d if, and only if, $\mathcal{E}''$ is a min-cut set packing for N. Due to the bijection between $\mathcal{E}(N)$ and $\mathcal{E}(N_d)$ shown in Lemma 14 on page 37, we know that N and N_e contain the same number of min-cuts. The claim follows.

Concerning the min-cut covers, we assert that a set of min-cuts $\mathcal{E}' \subseteq \mathcal{E}(N_d)$ covers the edge set E' in N_d if, and only if, $\mathcal{E}'' = \{\mu(E') \mid E' \in \mathcal{E}'\}$ covers the edges $\{\mu(e) \mid e \in E'\}$ in N. Since μ partitions the edges of N, we see that all min-cut edges are covered in N by $\mathcal{E}''$ if, and only if, all edges are covered in N_d by $\mathcal{E}'$. The first part of the claim thus is proven.

For the second part, we claim that $H \subseteq E_d$ is a min-cut hitting set for N_d if, and only if, it is a min-cut hitting set for N. For any min-cut $E' \in \mathcal{E}(N_d)$ and any edge $(u, v) \in E_d$, we have $(u, v) \in E'$ if, and only if, $(u, v) \in \mu(E')$. Hence, a min-cut E' in N_d is hit by an edge e if, and only if, $\mu(E')$ is hit by e

as well. An edge $e \in E - E_d$ either is not a min-cut edge at all or hits exactly those min-cuts hit by $\mu(e)$.

Finally, two edges $e_1, e_2 \in N_d$ are cut by the same min-cut $E' \subseteq E$ if, and only if, e_1, e_2 are also cut by the same min-cut $\mu(E')$ in N. Again, any edge $e \in E - E_d$ is either not cut by any min-cut or cut by exactly the same min-cuts as some edge $e' \in N_d$. □

This lemma allows us to solve many min-cut variants on the max-flow DAG instead of the original graph. Now we are ready to show the complexity of these min-cut variants.

▷ LEMMA 32: MIN-CUT HITTING SET $\in$ P.

Proof. Due to the Representation Lemma, we may solve the problem using the max-flow DAG N_d, which can be efficiently computed. We obtain a new network N'_d from N_d with distance function w by setting $w(u,v) = 1$, adding (v,u), and setting $w(v,u) = 0$ for each edge (u,v) in N_d. We claim that the size of the smallest min-cut hitting set is equal to the length of a shortest s-t path with respect to w in N'_d. For this, we show that for any shortest s-t path p_s in N'_d with respect to w, $H = \{e \in p_s \mid w(e) = 1\}$ is a smallest min-cut hitting set for N_d. It suffices to show that a set E_h is a min-cut hitting set if, and only if, there is an s-t path p in N'_d with $E_h \supseteq \{e \in p \mid w(e) = 1\}$.

Suppose that $E_h \subseteq E_d$ is a min-cut hitting set. For the sake of contradiction, we assume there is no s-t path p in N'_d with $\{e \in p \mid w(e) = 1\} \subseteq E_h$. Let S be the set of all nodes reachable from s in N'_d via edges in E_h or edges e with $w(e) = 0$. By assumption, $t \notin S$. Moreover, all edges of N_d with exactly one endpoint in S leave S. To see this, suppose (u,v) is an edge with $u \notin S$ and $v \in S$. Then, there is a counterpart (v,u) in N'_d with $w(v,u) = 0$, and hence $v \in S$ by definition – such an edge thus does not exist. By Lemma 16 (page 38), S is a min-cut in N_d, and the corresponding min-cut edge set is not hit by E_h contradicting the fact that E_h is a min-cut hitting set.

Now let $E_h \subseteq E_d$ be a set of edges such that there is an s-t path p in N'_d with $\{e \in p \mid w(e) = 1\} \subseteq E_h$. We will prove that E_h is a min-cut hitting set by induction over $p_0 = \big|\{e \in p \mid w(e) = 0\}\big|$. If $p_0 = 0$, then p is an s-t path in N_d which clearly is cut by every min-cut. We may now assume that the statement is true for an arbitrary but fixed p_0.

Now let $p'_0 = p_0 + 1$ and p an s-t path in N'_d. Any min-cut (S,T) of N_d cuts p as p is an s-t path in N'_d. So assume that (S,T) cuts an edge (u,v) of N'_d with $w(u,v) = 0$. This means that (u,v) does not exist in N_d, but (v,u). As (S,T) is still a min-cut in N_d, we know that $v \in S$ and $u \in T$. This implies that (S,T) cuts every s-u path and every v-t path. These contain at most p_0 edges of weight 0 and hence, we have proven that E_h indeed is a min-cut hitting set. □

▷ LEMMA 33: MIN-CUT SET PACKING $\in$ P.

Proof. We claim that the size of the largest set packing is also equal to the length of a shortest s-t path p_s in N'_d with respect to w as described in the proof of Lemma 32. We first give an algorithm that runs on N_d to find such a min-cut set packing and then show that there is no larger one.

```
1: $S \leftarrow \{s\}$
2: $P \leftarrow \{E(S)\}$
3: WHILE there is no $(u,t)$ with $u \in S$ DO
```

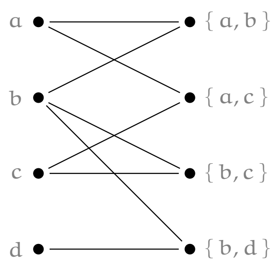


Figure 13: For the system $\mathcal{S} = \big\{\{a,b\},\{a,c\},\{b,c\},\{b,d\}\big\}$, the LPs (72), (73), (74), and (75) have integer optima. Still, there is no network N with $\mathcal{E}(N) = \mathcal{S}$. By Lemma 23, the set system has the Δ-property rendering it impossible to find such a network.

```
 4:     FOR all $(u, v)$ with $u \in S$ and $v \notin S$ DO
 5:         $S \leftarrow S \cup \{v\}$
 6:     END FOR
 7:     WHILE there is an edge $(u, v)$ with $u \notin S$ and $v \in S$ DO
 8:         $S \leftarrow S \cup \{u\}$
 9:     END WHILE
10:     $P \leftarrow P \cup \{E(S)\}$
11: END WHILE
```

This algorithm produces a set P of pairwise disjoint min-cuts. Every edge set in P is a min-cut since the set S from which it is derived contains s but not t, and every edge with exactly one endpoint in S leaves S. This is ensured in lines 7–9. In this loop, t cannot be added to S as there is no outgoing edge from t in N_d. Further, let $E(S_i)$ be the set added to P in the i^{th} iteration where the first set added is S_0. Then S contains exactly those nodes v reachable in N'_d from s via a path of length at most i with respect to w because all nodes added in lines 4–6 have a distance to s increased by 1 compared to the previous step. Moreover, lines 7–9 add all those nodes to S whose distance to S is 0. Therefore, $E(S_i)$ contains exactly those edges (u, v) with $w(u, v) = 1$ to S where v is exactly at a distance of i from s in N'_d with respect to w. Hence, $|P| = |\{\, e \in p_s \mid w(e) = 1 \,\}|$.

To see that there is no greater packing, notice that the min-cuts of any maximum min-cut packing cut all edges in $\{\, e \in p_s \mid w(e) = 1 \,\}$ for any shortest s-t path p_s in N_d. By the last lemma, this is a min-cut hitting set. Since every min-cut cuts an edge of this min-cut hitting set, there cannot be more disjoint min-cuts than |P|. □

▷ THEOREM 34: MIN-CUT HITTING SET and MIN-CUT SET PACKING are P-complete.

Proof. We know from Lemma 32 and Lemma 33 that both problems are in P. Even more, we know that the sizes of the minimum min-cut hitting set and the maximum min-cut set packing are equal to the length of the shortest s-t path in a modified max-flow DAG N'_d. By Lemma 28 we know that deciding whether this is at most k is P-complete. □

▷ COROLLARY 35: MIN-CUT HITTING SET and MIN-CUT SET PACKING are dual to each other.

Figure 14 shows an example of the just discussed problems and their duality. This duality implies that our encoding restricts HITTING SET and SET PACKING to those instances, where the solutions to the ILPs (72) and (73)

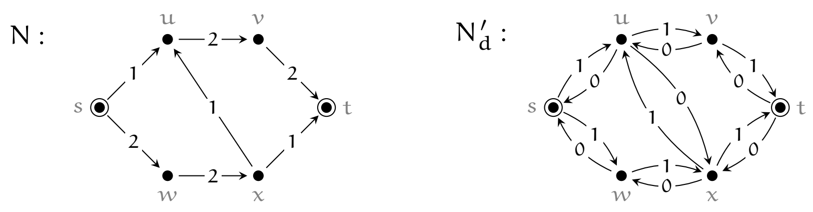


Figure 14: Let N be the depicted network such that $N_d = N$. N'_d is N_d with the weight function w as described in the proof of Lemma 32. The smallest min-cut hitting set is $\{ (s,u), (x,t) \}$, and the largest min-cut set packing is also of size 2. This corresponds to the length of the shortest s-t path in N'_d, which is s-u-x-t. The smallest min-cut cover needs 5 min-cuts as the longest path in N is of length 5. Moreover, the largest min-cut edge packing has also size 5. Note, that a min-cut hitting set needs not to be an s-t path in N or part of one. But clearly, every s-t path in N is a min-cut hitting set.

(page 56) are equal. There are, however, instances where the solutions to these ILPs are equal, but that cannot be encoded in flow networks. For such a network, see Figure 13.

The problem MIN-CUT HITTING SET is closely related to the following question: If k capacity units can be spent to increase edge capacities of existing edges of a network N, does this suffice to increase the capacity of N by at least one unit? The answer is yes if, and only if, (N, k) is a positive instance of MIN-CUT HITTING SET. This holds at least if all capacities are natural numbers. In that case, we can increase the capacity of every edge of a smallest min-cut hitting set by 1 unit to increase the capacity of the network by 1 unit.

If we allow rational numbers or fixed precision real numbers, it is a bit more technical. This is due to the fact that spending a capacity unit on each of the edges of a smallest min-cut hitting set is not guaranteed to increase the capacity of the entire network by 1 unit. Imagine, for example, the very simple case of a source s and a sink t that are joined by a path of length 2. The first edge has capacity $1/2$, while the second edge has capacity 1. The smallest min-cut hitting set consists of the first edge, but still, 1 capacity unit does not suffice to increase the capacity of the network by 1.

Hence, we first find the lowest common denominator, say d, and then see that if $k \cdot d$ capacity units can be spent, then this suffices to increase the capacity of N by at least d if, and only if, $(N, k) \in$ MIN-CUT HITTING SET. In the just mentioned example, we have $d = 1/2$. If we are given $1/2$ capacity units to spend on edges, then this certainly suffices to increase the network's capacity by $1/2$ unit.

But what if we consider a variation of this problem and try to maximize the amount by which we can increase the capacity of N? Given a network N and a number k, what is the maximum amount by which the network capacity can be increased? We define the following decision version of this problem:

▷ PROBLEM 15 (NETWORK CAPACITY AUGMENTATION (NCA)):

Input: A directed s-t network N and two numbers $k, l > 0$.

Question: Is it possible to increase the capacity of existing edges by a total of k capacity units such that the capacity of N is increased by l units?

In order to increase the capacity of a network, we need to increase the capacities of edges belonging to a min-cut hitting set, in particular a minimal min-cut hitting set (or MCHS), i. e., a min-cut hitting set such that no subset is a min-cut hitting set as well. Doing this, we increase the capacity of the network by at most the minimum increase over all edges of that MCHS. Hence, we will only increase all edges of an MCHS by the same amount. For any MCHS H, by *increasing* H *by* k *capacity units*, we refer to increasing the capacity of each edge in H by k units. This costs us $k \cdot |H|$ capacity units. After H has been increased by k capacity units, H may or may not still be a min-cut hitting set. Let Δ_H be the amount such that increasing H by Δ_H causes H to be no min-cut hitting set anymore, but increasing H by less than Δ_H does not. We call Δ_H the *saturation point* of H. We say that the min-cut hitting set H is *saturated* when increased by Δ_H. If an MCHS cannot be saturated, then $\Delta_H = \infty$.

After having saturated an MCHS H, we may increase a former max-flow by a total of Δ_H units via augmenting paths using the edges of H and possibly more edges. There are edges that are saturated by all new max-flows but have not been saturated by all old max-flows (if this was not the case, then H is not saturated). Recall that by Lemma 9 (page 34), these are exactly the new min-cut edges. The set of all these new min-cut edges after saturating H is denoted by E_H. Two MCHS H_1, H_2 *interfere* if $E_{H_1} \cap E_{H_2} \neq \emptyset$. If N is a network and H an MCHS in N, then let $N(H)$ be network N after saturating H.

Let $\mathcal{H}$ be the set of all smallest min-cut hitting sets for a given network with at least two distinct min-cut hitting sets, and $H_1, H_2 \in \mathcal{H}$ any distinct min-cut hitting sets. Then, there are the following two cases.

- $E_{H_1} \cap E_{H_2} = \emptyset$. In this case, saturating either one of the two min-cut hitting sets leaves the other one unchanged. Hence, if H_1 is saturated first, then H_2 remains unchanged, and vice versa.
- $E_{H_1} \cap E_{H_2} \neq \emptyset$. Saturating one of the two min-cut hitting sets lowers the saturation point of the other. As H_1 and H_2 interfere, the saturation point of one of them is lowered by just the amount of flow that can additionally flow through $E_{H_1} \cap E_{H_2}$ after having saturated the other one. Again, this is symmetric.

It can be concluded, that the total amount of capacity units needed to saturate two min-cut hitting sets is independent of the order in which we saturate them. By induction, the same holds true for any set of min-cut hitting sets. Moreover, no new min-cut hitting sets emerge, they have existed before but may not have been smallest min-cut hitting sets. Hence, in order to maximally increase the capacity of a network N with k available capacity units, we start with a minimum min-cut hitting set, saturate it, and repeat until all k units are spent – see Algorithm 2 for pseudocode.

▷ THEOREM 36: Algorithm 2 is correct and runs in polynomial time.

Proof. The correctness follows from the considerations above. For the runtime, we make the following observations. If the currently smallest min-cut hitting set cannot be saturated, then all capacity units are spent on it and the algorithm terminates. If it can be saturated, then at least one more edge becomes a min-cut edge (while all former min-cut edges stay min-cut edges). After at most $O(|E|)$ iterations, every edge is a min-cut edge and the remaining available capacity units can be completely spent on the smallest min-cut

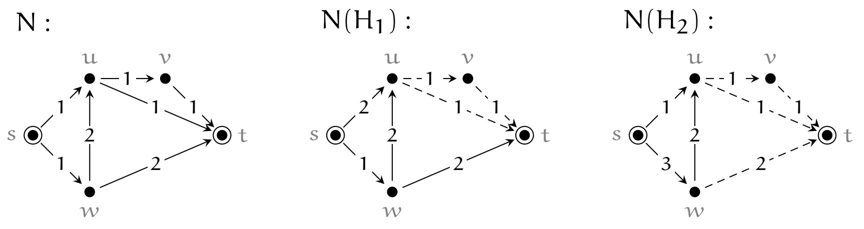


Figure 15: Network N has a capacity of 2 and the only min-cut is $\{(s,u),(s,w)\}$. There are two smallest min-cut hitting sets, namely $H_1 = \{(s,u)\}$ and $H_2 = \{(s,w)\}$. The saturation points are $\Delta_{H_1} = 1$ and $\Delta_{H_2} = 2$. Moreover, we have $E_{H_1} = \{(u,v),(u,t),(v,t)\}$ and $E_{H_2} = \{(u,v),(u,t),(w,t),(v,t)\}$. $N(H_1)$ is N after saturating H_1, and $N(H_2)$ is N after saturating H_2. E_{H_1} and E_{H_2} are represented as dashed edges in the appropriate networks. These are the edges that have not been saturated by all max-flows in N, but are now saturated by all max-flows. Saturating either one of the two min-cut hitting sets lowers the saturation point of the other by 1 unit, as this is the amount of flow that is additionally passing through $E_{H_1} \cap E_{H_2}$, i.e., through the edges (u,v), (u,t), and (v,t). The amount already flowing through these three edges is 1 in N for any max-flow, and it is 2 in $N(H_1)$ and $N(H_2)$ for any max-flow. As the saturation point of H_1 is 1, this min-cut hitting set ceases to exist after H_2 has been saturated, while H_2 is still a min-cut hitting set in $N(H_1)$ – now with a saturation point of 1.

Algorithm 2: Algorithm to maximally increase the capacity of a network by increasing the capacity of existing edges by a total of k units.

```
INPUT: A network $N$ and a number $k > 0$.
OUTPUT: Network $N'$, a maximally augmented version of $N$.
 1: $N' \leftarrow N$
 2: WHILE $k > 0$ DO
 3:    $H \leftarrow$ a smallest min-cut hitting set in $N'$
 4:    $c_b \leftarrow c(N')$
 5:    $c'(e) \leftarrow \infty$ for all $e \in H$
 6:    $\Delta_H \leftarrow c(N') - c_b$
 7:    IF $\Delta_H = \infty$ THEN
 8:       $c'(e) \leftarrow c'(e) + k/|H|$ for all $e \in H$
 9:       RETURN $N'$
10:    ELSE
11:       $d \leftarrow \min\{\Delta_H, k/|H|\}$
12:       $c'(e) \leftarrow c'(e) + d$ for all $e \in H$
13:       $k \leftarrow k - |H| \cdot d$
14:    END IF
15: END WHILE
16: RETURN $N'$
```

hitting set. Since the runtime of each iteration can be bounded by some fixed polynomial p in $|N|$, the runtime is bounded by $O(|E| \cdot p(|N|))$. □

▷ THEOREM 37: NCA is P-complete.

Proof. NCA is in P by Theorem 36. Hardness is shown by a reduction from MIN-CUT HITTING SET. For hardness, we observe that (N, k) belongs to MIN-CUT HITTING SET if, and only if, $(N, k, 1) \in$ NCA – at least if all capacities are natural numbers. If this is not the case, then we scale the capacities of N to obtain natural numbers.

Now, if $(N, k) \in$ MIN-CUT HITTING SET, then there is a min-cut hitting set of size k. Increasing the capacity of each of these k edges by 1 unit suffices to increase the capacity of the network by 1 unit. Similarly, if the capacity of the network can be increased by increasing the capacity of k edges, then there is a min-cut hitting set of size k. □

▷ LEMMA 38: MIN-CUT MINIMUM COVER $\in$ P.

Proof. Due to the Representation Lemma, we solve the problem using N_d. We show that the size of a minimum min-cut cover is equal to the length of a longest s-t path p_l in N_d. By Lemma 15, no min-cut cuts more than one edge of p_l. Thus, in order to cover all min-cut edges, we need at least $|p_l|$ min-cuts. To see that we do not need more either, consider the following algorithm running on N_d that constructs a min-cut cover C of size $|p_l|$.

```
1: $S \leftarrow \{s\}$
2: $C \leftarrow \{S\}$
3: WHILE $S \neq V - \{t\}$ DO
4:     $V' \leftarrow \{v \mid (u, v) \in E(S, V - S)$ and $(w, v) \in E_d \Rightarrow w \in S\}$
5:     $S \leftarrow S \cup V'$
6:     $C \leftarrow C \cup \{S\}$
7: END WHILE
```

We show that C is a min-cut cover for N_d. First, every set S in C is a min-cut since $s \in S$, $t \notin S$, and every edge with one endpoint in S leaves S. This is due to line 4 – we never add a node v to S incident to an edge (w, v) with $w \notin S$. Moreover, for every edge $(u, v) \in E_d$, the algorithm will eventually add u, but not v, to S. Hence, for every edge there is a min-cut cutting it.

The above algorithm finds $|p_l|$ min-cuts since in each step it adds one node of p_l to S, beginning with only the source s. □

▷ LEMMA 39: MIN-CUT EDGE PACKING $\in$ P.

Proof. We show that the largest number of edges in an edge packing is equal to the length of a longest s-t path p_l in N_d. Again Lemma 15 implies that no two edges are cut by the same min-cut. Hence, the largest edge packing has at least $|p_l|$ edges. At the same time, we know that each min-cut cuts p_l leading directly to the fact that there cannot be more than $|p_l|$ edges in an edge packing. It thus suffices to check the length of a longest s-t path in N_d which can be done in polynomial time since N_d is a DAG. □

▷ THEOREM 40: MIN-CUT MINIMUM COVER and MIN-CUT EDGE PACKING are P-complete.

Proof. We have just shown that these problem are in P and that the size of a minimum min-cut cover and the size of a maximum min-cut edge packing are equal to the length of a longest s-t path in N_d. We have already seen in Lemma 28 that this problem is P-complete. □

Again, we observe that the LPs (74) and (75) are restricted to solutions with equal optima. Refer to Figure 14 on page 60 for examples.

▷ COROLLARY 41: MIN-CUT MINIMUM COVER and MIN-CUT ELEMENT PACKING are dual to each other.

Again, while the requirement that an edge set system can be encoded in the min-cuts of a flow network imposes a restriction on these set systems, the restrictions are not characterized by this. There are set systems for which MIN-CUT MINIMUM COVER and MIN-CUT ELEMENT PACKING have integer solutions, but that cannot be encoded in a network, see Figure 13.

Next, we focus on the complexity of another group of problems. While MIN-CUT SET SPLITTING originates from encoding the well-known NP-complete problem SET SPLITTING in networks in the described manner, the others are listed due to their close relationship to this problem.

▷ PROBLEM 16 (MIN-CUT SET SPLITTING):

Input: A directed s-t network N.

Question: Is there a partition of the set of all min-cut edges into two subsets E_1 and E_2 such that no min-cut is entirely contained in either E_1 or E_2?

▷ PROBLEM 17 (SINGLE EDGE MIN-CUT):

Input: A directed s-t network N.

Question: Is there a min-cut in N cutting just one edge?

▷ PROBLEM 18 (MAXIMUM MIN-CUT EDGE SET):

Input: A directed s-t network N and a natural number k.

Question: Is there a min-cut in N cutting at least k edges?

▷ PROBLEM 19 (MINIMUM MIN-CUT EDGE SET):

Input: A directed s-t network N and a natural number k.

Question: Is there a min-cut in N cutting at most k edges?

▷ PROBLEM 20 (MINIMUM MIN-CUT EDGE PATH-COVER):

Input: A directed s-t network and a natural number k.

Question: Is there a *min-cut edge path-cover* of size at most k, i. e., is there a set P of at most k paths such that for every min-cut edge e there is a $p \in P$ with $e \in p$?

▷ PROBLEM 21 (MIN-FLOW):

Input: A directed s-t flow network N with edge demands and a natural number k.

Question: Is there a min-flow of value at most k?

Two of these problems can be quite easily log-space reduced to each other. It holds that N is in SINGLE EDGE MIN-CUT if, and only if, $(N, 1)$ is in MINIMUM MIN-CUT EDGE SET. Due to the connectedness assumption, there is no empty min-cut. The following lemma states that there is another equivalence.

▷ LEMMA 42: MIN-CUT SET SPLITTING $= \overline{\text{SINGLE EDGE MIN-CUT}}$.

Proof. If there is a min-cut set splitting for N, then clearly there is no min-cut in N cutting just one edge. So suppose there is no min-cut in N that cuts just one edge. Pick an arbitrary s-t path p in N_d. Every min-cut cuts p exactly once according to Lemma 15, i. e., every min-cut contains exactly one edge of this path. Since there is no min-cut cutting just one edge, no min-cut is entirely contained in this path. Thus, the edges in p on the one hand and all other edges on the other hand are a min-cut set splitting. □

We have already seen on page 22 how to compute a flow of minimum value in a network with edge demands. The algorithm that has been described there is a polynomial time algorithm. But P is also the lower bound for the complexity of MIN-FLOW.

▷ THEOREM 43: MIN-FLOW is P-complete.

Proof. MIN-FLOW clearly is in P. For hardness, we reduce BANF. The given BANF instance (N, k) is altered to obtain a MIN-FLOW instance (N', k) by degrading the terminal t of N to a non-terminal. Then, we add a new terminal t' and connect t to t' via an edge of infinite capacity and a lower bound of k. All other edges are assigned a lower bound of 0 and maintain their capacity. Now, there is a max-flow in N of value at least k if, and only if, there is a min-flow of value at most k in N'. □

▷ THEOREM 44: MIN-CUT SET SPLITTING and SINGLE EDGE MIN-CUT are P-complete.

Proof. The problems are in P by the algorithm described in Lemma 42.

For hardness, we reduce BANF to SINGLE EDGE MIN-CUT via a log-space function. The network from the instance (N, m) is altered by adding the edge (s, t) with capacity 1. Then, a new node t' and a new edge (t, t') with capacity $m + 1$ is added. This yields network N' and the SINGLE EDGE MIN-CUT instance N'.

If $(N, m) \notin$ BANF, then there is a min-cut of capacity at most $m - 1$ in N and a min-cut of capacity at most m in N'. Hence, the edge (t, t') is not a min-cut. Moreover, every min-cut contains at least two edges due to the (s, t) edge. Therefore, $N' \notin$ SINGLE EDGE MIN-CUT.

If $(N, m) \in$ BANF, then there is a min-cut of capacity at least m in N and of capacity $m + 1$ in N'. Hence, (t, t') represents a min-cut and concluding, $N' \in$ SINGLE EDGE MIN-CUT. □

▷ LEMMA 45: MINIMUM MIN-CUT EDGE SET is P-complete.

Proof. For the given network N, compute N_d, and let $\hat{N}_d$ be network N_d where all edges e have capacity $|\mu(e)|$. Now, let E' be a min-cut in $\hat{N}_d$ with $E' = E(S, T)$. Since E' is a min-cut in $\hat{N}_d$, we know that all edges with exactly one endpoint in S leave S. Hence, (S, T) is also a min-cut in N_d. The min-cut in N cutting the least number of edges thus cuts exactly $c(E')$ edges.

To show hardness, we use the fact that $N \in$ SINGLE EDGE MIN-CUT if, and only if, $(N, 1) \in$ MINIMUM MIN-CUT EDGE SET. □

▷ THEOREM 46: MAXIMUM MIN-CUT EDGE SET is P-complete.

Proof. To determine the min-cut that cuts the largest number of edges, we first compute the max-flow DAG $N_d = (V_d, E_d, \{s, t\}, c_d)$ of N. Then, we set the capacity of each edge e in N_d to $|\mu(e)|$ to obtain $N' = (V_d, E_d, \{s, t\}, c')$. If (S, T) is a min-cut in N_d, then the capacity of (S, T) in N' tells us how many edges are cut by the corresponding min-cut in N. We want to maximize this number now.

To do this, we make use of the Min-Flow Max-Cut Theorem (page 23). We construct a new network $N'' = (V_d, E_d, \{s, t\}, l, c'')$ with edge demands by setting $l(e) = c'(e)$ and $c''(e) = \infty$ for all $e \in E_d$. Now recall the definition of the cut capacity for networks with edge demands (Equation (13) on page 23). With all capacities set to infinity, it implies that the capacity of a cut (S, T) in N'' is maximum if $c(T, S) = 0$, i.e., if there is no edge from

T to S. Under this restriction, $\sum_{e \in E(S,T)} l(e)$ is maximized. For N', this is equivalent to maximizing $\sum_{e \in E(S,T)} c(e)$ under the restriction that (S,T) is a min-cut in N_d. As the sought value is equal to the value of a min-flow, we now compute a min-flow f'' for N'' as described on page 22. The largest number of edges being cut by any min-cut in N is now $|f''|$.

For hardness, we transform a BANF instance (N, m). Let E be the edges in N. Multiply the capacity of each edge with $|E| + 1$. Then, add a new sink t' and $|E| + 1$ new nodes v_i. The original sink t is connected to each v_i and each v_i is connected to t' with an edge of capacity m.

Now assume that $(N, m) \in$ BANF, meaning that $c(N) \geqslant m$. This implies that $c(N') \geqslant m(|E| + 1)$ and $(N', |E| + 1)$ hence belongs to MAXIMUM MIN-CUT EDGE SET.

If $(N, m) \notin$ BANF, then $c(N) < m$ and $c(N') < m(|E| + 1)$ and consequently $(N', |E| + 1) \notin$ MAXIMUM MIN-CUT EDGE SET. □

▷ THEOREM 47: MINIMUM MIN-CUT EDGE PATH-COVER is P-complete.

Proof. To find a minimum min-cut edge path-cover, we compute the max-flow DAG $N_d = (V_d, E_d, \{s, t\}, c_d)$ and then add edge demands to this network in order to obtain $N' = (V_d, E_d, \{s, t\}, l, c')$ with $l(e) = |\mu(e)|$ for each edge e. The capacity $c'(e)$ of each edge is set to infinity. We then find a min-flow f' in N'. Observe that on every s-t path in N', there is an edge e with $f'(e) = |\mu(e)|$ and thus an edge just fulfilling the lower bound on the flow. If there was an s-t path p without this property, then we could subtract a path flow along p of value 1 from f' to obtain a flow of smaller value than f', which is not possible since f' is a min-flow.

By a slight modification of the algorithm for the flow decomposition, we can decompose the flow f' into a set of path flows each of value 1. From this set, we can easily construct a min-cut edge path-cover for the original network N. Moreover, there is no smaller such path-cover as f' is of minimum value in N'.

For hardness, we observe that the smallest min-cut edge path-cover consists of as many paths as the min-cut with the largest min-cut edge set cuts edges. As MAXIMUM MIN-CUT EDGE SET is P-hard, this holds true for MINIMUM MIN-CUT EDGE PATH-COVER as well. □

▷ COROLLARY 48: There is a duality between the number of the fewest paths needed to cover all min-cut edges of a network and the number of min-cut edges cut by the min-cut cutting the most edges.

This corollary is remindful of Menger's Theorem [62]. It states that the maximum number of edge-disjoint paths is equal to the minimum number of edges cut by a min-cut, for unweighted networks. It later was generalized as the famous Min-Cut Max-Flow Theorem.

For the problems considered so far, we cannot only decide in polynomial time whether there is a solution of size k, but also compute a solution of optimal size in polynomial time. Moreover, they all have solutions – there always is a min-cut hitting set, a min-cut set cover, and so on. The difficulty is to identify the optimal one. We now investigate MIN-CUT MINIMUM TEST SET and will show that there may be no solution at all regardless of k. These cases, however, can easily be identified.

We call a min-cut (S,T) for N a *test* for the distinct edges $e_1, e_2 \in E$, if $|\{e_1, e_2\} \cap E(S,T)| = 1$. A *test set* $\mathcal{S}$ for N is a set of min-cuts such that for every pair of min-cut edges in N, there is a test in $\mathcal{S}$.

▷ PROBLEM 22 (MIN-CUT MINIMUM TEST SET):

Input: A directed s-t network N and a natural number k.
Question: Is there a test set for N of size at most k?

▷ LEMMA 49: For a network N, there is no test set at all if, and only if, there is an edge e in the corresponding max-flow DAG N_d with $|\mu(e)| > 1$.

Proof. Assume e is such an edge with $e_1, e_2 \in \mu(e)$. Then, a min-cut in N cuts e_1 if, and only if, it cuts e_2. Hence, there is no test for (e_1, e_2).

Now, if there is no test set for N, then there are two min-cut edges $(u,v),(u',v')$ such that a min-cut in N cuts (u,v) if, and only if, it cuts (u',v'). Suppose there is a min-cut separating u and u', i. e., assume there is a source side S of a min-cut in N with $u \in S$ and $u' \notin S$. Let $\hat{S}$ be the source side of some min-cut in N that cuts (u,v) and (u',v') such that $u, u' \in S$. Then, $\hat{S} \cap S$ is a test for (u,u') as $u \in \hat{S} \cap S$ and $u' \notin \hat{S} \cap S$. This implies that there is no min-cut in N separating u and u'. The same holds true for v and v'. So by definition, u, u' belong to the same SCC in N_f, and v, v' belong to the same SCC in N_f with respect to any max-flow f, represented by u and v. Therefore, we have $|\mu(u,v)| > 1$. □

If there is a test set and p_l is the longest s-t path in N_d, then every test set needs at least $|p_l| - 1$ many tests as otherwise there would be two min-cut edges not cut by any test. On the other hand, a test set of size $|E_d|(|E_d|-1)/2$ is always sufficient. We may obtain it by choosing a test for each pair of edges in N_d. As every solution is thus at most of polynomial size, we have MIN-CUT MINIMUM TEST SET $\in$ NP.

▷ THEOREM 50: MIN-CUT MINIMUM TEST SET is P-hard.

Proof. The proof idea is to reduce the question whether there is an edge e with $|\mu(e)| > 1$ in N_d to the question whether (N, ∞) is a positive instance of MIN-CUT MINIMUM TEST SET. For this, observe that the max-flow DAG of the network created by the reduction in the proof of Theorem 27 on page 52 contains only edges e with $|\mu(e)| = 1$. This is due to the fact that there are no two solid edges e_1, e_2 in Figure 11 on page 54 such that every min-cut cuts e_1 if, and only if, it also cuts e_2. We can, however, artificially introduce such edges by adding a new node v, removing the edge from $(k, x_0, 2)$ to t, adding an edge from $(k, x_0, 2)$ to t with capacity $1/2$, an edge from $(k, x_0, 2)$ to v with capacity ∞, and an edge from v to t with capacity $1/2$. Now, $((k, x_0, 2), t)$ is cut by a min-cut if, and only if, (v,t) is, too. This, in turn, happens if, and only if, the value of the original circuit is `0`. The needed manipulations can easily be done using only logarithmic space. □

We believe that MIN-CUT MINIMUM TEST SET is P-complete as most of the presented problems, i. e., we believe there is an efficient algorithm to find a smallest min-cut test set. However, we can only prove the shown bounds.

The next set of problems involves the counting of min-cuts. The first one of these is a min-cut variant of the well-known problem SET BASIS. The other problems are listed due to their close relationship to this one.

▷ PROBLEM 23 (MIN-CUT SET BASIS):

Input: A directed s-t network N and a natural number k.
Question: Is there a set $\mathcal{E}'$ of at most k min-cuts in N such that for each min-cut E' there is a subset of $\mathcal{E}'$ whose union is exactly E'?

▷ PROBLEM 24 (MIN-CUT MINIMUM NUMBER):

Input: A directed s-t network N and a natural number k.
Question: Does N have at most k distinct min-cuts?

▷ PROBLEM 25 (MIN-CUT MAXIMUM NUMBER):

Input: A directed s-t network N and a natural number k.
Question: Does N have at least k distinct min-cuts?

▷ PROBLEM 26 (MIN-CUT MINIMUM EDGE CUT NUMBER):

Input: A directed s-t network N, an edge e, and a natural number k.
Question: Are there at most k distinct min-cuts in N cutting e?

▷ PROBLEM 27 (MIN-CUT MAXIMUM EDGE CUT NUMBER):

Input: A flow network N, an edge e, and a natural number k.
Question: Are there at least k distinct min-cuts in N cutting e?

▷ LEMMA 51: MIN-CUT MAXIMUM NUMBER and MIN-CUT MAXIMUM EDGE CUT NUMBER are equivalent with respect to polynomial time reductions.

Proof. First, we reduce MIN-CUT MAXIMUM NUMBER by simply adding the edge (s,t) with capacity 1 to the network N to obtain N'. If $\mathcal{E}$ is the set of min-cuts in N, then $\{E' \cup \{(s,t)\} \mid E' \in \mathcal{E}\}$ is the set of min-cuts in N'. Hence, there are at least k min-cuts in N if, and only if, there are at least k min-cuts in N' cutting (s,t).

To reduce MIN-CUT MAXIMUM EDGE CUT NUMBER, we first check if $k = 0$. If so, we transform the instance to $(N, 0)$ which also is a positive instance. If $k > 0$ and if e is no min-cut edge, then (N, k, e) is transformed to a negative instance of MIN-CUT MAXIMUM NUMBER. If $k > 0$ and e is cut by some min-cut, then we transform the instance to the new instance (N', k), where N' is obtained from N lowering the capacity of e by 1 unit. This way, all min-cuts cutting e are preserved (with a capacity one unit less), while all other min-cuts are now no min-cuts anymore. Moreover, every cut that was not a min-cut still is not a min-cut. Hence, (N, k, e) belongs to MIN-CUT MAXIMUM EDGE CUT NUMBER if, and only if, (N', k) belongs to MIN-CUT MAXIMUM NUMBER. □

▷ LEMMA 52: MIN-CUT SET BASIS = MIN-CUT MINIMUM NUMBER.

Proof. Suppose $(N, k) \in$ MIN-CUT SET BASIS. Thus, there is a set $\mathcal{E}'$ of k min-cuts in N such that for each min-cut E' there is a subset of $\mathcal{E}'$ whose union is exactly E'. Since no min-cut is a subset of another min-cut (compare to Lemma 23, page 47), it follows that $\mathcal{E}'$ contains all min-cuts and consequently, there are at most k min-cuts in N.

If $(N, k) \in$ MIN-CUT MINIMUM NUMBER, then there are at most k min-cuts in N, and hence we can simply choose the set of all min-cuts as solution to MIN-CUT SET BASIS. □

MIN-CUT MINIMUM NUMBER is the decision version of the counting problem to determine the number of all min-cuts. This problem is #P-complete [71]. The complexity of MIN-CUT MINIMUM NUMBER is unclear. It does not seem to be in NP as it seems unlikely that a short proof can be guessed and efficiently verified that there are no more than k distinct min-cuts. Rather, we show membership in PP.

▷ LEMMA 53: MIN-CUT MAXIMUM NUMBER $\in$ PP.

Proof. We describe the construction of a Turing machine M that on input (N, k) with $N = (V, E, \{s, t\}, c)$ starts a computation for each subset of E and additionally starts $2^{|E|} - 2k + 1$ computations that accept unconditionally. The machine checks for each subset of E whether it is a min-cut. If so, the computation accepts. This machine has a total of $2^{|E|+1} - 2k + 1$ computations. The number of accepting paths is more than one half if, and only if, at least k subsets of E are min-cuts. □

If MIN-CUT MAXIMUM NUMBER (Problem 25) is not only in PP, but also hard for that complexity class, then the same result is implied for the other four problems defined above. It follows for

- Problem 27 due to Lemma 51,
- Problem 26 since (N, k, e) belongs to Problem 27 if, and only if, $(N, k + 1, e)$ does not belong to Problem 26, and the fact that PP is closed under complement,
- Problem 24 since (N, k) belongs to Problem 25 if, and only if, $(N, k + 1)$ does not belong to Problem 24, and again the fact that PP is closed under complement, and
- Problem 23 due to Lemma 52.

We conjecture that all of these problem are PP-complete. To prove this, we have tried to modify the reduction chain given by Provan and Ball [71] to show #P-completeness of counting the min-cuts. Completeness for #P, however, is defined with respect to Turing reductions and not many-one reduction, as it is the case for PP. The reduction of [71] makes heavy use of Turing reductions such that it seems unclear how to replace them with many-one reductions.

Concluding this section, we can say that the complexity of the min-cut variants we defined depends on whether the problem involves the counting of min-cuts or not. The problems that do not all seem to be P-complete, while the problems that do seem to be PP-complete. Earlier, we stated the question which of the two effects on the complexity of our min-cut variants outweighs. Is it the restriction to the encoded set systems that makes the problems potentially easier or is it the fact that networks may quite succinctly encode huge set systems? From what we have learned, it depends on the problem. For problems not involving the counting of min-cuts, the restriction to the encoded set system outweighs. If a problem does involve the counting of min-cuts, then, quite naturally, the succinctness is a more important aspect than the restrictions to the set systems.

Even though we could ask many of the presented problems for multi-terminal networks, we will not consider this generalization for edge set systems given the open problems for the 2-terminal case.

4 MIMICKING NETWORKS

Early multi-terminal problems deal with networks in which every node is a potential terminal – an example is the problem of determining the min-cut capacity for each pair of nodes mentioned on page 25. Gomory and Hu [36] solved this problem in 1961, and even more: They showed how to construct a tree that maintains the minimum s-t cut values for all pairs $s, t \in V$.

Mimicking networks are undirected networks whose concept has been introduced by Hagerup et al. [40] in 1995. They took the idea of Gomory and Hu one step further by maintaining the min-cut values for all non-trivial terminal bipartitions. They generalized the concept of Gomory and Hu in another way as well. The underlying multi-terminal networks distinguish between terminals and non-terminals. In many network applications, we are mainly interested in the networks' behavior exhibited at terminal nodes. In particular, we are interested in networks that have the same external flow pattern (see page 18) as other networks and thus *mimick* their behavior.

▷ DEFINITION 7 (MIMICKING NETWORK, [40]): Two undirected multi-terminal networks N_1, N_2 over the same terminal set $\mathcal{T}$ are *mimicking networks* of each other if, and only if, $F_{N_1} = F_{N_2}$. N_1 is a *contraction-based mimicking network* (CBMN for short) of N_2 if, and only if, N_1 is a mimicking network of N_2 and the nodes of N_2 can be partitioned such that merging the parts yields N_1.

The term *contraction-based* does not only refer to the contraction of edges, but also to the merging unconnected nodes.

Applications for mimicking networks include those where a network can be decomposed into smaller networks that are interconnected only by a small number of nodes. By temporarily declaring these nodes to be terminals, we may consider each part of the decomposition independently as long as we make sure that the external flow pattern remains unchanged – and this is exactly what mimicking networks do. This technique can be put to good use in networks of bounded treewidth. For example, it allowed Arikati, Chaudhuri, and Zaroliagis [5] to find an optimal solution to the all-pairs min-cut problem in bounded treewidth networks. Also, for integrated circuit layout problems a circuit (network) can be decomposed into smaller circuits that are connected by pins (terminals).

Other problem areas, where decompositions are not applicable, may profit from these considerations as well. Flow networks are used to model problems in a wide range of settings and in many these networks tend to get larger and larger making computations more costly. A strategy to tackle this problem is graph compression – a concept describing techniques to obtain smaller networks that preserve key properties of the original graph. Finding a smaller or even smallest mimicking network tackles this question.

This chapter starts off with a review of the results related to mimicking networks in Section 4.1. This includes a closer look at the different types of mimicking networks and results from the literature mainly comprising size bounds for mimicking networks.

In Section 4.2 we prove that the unique min-cuts assumption is a restriction. A family of networks N_k with k terminals is constructed such that N_k has several min-cuts for most terminal bipartitions. By an extensive analysis of the flow capacities of all possible cuts we show that depending on which nodes are merged one can arrive either at the minimum CBMN that has two non-terminals left or at an *irreducible* CBMN of size exponential in k. Thus, in this general setting a simple greedy implementation merging nodes if they can be merged may produce solutions very far from optimality.

By slightly changing the edge capacities we can force either solution to be the unique minimum CBMN. This implies that slight perturbations may drastically change the topology and size of the minimum CBMN. We also derive implications on the quality of so-called *cut sparsifiers* that approximately mimick the behavior of the original network with a limited number of non-terminals [65].

In Section 4.3 we analyze the complexity of deciding whether a CBMN of a given size exists without making the assumption that min-cuts are unique. Since the described techniques to make min-cuts unique may increase the size of a minimum CBMN greatly, all min-cuts need to be taken into account to correctly find a smallest CBMN. This comes at the expense of increasing the complexity of the problem.

We are first to consider this problem in a complexity theoretic way that does not only derive upper bounds, but also lower bounds. It will be shown that deciding whether a given network has a CBMN of a specific size is coNP-hard and contained in Σ_2^P. We also investigate these questions in the fixed-parameter setting. For the problem of deciding whether l nodes exist that can be merged to one node, we prove tractability when parameterized by the number of terminals.

Furthermore, we show that finding the smallest contraction-based mimicking network is equivalent to MIN-CUT MINIMUM TYPE SELECTION defined in Chapter 3 on page 43 linking the question of the complexity to the difficulties arising when describing the min-cut set systems.

In order to further understand the question of finding a minimum CBMN, we carefully analyze this problem restricted to networks with 4 terminals in Section 4.4. We pinpoint those cases causing the difficulty and hence give starting points for further investigations.

4.1 REVIEW OF MIMICKING NETWORKS

The key lemma for finding contraction-based mimicking networks is the following lemma by Hagerup et al. [40].

▷ LEMMA 54 ([40]): N_1, N_2 over the same terminal set $\mathcal{T}$ are mimicking networks of each other if, and only if, $c_{N_1,B} = c_{N_2,B}$ for all $B \in \mathcal{B}$.

If there are two nodes u, v such that for every $B \in \mathcal{B}$ there is a min-cut not separating u and v, then u, v may be merged to obtain a contraction-based mimicking network. Consider network N of Figure 16 on page 75. There are two t_1-min-cuts ($\{t_1\}$ and $\{t_1, u\}$), two t_2-min-cuts ($\{t_2\}$ and $\{t_2, v\}$), and one t_3-min-cut ($\{t_3\}$). Merging the pair t_1,u maintains the min-cut values and thus yields a contraction-based mimicking network.

When we first defined the multi-terminal network in Chapter 2, we said that we assume a total order imposed over the terminals $\mathcal{T}$. If there is another network over the same set of terminals $\mathcal{T}$, then we assume that the same total order is imposed over it. We just defined mimicking networks over the same set of terminals. However, networks may be mimicking each other even if the terminal sets are not identical, but of equal size. In that case a bijection is needed telling us which terminals correspond to each other. Since mimicking networks are usually constructed from a given one, we may assume that both networks have the same terminal set. In the context of computational complexity, however, we will also consider the case where the terminal sets are not the same.

In order to clearly differentiate between the two kinds of mimicking networks described in the last definition, we will from now on refer to mimicking networks as *unrestricted mimicking networks*. Contraction-based mimicking networks of a given network N often are a true subset of unrestricted mimicking networks. In particular, the smallest contraction-based mimicking network often is larger than the smallest unrestricted mimicking network. Network N'' of Figure 16 (page 75) is its own smallest contraction-based mimicking network. We call such a network an *irreducible contraction-based mimicking network*. There is, however, an unrestricted mimicking network of N'' with no non-terminals at all – network N''' of Figure 16.

Even though CBMNs are obtained by merging nodes, a CBMN is not necessarily a minor of the original network. This is due to the fact that we may merge nodes that are not joined by an edge. In particular, there are networks whose smallest CBMN is not a minor of the original network.

The problem of finding mimicking networks with as few nodes as possible has recently experienced a renewed interest. Since a network may have a vast number of non-terminals in comparison to the number of terminals, this question is very natural. Research on mimicking networks has focused on bounding the size of the smallest unrestricted mimicking network with respect to the number k of terminals for arbitrary graphs and for restricted classes, for example graphs with bounded treewidth. Since one has to consider exponentially in k many cuts, a quite obvious upper bound for the size of any smallest mimicking network for arbitrary networks is 2^{2^k} as already noted by Hagerup et al. [40]. Chambers and Eppstein [10] as well as Khan and Raghavendra [52] noticed that the size bound can be reduced slightly to $2^{\binom{k-1}{(k-1)/2}}$. They define a cluster to be a set of non-terminals not separated by any min-cut and then show that many of the at most 2^{2^k} clusters are empty.

For certain graph classes the size bound has been further improved. Networks of treewidth t have mimicking networks of size at most $k\,2^{2^{3(t+1)}}$, which grows only linear in k, and for outerplanar networks (having treewidth 2) the size bound can be improved to $10k-6$ as found by Chaudhuri et al. [12] already in 2000. The best known lower bound for arbitrary k-terminal networks is $2^{\Omega(k)}$, which has been established for a family of bipartite graphs by Khan and Raghavendra [52] and Krauthgamer and Rika [55]. Furthermore, for $k \leqslant 3$ one can always find a mimicking network with terminals only. For $4 \leqslant k \leqslant 5$ at most one non-terminal is needed [12].

Khan and Raghavendra [52] and Krauthgamer and Rika [55] assume that for every terminal bipartition there is a unique min-cut. We will refer to it as the *unique min-cuts assumption*. It is argued that this can always be achieved by slight perturbations of the edge capacities or by maintaining only those min-cuts with, for example, smallest source side. Under this assumption, Krauthgamer and Rika [55] show that planar networks have a mimicking network of size $O(k^2\,2^{2k})$ and Khan and Raghavendra [52] present an algorithm that always yields a *contraction-based* mimicking network of minimum size. This algorithm runs in time polynomial in n and 2^k. This is done by computing the by assumption unique min-cut for each terminal bipartition and then merging all nodes that are not separated by any of these min-cuts. Thus, the problem of deciding whether a CBMN of size k exists for a given network with unique min-cuts is in FPT when parameterized over the number of terminals.

By relaxing the goal of exactly preserving the min-cut values and allowing no non-terminals to do so, one arrives at *vertex sparsifiers* or *cut sparsifiers* (these terms are used synonymously). This notion has first been introduced by Moitra [64] in 2009 and was quickly followed by several publications by Englert et al. [27] as well as Leighton and Moitra [57]. Ankur Moitra dedicated his PhD thesis to this topic [66]. Again, one may differentiate between unrestricted and contraction-based cut sparsifiers. Formally, the *quality* (sometimes called *ratio*) of a cut sparsifier is α if the min-cut values of corresponding terminal bipartitions in the original network and the cut sparsifier are within a factor of α of each other. Mimicking networks are thus cut sparsifiers with quality 1. In 2012, Chuzhoy [13] expanded the notion of cut sparsifiers to include those that *do* contain non-terminals. These are the cut sparsifiers we will refer to. Without any additional non-terminals the best possible approximation ratio achievable has been shown to be between $\Omega(\log^{1/4} k)$ and $O(\log k/\log\log k)$ [11, 65]. A constant ratio can be obtained by cut sparsifiers with additional non-terminals the number of which is polynomially related to the edge capacities of the terminals [13].

While the size bounds are usually proven for unrestricted mimicking networks and are hence also applicable for contraction-based mimicking networks, the only known algorithm for actually computing a mimicking network is the algorithm by Khan and Raghavendra [52] computing *contraction-based* mimicking networks. From a complexity point of view, this algorithm is only an upper bound. It is unknown whether there exists a more efficient algorithm.

By a slight change of our network family N_k in dependence of a parameter δ, we can show that for arbitrary $\delta > 0$ one can find a smaller $\delta' > 0$ such that N_k has a cut sparsifier with 2 non-terminals and ratio at most $1 + \delta$, but every contraction-based cut sparsifier achieving approximation ratio $1 + \delta'$ requires exponentially in k many non-terminals.

4.2 THE UNIQUE MIN-CUTS ASSUMPTION

This section is dedicated to the unique min-cuts assumption. To assume that there is a unique min-cut for each terminal bipartition may be realistic if each edge has a unique capacity. In that case, it may be highly unlikely that the sum of different edge sets add up to the same value. In particular if the capacities are all unique and irrational, the probability for this is 0. In many real applications, however, many edges will have equal capacities. This, in turn, quickly leads to multiple min-cuts.

We could deal with this by perturbing each edge's capacity by a small random amount. While this does make the min-cuts unique, it also changes the network. One could think that this is equivalent to randomly choosing a min-cut for each terminal bipartition. However, not every choice of min-cuts corresponds to a random perturbation. This is implied by Lemma 19 (page 42), as two min-cuts of different terminal bipartitions may induce another min-cut for the same terminal bipartition leading to multiple min-cuts. But this is impossible after the random perturbation. So what if the optimal choice of min-cuts is one that does not correspond to a random perturbation?

We will show in this section that there is a network whose smallest CBMN contains two non-terminals, but small random perturbations may change the network in such a way that the smallest CBMN is exponential in the number of terminals.

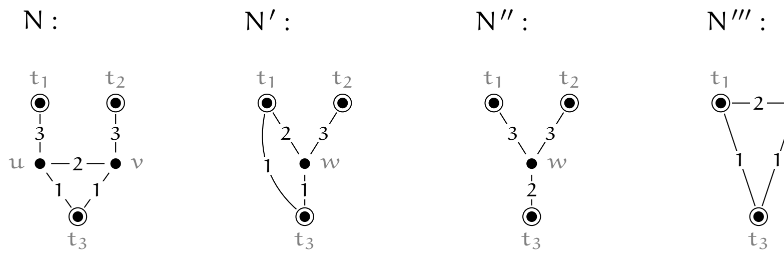


Figure 16: Network N allows the three CBMNs to its right where N'' and N''' are irreducible, and N''' clearly is the minimum CBMN. When picking one min-cut for each terminal bipartition and then merging all nodes not separated by any picked min-cut, one does not necessarily reach an irreducible CBMN. In this example, picking $\{t_1, u\}$, $\{t_2\}$, and $\{t_3\}$ for N yields N' which can be further reduced by contractions to obtain N'''. Another effect that can be witnessed is that merging some nodes may yield an irreducible CBMN that is not minimum in size and thus a local minimum – N'' is such a dead end. Theorem 59 on page 86 states that a minimum CBMN cannot be missed when first merging non-terminals with terminals only. This rule here leads to N'''. Moreover, N'' is its own smallest CBMN, while N''' is the smallest unrestricted mimicking network for N''.

Before getting to this result, we need some more tools. The following lemma is a direct consequence of Lemma 54 and a well-known fact.

▷ LEMMA 55: Two nodes u, v of a network N can be merged to obtain a CBMN N' of N if, and only if, for every $B \in \mathcal{B}$ there is a B-min-cut that does not separate u, v.

As a consequence, each CBMN of N corresponds to a specific choice of a min-cut for each terminal bipartition. This is formally described by the function $\gamma : \mathcal{B} \to 2^V$ such that for every $B \in \mathcal{B}$, $(\gamma(B),\ V - \gamma(B))$ is the *chosen* B-min-cut. Two nodes (u, v) are in relation with respect to the equivalence relation R_γ if there is no $B \in \mathcal{B}$ with $|\gamma(B) \cap \{u, v\}| = 1$, which would mean that the nodes are separated by some chosen cut. Merging all nodes of the same equivalence class to a single node yields a CBMN that might or might not be an irreducible CBMN as there still may be contractible nodes left, see Figure 16. A *minimum* CBMN for a given network N is a CBMN with the smallest number of nodes among all CBMNs for N.

Note that if there are two pairs of nodes u, u' and v, v' such that each pair of nodes can be merged in the sense of Lemma 55, then merging one pair may render the other pair unmergeable. Node pairs t_1, u and nodes u, v of network N in Figure 16, for example, both can be merged, but still we cannot merge all three of these nodes. Figure 19 on page 91 is another example of such a situation with two pairs of nodes that do not share a common node.

As the example in Figure 16 shows, the selection of min-cuts, may have an influence on the size of the irreducible CBMN obtained. Thus, it is not clear how to construct a minimum CBMN – a simple greedy strategy does not work. Knowing this, a natural question is how far away from optimality an arbitrary choice of min-cuts can lead. Note that there may be several minimum CBMNs as Figure 19 on page 91 illustrates.

We will show that the difference in size of irreducible CBMNs for the same network can be exponential in the number of terminals. For this purpose, a family of networks $\mathcal{N} = \{N_k \mid k \geqslant 4\}$ with k terminals will be constructed. Let us first consider even k and let $K = \binom{k}{k/2}$.

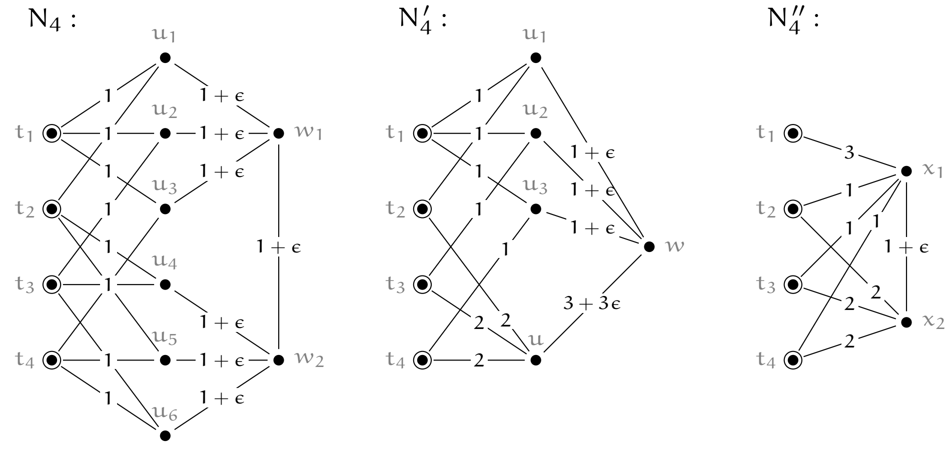


Figure 17: N_4 is an example of our network family $\mathcal{N}$. Its nodes V_4 consist of terminals $T_4 = \{t_1, t_2, t_3, t_4\}$, non-terminals $U_4 = \{u_1, u_2, \ldots, u_6\}$, and non-terminals $W_4 = \{w_1, w_2\}$. The edges and their capacities are shown according to the described construction. Now, N_4 allows the irreducible CBMN N_4' with 5 non-terminals (by merging u_4, u_5, u_6 to u and w_1, w_2 to w) and the minimum CBMN N_4'' with 2 non-terminals (by merging u_1, u_2, u_3, w_1 to x_1 and u_4, u_5, u_6, w_2 to x_2). The capacity of every T'-min-cut for non-trivial $T' \subset \mathcal{T}$ and $|T'| = 2$ is $5 + \epsilon$ in every network: In N_4, there are three $\{t_2, t_3\}$-cuts, namely $\{t_2, t_3, u_4\}$, $\{t_2, t_3, u_1, u_2, u_4, u_5, u_6, w_1, w_2\}$, and $\{t_2, t_3, u_4, u_5, u_6, w_2\}$. The only $\{t_2, t_3\}$-min-cut in N_4' is $\{t_2, t_3, u_1, u_2, u, w\}$, in N_4'' it is $\{t_2, t_3, x_2\}$.

The nodes V_k of N_k consist of three sets: terminals $\mathcal{T}_k = \{t_1, t_2, \ldots, t_k\}$, non-terminals $U_k = \{u_1, u_2, \ldots, u_K\}$, and non-terminals $W_k = \{w_1, w_2\}$. There thus is an exponential number of non-terminals. When clear from the context, we neglect the k in the subscript of these sets. To the nodes in U we assign as labels the binary strings of length k with exactly $^k/_2$ 1s. Let U_1 be those nodes for which the first symbol of the label is a `1` and U_2 those starting with a `0`. A terminal t_i is connected to node u_j by an edge of capacity 1 if, and only if, the label of u_j has a `1` at the i^{th} position. Note that U_1 now is the neighborhood of t_1. Furthermore, all nodes in U_1 have an edge to w_1, and all nodes in U_2 an edge to w_2, each with capacity $^k/_2 - 1 + \epsilon$ for some $0 < \epsilon \ll 1$. Finally, w_1 and w_2 are connected by an edge with capacity $^k/_{(4(k-1))}K - 1 + \epsilon$. Thus, each terminal has degree $^K/_2$, each node in U degree $^k/_2 + 1$, and each node in W degree $^K/_2 + 1$. All capacities are clearly positive for all considered k. For odd k, we use the construction for $k - 1$ and add an additional terminal t_k that is only connected to t_{k-1} by an edge of capacity 1.

As an example, N_4 is illustrated in Figure 17. Note that the capacity of edges between nodes of U and W and between the two nodes in W is $1 + \epsilon$ and hence equal just for $k = 4$ by coincidence. In general these numbers differ and grow with k.

In this setting, we will not talk about terminal bipartitions $B \in \mathcal{B}$, but about arbitrary non-trivial terminal subsets $T' \subset \mathcal{T}$. Moreover, we abbreviate the set-theoretic union and set-theoretic difference of a set A with a singleton $\{b\}$ with $A + b$ and $A - b$, respectively.

▷ THEOREM 56: For all $k \geqslant 4$, N_k is a network with k terminals that has an irreducible CBMN of size $k + 2 + \binom{k}{k/2}/2$ and a minimum CBMN of size $k + 2$.

Proof. We will show that $N_k \in \mathcal{N}$ is such a network. For this, we first assume that k is even. For every $T' \subset \mathcal{T}$ with $|T'| \neq k/2$, there is exactly one T'-min-cut (T' if $|T'| < k/2$ and $V - \mathcal{T} \cup T'$ if $|T'| > k/2$) not separating any non-terminals. We continue explaining the idea of the construction before giving a formal proof of this claim later. Choosing these min-cuts is optimal and hence, we will from now on only consider those $T' \subset \mathcal{T}$ with $|T'| = k/2$.

N_k is constructed such that there are three T'-min-cuts for every T' with $|T'| = k/2$. For the first min-cut note that for every T', there is exactly one node in U denoted by $u_{T'}$ which is connected to all nodes in T'. Due to the chosen capacities, $T' + u_{T'}$ is the first min-cut with the capacity $c(T' + u_{T'}) = k/4\ K - 1 + \epsilon$.

Without $u_{T'}$ the capacity is slightly higher with $c(T') = k/4\,K$. Adding more nodes to $T' + u_{T'}$ results in a higher capacity unless adding all nodes except for $\mathcal{T} - T'$ and the node $u_{\overline{T'}}$ that is not connected to any nodes in T'. This cut $T' \cup (U - u_{\overline{T'}}) \cup W$ has capacity $k/4\ K - 1 + \epsilon$ as well which is due to the fact that it is the complement of the $\overline{T'}$-min-cut $\overline{T'} + u_{\overline{T'}}$.

For the third min-cut, an important observation is that while the neighborhood of t_1 is exactly U_1, there is no $t_i \in \mathcal{T}$ whose neighborhood is exactly U_2. If there were such a node t_i, then labels having a 1 at the positions 1 and i do not occur (remember $k \geqslant 4$, thus each label has at least two 1s and they must occur at any pair of positions). This asymmetry is the reason why $T' \cup U_1 + w_1$ ($T' \cup U_2 + w_2$) is a T'-min-cut for all $T' \subset \mathcal{T}$ with $|T'| = k/2$ and $t_1 \in T'$ ($t_1 \notin T'$). In the following, we will calculate the capacity of this cut.

First let us calculate the number of edges from subsets $T' \subseteq \mathcal{T}$ to U_1 and U_2. Due to the just described asymmetry, we introduce two new versions of the capacity function $c : 2^V \times 2^V \to \mathbb{R}$, namely c_{t_1} and $c_{\overline{t_1}}$. The former, $c_{t_1}(V', V'')$, is only defined for all V' with $t_1 \in V'$, and $c_{\overline{t_1}}(V', V'')$ is only defined if $t_1 \notin V'$. We do this to clearly differentiate between cases where V' contains t_1 and where it does not. By construction, t_1 is incident to all nodes in U_1. Every node in U_1 is incident to $k/2 - 1$ nodes in $\mathcal{T} - t_1$. Since there are $K/2$ nodes in U_1, there are $(k/2 - 1) \cdot K/2$ edges between $\mathcal{T} - t_1$ and U_1. As $|\mathcal{T} - t_1| = k - 1$, the total number of edges connecting nodes in T' with nodes in U_1 is

$$c_{t_1}(T', U_1) = \frac{1}{2}K + \frac{|T'| - 1}{k - 1} \cdot \left(\frac{k}{2} - 1\right) \cdot \frac{K}{2}. \tag{76}$$

The total number of edges between $\mathcal{T}$ and U is $k/2 \cdot K$. The number of edges that is incident to nodes in T' is $c(T', U) = |T'| \cdot K/2$. Hence, the number of edges from T' to U_2 is

$$\begin{aligned}
&c_{t_1}(T', U_2)\\
&= c(T', U) - c_{t_1}(T', U_1)\\
&= \frac{|T'|}{2}K - \left(\frac{1}{2}K + \frac{|T'| - 1}{k - 1} \cdot \left(\frac{k}{2} - 1\right) \cdot \frac{K}{2}\right)\\
&= \left(\frac{|T'|}{2} - \frac{1}{2} - \frac{|T'| - 1}{2(k-1)}\left(\frac{k}{2} - 1\right)\right)K\\
&= \left(\frac{2|T'|k - 2|T'| - 2k + 2 - |T'|k + k + 2|T'| - 2}{4(k-1)}\right)K\\
&= \frac{k(|T'| - 1)}{4(k-1)}K.
\end{aligned} \tag{77}$$

Note that

$$c(T',U_1)+c(T',U_2)=c(T',U)=|T'|\cdot {}^{K}\!/2 \text{ and} \tag{78}$$

$$c(T',U_1)+c(\overline{T'},U_1)=c(\mathcal{T},U_1)=k\cdot {}^{K}\!/4\,. \tag{79}$$

It follows

$$\begin{aligned} &c(\overline{T'},U_1)\\ &=c(\mathcal{T},U_1)-c(T',U_1)\\ &=\frac{k}{4}K-\left(\frac{|T'|}{2}K-c(T',U_2)\right)\\ &=\frac{k-2|T'|}{4}K+c(T',U_2)\,. \end{aligned} \tag{80}$$

Let us take a look at $T'\subseteq\mathcal{T}$ with $t_1\notin T'$. In this case, we have

$$c_{\overline{t_1}}(T',U_1)=\frac{|T'|}{k-1}\cdot\left(\frac{k}{2}-1\right)\cdot\frac{K}{2} \text{ and} \tag{81}$$

$$\begin{aligned} c_{\overline{t_1}}(T',U_2)&=c(T',U)-c(T',U_1)\\ &=\frac{|T'|}{2}K-\frac{|T'|}{k-1}\cdot\left(\frac{k}{2}-1\right)\cdot\frac{K}{2}\\ &=\frac{k|T'|}{4(k-1)}K\,. \end{aligned} \tag{82}$$

The capacity of $T'\cup U_1+w_1$ now amounts to

$$\begin{aligned} &c(T'\cup U_1+w_1)\\ &=c_{t_1}(T',U_2)+c_{\overline{t_1}}(\overline{T'},U_1)+c(w_1,w_2)\\ &=\frac{k(k/2-1)}{4(k-1)}K+\frac{k/2}{k-1}\cdot\left(\frac{k}{2}-1\right)\cdot\frac{K}{2}+\frac{k}{4(k-1)}K-1+\epsilon\\ &=\frac{k}{4}K-1+\epsilon\,. \end{aligned} \tag{83}$$

So far, we have shown that the claimed min-cuts are all of equal capacity. Assuming they really are min-cuts and that there are no further min-cuts, we now continue explaining the idea of the construction. That they truly are min-cuts will be shown afterwards.

Due to these three min-cuts, merging w_1 and w_2 eliminates the possibility of choosing $T'\cup U_1+w_1$ as a min-cut since it separates w_1 and w_2. So now, for each T' with $|T'|={}^{k}\!/2$, there are two T'-min-cuts left in N'_k which are $T'+u_{T'}$ and $T'\cup(U-u_{\overline{T'}})\cup W$. Hence, it is now possible to choose min-cuts in order to merge half of the nodes in U, for example all nodes in U_2 as pictured in Figure 17. The size of this irreducible network then is $k+2+{}^{K}\!/2$.

If we do not merge w_1 and w_2, we may choose the min-cut $T'\cup U_1+w_1$ for each T'. Then all nodes in U_1+w_1 as well as all nodes within U_2+w_2 can be merged resulting in the minimum CBMN N''_k of size $k+2$. Both irreducible networks are shown in Figure 17 for $k=4$.

We will now start showing, that all claimed min-cuts truly are min-cuts and that there are no more min-cuts. This is achieved by comparing their capacities to all other possible cuts. This is rather long and technical and continues to page 85.

Recall that $K = \binom{k}{k/2}$ and that k was assumed to be even. For odd k we use the construction for $k-1$ and add an additional terminal t_k that is only connected to t_{k-1} by an edge of capacity 1.

We now compare the capacities of the two T'-cuts $T' \cup U_1 + w_1$ and $T' \cup U_2 + w_2$. First, we assume $t_1 \in T'$:

$$\begin{aligned}
&c(T' \cup U_2 + w_2) - c(T' \cup U_1 + w_1)\\
&= c_{t_1}(\overline{T'}, U_2) + c_{t_1}(T', U_1) + c(w_1, w_2)\\
&\quad - \big(c_{t_1}(\overline{T'}, U_1) + c_{t_1}(T', U_2) + c(w_1, w_2)\big)\\
&= \frac{k}{4}K - 2c_{t_1}(T', U_2) + c_{t_1}(T', U_1) - \left(\frac{k}{4}K - c_{t_1}(T', U_1)\right)\\
&= 2c_{t_1}(T', U_1) - 2c_{t_1}(T', U_2)\\
&= K + \frac{|T'|-1}{k-1} \cdot \left(\frac{k}{2} - 1\right) \cdot K - \frac{k(|T'|-1)}{2(k-1)}K\\
&= \left(\frac{2k - 2 + k|T'| - 2|T'| - k + 2 - k|T'| + k}{2(k-1)}\right)K\\
&= \frac{k - |T'|}{k-1}K
\end{aligned} \tag{84}$$

Now, we assume $t_1 \notin T'$:

$$\begin{aligned}
&c(T' \cup U_1 + w_1) - c(T' \cup U_2 + w_2)\\
&= c_{\overline{t_1}}(\overline{T'}, U_1) + c_{\overline{t_1}}(T', U_2) + c(w_1, w_2)\\
&\quad - \left(c_{\overline{t_1}}(T', U_1) + c_{\overline{t_1}}(\overline{T'}, U_2) + c(w_1, w_2)\right)\\
&= \frac{k}{4}K - 2c_{\overline{t_1}}(T', U_1) + c_{\overline{t_1}}(T', U_2) - \left(\frac{k}{4}K - 2c_{\overline{t_1}}(T', U_2)\right)\\
&= 2c_{\overline{t_1}}(T', U_2) - 2c_{\overline{t_1}}(T', U_1)\\
&= 2\left(\frac{k|T'|}{4(k-1)}K\right) - 2\left(\frac{|T'|}{k-1} \cdot \left(\frac{k}{2} - 1\right) \cdot \frac{K}{2}\right)\\
&= \frac{k|T'|}{2(k-1)}K - \frac{|T'|}{k-1} \cdot \frac{k-2}{2} \cdot K\\
&= \frac{|T'|}{k-1}K
\end{aligned} \tag{85}$$

The terms (84) and (85) are greater than zero for all non-trivial T' with $t_1 \in T'$ and $t_1 \notin T'$. This implies that for all non-trivial T' we have $c(T' \cup U_1 + w_1) < c(T' \cup U_2 + w_2)$ if $t_1 \in T'$ and $c(T' \cup U_2 + w_2) < c(T' \cup U_1 + w_1)$ if $t_1 \notin T'$.

We are now ready to show that for every $T' \subseteq \mathcal{T}$ with $|T'| \neq k/2$, the set of T'-min-cuts is $\{T'\}$ if $|T'| < k/2$ and $\{V - T'\}$ if $|T'| > k/2$, and for every T' with $|T'| = k/2$ and $t_1 \in T'$ this set is $\big\{T' + u_{T'}, T' \cup U_1 + w_1, T' \cup (U - u_{\overline{T'}}) \cup W\big\}$.

1. Let $T' \subseteq \mathcal{T}$ be any set with $|T'| \neq k/2$. Due to symmetry it suffices to consider those sets with $|T'| < k/2$. In the following, we will analyze the capacities of all T'-cuts $(S, V - S)$.

   a) Let S be an T'-cut with $W \cap S = \emptyset$. T' is such a cut with capacity

$$c(T') = |T'| \cdot \frac{K}{2}. \tag{86}$$

Now let $S = T' \cup U'$ with $U' \subset U$. Obviously, S is a T'-cut. We show that $c(S + u_i) > c(S)$ for any $U' \subset U$ and $u_i \in U - U'$. Since $|T'| < k/2$, u_i is connected to at least one node in $\mathcal{T} - T'$. Hence, the new capacity is

$$\begin{aligned} c(T' \cup U' + u_i) &\geqslant c(T' \cup U') - \left(\frac{k}{2} - 1\right) + \frac{k}{2} - 1 + \epsilon \\ &= c(T' \cup U') + \epsilon \,. \end{aligned} \tag{87}$$

The capacity is larger with u_i than it is without u_i. We conclude that without any nodes of W, the cut is smallest if no node of U is contained in the cut, that is if the cut is T'.

b) Next, consider any T'-cut $(S, V - S)$ with $|S \cap W| = 1$. First, we assume $w_1 \in S$. It pays off to move a non-terminal $u_i \in U_1$ to U, if the capacity that is saved this way is at least as large as the capacity gained, i. e., if

$$\begin{aligned} & c(T', \{u_i\}) + c(u_i, w_1) \geqslant c(\overline{T'}, \{u_i\}) \\ \Longleftrightarrow \quad & c(T', \{u_i\}) \geqslant c(\overline{T'}, \{u_i\}) - \frac{k}{2} + 1 - \epsilon \,. \end{aligned} \tag{88}$$

Since $c(\overline{T'}, \{u_i\}) = k/2 - c(T', \{u_i\})$, this is equivalent to

$$\begin{aligned} & c(T', \{u_i\}) \geqslant \frac{k}{2} - c(T', \{u_i\}) - \frac{k}{2} + 1 - \epsilon \\ \Longleftrightarrow \quad & c(T', \{u_i\}) \geqslant \frac{1}{2}(1 - \epsilon) \,. \end{aligned} \tag{89}$$

This inequality holds for all $u_i \in U_1$ that are connected to at least one node in T'. Hence, we conclude that the cut containing w_1 but not w_2 can only be smallest, if it contains all nodes $U_1' \subseteq U_1$ that are connected to at least one in T'. If $t_1 \in T'$, then clearly $U_1' = U_1$. If $t_1 \notin T'$, then

$$\begin{aligned} |U_1 - U_1'| &= |\{u_i \in U_1 \mid c(\{u_i\}, T') = 0\}| \\ &\leqslant |U_1| - \left(\frac{1}{k-1} \cdot \frac{k-2}{2} \cdot \frac{1}{2} \cdot K + |T'| - 1\right) \\ &= \frac{k}{4(k-1)} K - |T'| + 1 \,. \end{aligned} \tag{90}$$

Now, we ask for which $u_i \in U_2$ it pays off to move it to U. This is the case if

$$\begin{aligned} & c(T', \{u_i\}) \geqslant c(\overline{T'}, \{u_i\}) + c(u_i, w_2) \\ \Longleftrightarrow \quad & c(T', \{u_i\}) \geqslant \frac{k}{2} - c(T', \{u_i\}) + \frac{k}{2} - 1 + \epsilon \\ \Longleftrightarrow \quad & c(T', \{u_i\}) \geqslant \frac{1}{2}(k - 1 + \epsilon) \,. \end{aligned} \tag{91}$$

Our assumption $|T'| \leqslant k/2 - 1$ implies $c(T', \{u_i\}) \leqslant k/2 - 1$. Consequently, the above inequality does not hold for any $u_i \in U_2$. Hence, the cut S containing w_1 but not w_2 can only be smallest if $S \cap U_2 = \emptyset$. Altogether, the smallest T'-cut S for $|T'| < k/2$ with $w_1 \in S$ and $w_2 \notin S$ is $T' \cup U_1' + w_1$ where $U_1' \subseteq U_1$ is the set of nodes that are connected to at least one node in T'. The capacity of this cut is

$$\begin{aligned} &c(T' \cup U_1' + w_1) \\ &= c(T' \cup U_1 + w_1) - |U_1 - U_1'| \left(\frac{k}{2} - \left(\frac{k}{2} - 1 + \epsilon \right) \right) \\ &\geqslant c(T' \cup U_1 + w_1) - \left(\frac{k}{4(k-1)} K - |T'| + 1 \right)(1 - \epsilon) \,. \end{aligned} \tag{92}$$

c) Now, we assume $w_2 \in S$. It pays off to move a node $u_i \in U_1$ to S if

$$\begin{aligned} & c(T', \{u_i\} \geqslant c(\overline{T'}, \{u_i\}) + c(u_i, w_1) \\ \iff \quad & c(T', \{u_i\}) \geqslant \frac{1}{2}(k - 1 + \epsilon) \,, \end{aligned} \tag{93}$$

which is true for no $u_i \in U_1$ since $|T'| < k/2$. It pays off to move a node $u_i \in U_2$ to S if

$$\begin{aligned} & c(T', \{u_i\}) + \frac{k}{2} - 1 + \epsilon \geqslant c(\overline{T'}, \{u_i\}) \\ \iff \quad & c(T', \{u_i\}) \geqslant \frac{1}{2}(1 - \epsilon) \end{aligned} \tag{94}$$

which is true for all $u_i \in U_2$ that have at least one edge to T'. Hence, quite symmetrically, S is a T'-cut of smallest capacity, if $S = T' \cup U_2' + w_2$ where $U_2' \subseteq U_2$ contains all nodes that are connected to at least one node in T'. Similarly to the analysis above, we get

$$\begin{aligned} |U_2 - U_2'| &= \left| \{ u_i \in U_2 \mid c(\{u_i\}, T') = 0 \} \right| \\ &\leqslant |U_2| - \left(\frac{k(|T'| - 1)}{4(k-1)} K + |T'| - 1 \right) \\ &= \frac{k-2}{4(k-1)} K - |T'| + 1 \,. \end{aligned} \tag{95}$$

The capacity of this cut is

$$\begin{aligned} &c(T' \cup U_2' + w_2) \\ &= c(T' \cup U_2 + w_2) - |U_2 - U_2'| \left(\frac{k}{2} - \left(\frac{k}{2} - 1 + \epsilon \right) \right) \\ &\geqslant c(T' \cup U_2 + w_2) - \left(\frac{k-2}{4(k-1)} K - |T'| + 1 \right)(1 - \epsilon) \,. \end{aligned} \tag{96}$$

Let us now assume $t_1 \in T'$. In this case, the smallest T'-cut containing w_1 but not w_2 is $T' \cup U_1 + w_1$ since every node in U_1

is connected to $t_1 \in T'$. The smallest T'-cut containing w_2 but not w_1 is $T' \cup U_2' + w_2$ where $U_2' \subseteq U_2$ is the set of nodes that are connected to at least one node in T'. We compare their capacities:

$$\begin{aligned}
&c(T' \cup U_2' + w_2) - c(T' \cup U_1 + w_1) \\
&= c(T' \cup U_2 + w_2) - |U_2 - U_2'|(1-\epsilon) - c(T' \cup U_1 + w_1) \\
&\geqslant c(T' \cup U_2 + w_2) \\
&\quad - \left(\frac{k-2}{4(k-1)}K - |T'| + 1\right)(1-\epsilon) - c(T' \cup U_1 + w_1) \\
&= \frac{k-|T'|}{k-1}K - \left(\frac{k-2}{4(k-1)}K - |T'| + 1\right)(1-\epsilon) \\
&\geqslant \frac{k-|T'|}{k-1}K - \left(\frac{k-2}{4(k-1)}K - |T'| + 1\right) \\
&= \frac{3k-4|T'|+2}{4(k-1)}K + |T'| - 1 \qquad (97)
\end{aligned}$$

Since this difference is positive for all considered T', $T' \cup U_1 + w_1$ is the smallest cut different from T'. So, we now compare the capacities of T' and $T' \cup U_1 + w_1$. The capacity of the T'-cut $S = T' \cup U_1 + w_1$ is

$$\begin{aligned}
&c\big(T' \cup U_1 + w_1\big) \\
&= c_{t_1}(\overline{T'}, U_1) + c_{t_1}(T', U_2) + c(w_1, w_2) \\
&= \frac{k-2|T'|}{4}K + 2c_{t_1}(T', U_2) + c(w_1, w_2) \\
&= \frac{k-2|T'|}{4}K + \frac{2k(|T'|-1)}{4(k-1)}K + \frac{k}{4(k-1)}K - 1 + \epsilon \\
&= \frac{k^2-2k+2|T'|}{4(k-1)}K - 1 + \epsilon\,. \qquad (98)
\end{aligned}$$

We compare this capacity to the capacity of the T'-cut T':

$$\begin{aligned}
&c(T' \cup U_1 + w_1) - c(T') \\
&= \frac{k^2-2k+2|T'|}{4(k-1)}K - 1 + \epsilon - \frac{|T'|}{2}K \\
&= \frac{k^2-2k+2|T'|-2k|T'|+2|T'|}{4(k-1)}K - 1 + \epsilon \\
&\geqslant \frac{k^2-2k(\frac{k}{2})+4(\frac{k}{2}-1)}{4(k-1)}K - 1 + \epsilon \\
&= \frac{\frac{k}{2}-1}{k-1}K - 1 + \epsilon \;>\; 0 \qquad (99)
\end{aligned}$$

So in case $|T'| < k/2$ and $t_1 \in T'$, the unique T'-min-cut is T'. Now, we assume $t_1 \notin T'$. The smallest cut containing w_1 but not w_2 is $T' \cup U_1' + w_1$ where $U_1' \subseteq U_1$ is the set of nodes that are connected to at least one node in U_1. The smallest cut containing w_2 but not w_1 is $T' \cup U_2 + w_2$. We compare their capacities:

$$\begin{aligned}
&c(T' \cup U_1' \cup + w_1) - c(T' \cup U_2 + w_2)\\
&= c(T' \cup U_1 + w_1) - |U_1 - U_1'|(1-\epsilon) - c(T' \cup U_2 + w_2)\\
&= \frac{|T'|}{k-1}K - |U_1 - U_1'|(1-\epsilon)\\
&\geqslant \frac{|T'|}{k-1}K - \left(\frac{k}{4(k-1)}K - |T'| + 1\right)(1-\epsilon)\\
&\geqslant \frac{|T'|}{k-1}K - \left(\frac{k}{4(k-1)}K - |T'| + 1\right)\\
&= \frac{4|T'| - k}{4(k-1)}K + |T'| - 1 \qquad (100)
\end{aligned}$$

This difference is positive for all considered T'. Hence, we have to compare the capacities of $T' \cup U_2 + w_2$ with T'. For this, we first determine the capacity of $T' \cup U_2 + w_2$:

$$\begin{aligned}
&c(T' \cup U_2 + w_2)\\
&= c_{\overline{t_1}}(T', U_1) + c_{\overline{t_1}}(\overline{T'}, U_2) + c(w_1, w_2)\\
&= \frac{|T'|}{k-1} \cdot \frac{k-2}{4} \cdot K + \left(\frac{k}{4}K - c_{\overline{t_1}}(T', U_2)\right) + c(w_1, w_2)\\
&= \frac{k|T'| - 2|T'| + k^2 - k - k|T'|}{4(k-1)}K + \frac{k}{4(k-1)}K - 1 + \epsilon\\
&= \frac{k^2 - 2|T'|}{4(k-1)}K - 1 + \epsilon \qquad (101)
\end{aligned}$$

The difference of the capacity of the two cuts amounts to

$$\begin{aligned}
&c(T' \cup U_2 + w_2) - c(T')\\
&= \frac{k^2 - 2|T'|}{4(k-1)}K - 1 + \epsilon - \frac{|T'|}{2}K\\
&= \frac{k^2 - 2k|T'|}{4(k-1)}K - 1 + \epsilon\,. \qquad (102)
\end{aligned}$$

Since $|T'| < k/2$, this is always positive.

d) Now, we consider all T'-cuts S with $W \subset S$. It pays off to move a node $u_i \in U$ to S, if

$$\begin{aligned}
& c(T', \{u_i\}) + c(\{u_i\}, C) \geqslant c(\overline{T'}, \{u_i\})\\
\Longleftrightarrow\quad & c(T', \{u_i\}) \geqslant c(\overline{T'}, \{u_i\}) - \left(\frac{k}{2} - 1 + \epsilon\right)\\
\Longleftrightarrow\quad & c(T', \{u_i\}) \geqslant \frac{k}{2} - c(T', \{u_i\}) - \frac{k}{2} + 1 - \epsilon\\
\Longleftrightarrow\quad & c(T', \{u_i\}) \geqslant \frac{1}{2}(1-\epsilon)\,. \qquad (103)
\end{aligned}$$

So again, it pays off to move all $u_i \in U$ to S that are connected to at least one node in T'. We denote the smallest such cut with $S = T' \cup U' \cup W$. The number of nodes in U' can be bounded by

$$\begin{aligned}
|U'| &= \left|\{u_i \in U \mid c(\{u_i\}, T') = 0\}\right|\\
&\leqslant |U| - \frac{k}{2}K - (|T'| - 1) \;=\; \frac{2-k}{2}K - |T'| + 1\,. \qquad (104)
\end{aligned}$$

Hence, the capacity of $T' \cup U' \cup W$ is

$$\begin{aligned}
&c(T' \cup U' \cup W) \\
&= c(T', \overline{U'}) + c(\overline{T'}, U') + c(\overline{U'}, W) \\
&= c(\overline{T'}, U') + c(\overline{U'}, W) \\
&\geqslant |U - U'| \left(\frac{k}{2} - 1 + \epsilon\right) \\
&= K - \left(\frac{2-k}{2}K - |T'| + 1\right)\left(\frac{k}{2} - 1 + \epsilon\right) \\
&= \left(\frac{k}{2}K + |T'| - 1\right)\left(\frac{k}{2} - 1 + \epsilon\right) \\
&\geqslant \left(\frac{k}{2}K + |T'| - 1\right)\left(\frac{k}{2} - 1\right) \\
&= \frac{k^2 - 2k}{4}K + \frac{k-2}{2}|T'| - \frac{k}{2} + 1\,. \qquad (105)
\end{aligned}$$

We compare this capacity to the capacity of T':

$$\begin{aligned}
&c(T' \cup U' \cup W) - c(T') \\
&\geqslant \frac{k^2 - 2k}{4}K + \frac{k-2}{2}|T'| - \frac{k}{2} + 1 - \frac{|T'|}{2}K \\
&= \frac{k^2 - 2k - 2|T'|}{4}K + \frac{k-2}{2}|T'| - \frac{k}{2} + 1 \qquad (106)
\end{aligned}$$

Since this difference is positive for all T' considered, we conclude that for any T' with $|T'| < k/2$, the unique T'-min-cut is T'.

2. Now, let us consider all $T' \subset \mathcal{T}$ with $|T'| = \frac{k}{2}$ and $t_1 \in T'$.

   a) First, we consider all T'-cuts S with $S \cap W = \emptyset$. We have

   $$c(T') = \frac{k}{4}K \quad \text{and} \quad c(T' + u_{T'}) = \frac{k}{4}K - 1 + \epsilon\,. \qquad (107)$$

   If any non-terminal $u_i \neq u_{T'}$ is added to S, the cut capacity is increased since this node is connected to nodes in $\mathcal{T} - T'$.

   b) We now consider all T'-cuts with $|S \cap W| = 1$. Since $t_1 \in T'$, the smallest such cut is $T' \cup U_1 + w_1$, since term (97) is positive for all T' with $t_1 \in T_1$ and $|T'| = k/2$. The capacity of this cut is

   $$c\big(T' \cup U_1 + w_1\big) = \frac{k}{4}K - 1 + \epsilon\,. \qquad (108)$$

   Observe, that $c(T' + u_{T'}) = c(T' \cup U_1 + w_1)$.

   c) Finally, consider all S with $W \cap S = W$. It pays off to move all nodes $u_i \in U$ to S that are connected to at least one node in T'. There is exactly one node in S that is not connected to any node in T'. This node is $u_{\overline{T'}}$. Hence, the capacity of this cut is

$$\begin{aligned} & c(T' \cup (U - u_{\overline{T'}}) \cup W) \\ &= c(T', \{u_{\overline{T'}}\}) + c(\overline{T'}, U - u_{\overline{T'}}) + c(\{u_{\overline{T'}}\}, W) \\ &= 0 + \frac{k}{4} \cdot K - \frac{k}{2} + \frac{k}{2} - 1 + \epsilon \\ &= \frac{k}{4} K - 1 + \epsilon \,. \end{aligned} \tag{109}$$

We conclude that the set of all T'-min-cuts for all considered T' with $t_1 \in T'$ is $\{T' + u_{T'}, T' \cup U_1 + w_1, T' \cup (U - u_{\overline{T'}}) \cup W\}$.

Thus, we have shown that the set of T'-min-cuts is

$$\begin{aligned} \left\{ T' \right\} \quad & \text{if } |T'| < \frac{k}{2} \text{ and} \\ \left\{ T' + u_{T'}, T' \cup U_1 + w_1, T' \cup (U - u_{\overline{T'}}) \cup W \right\} \quad & \text{if } |T'| = \frac{k}{2}, t_1 \in T'. \end{aligned}$$

Thus, nodes w_1 and w_2 can be merged to obtain a contraction-based mimicking network for the original network, since for every $T' \subset \mathcal{T}$, there is a T'-min-cut that does not separate w_1 and w_2. After merging w_1 and w_2, only min-cuts are left that do not separate w_1 and w_2. The best choice of min-cuts among these yields an irreducible CBMN of size $k + 2 + K/2$.

Alternatively, we can merge all nodes in $U_1 + w_1$ and all nodes in $U_2 + w_2$ to obtain a minimum contraction-based mimicking network of size $k + 2$. □

The network family $\mathcal{N}$ also provides some insight how much little perturbations of the capacities can impact the size of a minimum CBMN. If the capacity of (w_1, w_2) is slightly lower, then the minimum CBMN is unique and of the form of N_4'' as seen in Figure 17 with $k + 2 + K/2$ non-terminals. If, however, the capacity of (w_1, w_2) is slightly increased, then the minimum CBMN has only 2 non-terminals. Thus, if two networks are identical as graphs and only differ in the vector of edge capacities, then with respect to any L_p-metric measuring their distance, the distance can be made arbitrarily small, yet the number of required non-terminals jumps from constant to exponential in k. Thus, we can conclude:

▷ COROLLARY 57: There exist pairs of k-terminal networks N_k^+ and N_k^- with the same topology and arbitrarily close capacities such that the minimum CBMN of the first one is of size $k + 2$ and exponential in k for the other.

Therefore, the technique of small perturbations to make min-cuts unique does not seem appropriate to find a minimum CBMN. This construction also implies that there can be a huge gap in the minimum size for contraction-based cut sparsifiers if one changes the approximation quality slightly as the next corollary states.

▷ COROLLARY 58: There exist k-terminal networks and a cut point $\alpha > 1$ such that minimum contraction-based cut sparsifiers with quality less than α have to be of exponential size, while a quality larger than α can be achieved by linear size contraction-based sparsifiers.

Proof. The same family of networks as in Theorem 56 is used with a slight variation. Let $\delta = 1/4 \cdot (\alpha - 1) \cdot (k \cdot K + 4\epsilon - 4)$. In network N_k, we now raise the capacity of the edge (w_1, w_2) by δ such that $c(w_1, w_2) = k \cdot (4(k - 1))K - 1 + \epsilon + \delta$.

The value of δ has been picked such that when choosing the min-cuts separating w_1 from w_2 (compare to the proof of Theorem 56) leading to a CBMN of size $k+2$, the quality is exactly α. Hence, a contraction-based cut sparsifier of quality less than α does not allow choosing any min-cuts that separate w_1 from w_2 and consequently, the smallest such contraction-based cut sparsifier is of size exponential in k. □

As another interpretation, we may see Theorem 56 as an indication that drastically smaller CBMNs may be found if we do not restrict ourselves to finding mimicking networks that allow the exact same set of flow patterns, but only approximate it with cut sparsifiers.

The results so far indicate that an optimal strategy for merging nodes will not be easy to find if one has to choose among several min-cuts. Here, we give a partial result saying that merging certain nodes is always a good strategy.

▷ THEOREM 59: Let N be a network and $v \in V$ any node. Further, let S_v be the source side of the v-min-cut ($v \in S$) with maximum $|S_v|$. Then, merging all nodes in S_v yields a CBMN N' whose minimum CBMN is also a minimum CBMN for N.

Proof. Let γ be a choice of min-cuts for a minimum CBMN for N and $B \in \mathcal{B}$ with $\emptyset \neq \gamma(B) \cap S_v \neq S_v$. In other words, $\gamma(B)$ is a chosen min-cut separating nodes within S_v. Let us say $\gamma(B) = S_B$ and $V - S_B = T_B$. Lemma 11 on page 35 tells us that if $v \in S_B$, then $S_B \cup S_v$ is a B-min-cut, and if $v \in T_B$, then $S_B - S_v$ is a B-min-cut. This allows us to adjust the equivalence relation R_γ by moving all nodes that are in S_v, but not in the equivalence class of v to that class. This way, no new classes are created and for each terminal bipartition there is a min-cut that does not cut any nodes in S_v as just shown. □

Intuitively, for the last theorem we reroute all min-cuts going through S_v, and this rerouting takes place locally. Note the similarity to Algorithm 1 on page 35. However, if S_v is a v-min-cut, we do not merge the nodes of S_v if $|S_v|$ is minimum, but if $|S_v|$ is maximum. This does not maintain all min-cuts as needed in Lemma 12, but suffices to ensure that we do not miss the minimum CBMN. Here, too, computing for each node v the v-min-cut with the largest source side and then merging all nodes of that side can be done in polynomial time and always yields a minor of the original graph. After this step, no terminal can be merged with any other nodes anymore since for every terminal t_i, there is exactly one t_i-min-cut – $\{t_i\}$. In general, however, non-terminals may be left that can be merged with each other. The next lemma states that if they can all be merged to a single non-terminal, then this can be detected by examining min-cuts (S, T) with maximum and minimum $|S|$ only.

▷ LEMMA 60: Let N be a network with non-terminals where no non-terminal can be merged with any terminals. Then, the size of a minimum CBMN is $|\mathcal{T}| + 1$ if, and only if, there exists a choice of extremal min-cuts where no chosen min-cut separates any non-terminals.

Proof. Since no non-terminal can be merged with any terminal, $|\mathcal{T}| + 1$ is a lower bound for the size of a minimum CBMN for N. If $|\mathcal{T}| + 1$ is the size of a minimum CBMN, then there is a choice of min-cuts such that none of the non-terminals are separated from each other. Each of the chosen cuts is an extremal min-cut as it contains terminals only on one side. For the other direction, if there is a choice of extremal min-cuts not separating any non-terminals, then all non-terminals can be merged to a single non-terminal. □

Note that Theorem 59 allows to find the minimum CBMN of network N in Figure 16 (page 75) in polynomial time. The next Theorem further helps to efficiently reduce the size of the CBMN still maintaining the possibility to reach a minimum CBMN.

▷ THEOREM 61: Let u, v be two non-terminals of a network N such that for all nodes $w \neq u, v$ we have $c(u, w) \geqslant c(v, w)$. Then, merging u, v yields a CBMN N' of N that can still be reduced to a minimum CBMN for N.

Proof. Assume there is a terminal bipartition for which a min-cut $(S, V - S)$ separates u from v with $u \in S$. The edges adjacent to u or v are categorized in the four disjoint sets

- $A = \{(u, x) \mid u, x \in S\}$,
- $B = \{(u, x) \mid u \in S, x \notin S\}$,
- $C = \{(v, x) \mid v \notin S, x \in S\}$,
- $D = \{(v, x) \mid v, x \notin S\}$, and
- $E = \{(u, v)\}$.

If X is a set of edges, we denote the sum of the edge capacities of the edges in X by $c(X)$. We will now compare $c(S)$ to $c(S + v)$. We have $c(S) - c(S + v) = c(E) + c(B) - c(D)$. Since $c(u, w) \geqslant c(v, w)$ for all $w \neq u, v$, it follows that $c(B) \geqslant c(D)$ and hence $c(S) - c(S + v) \geqslant 0$. This implies, that $S + v$ is a min-cut as well that does not separate u and v. Similar to the proof of Theorem 59, assume that γ is a choice of min-cuts leading to a minimum CBMN. If a chosen min-cut separates u, v, we may locally alter the chosen min-cut in the just described manner to obtain a new optimal choice of min-cuts. □

While finding and merging nodes as described in Theorem 61 can again be done in polynomial time, this time it is not guaranteed to yield a CBMN that is a minor of the original graph. In general, there are networks N whose minimum CBMN is not a minor of N.

Applying Theorem 59 and Theorem 61 results in a smaller CBMN that in general still has many non-terminals left to be merged. To find out which ones in order to reach a minimum CBMN is still unknown and further examined in the next two sections.

4.3 THE COMPLEXITY OF FINDING SMALL MIMICKING NETWORKS

This section investigates the complexity of various problems related to mimicking networks. For some of these problems, we present completeness results, for others we obtain partial results in form of lower and upper bounds. Lower bounds have not yet been considered at all in the literature.

The central problem that motivates much of this section is to decide whether for a given network N a CBMN of a given size m exists. Formally, this problem is stated as

▷ PROBLEM 28 (MINIMUM CONTRACTION-BASED MIMICKING NETWORK (MCBMN)):

Input: A tuple (N, m) of an undirected multi-terminal network N and a natural number m.

Question: Does N have a CBMN of size m?

We are also interested in the question of how much the complexities of this and related problems change in presence and absence of the unique min-cuts assumption. Khan and Raghavendra [52] present an algorithm that finds minimum CBMNs for networks with unique min-cuts in time polynomial in n and 2^k where n is the size of the input network and k the number of terminals. This restricted problem is therefore fixed-parameter tractable when parameterized by the number of terminals. In order to analyze the complexity of the problem to find a minimum CBMN under the unique min-cuts assumption, we introduce the problem

▷ PROBLEM 29 (MCBMN-U):

Input: A tuple (N, m) of an undirected multi-terminal network N and a natural number m.

Question: Are all min-cuts of N unique and does N have a CBMN of size m?

A lower bound for the complexity of this problem can be obtained by reducing the well-known PARTITION problem, whose notation is as follows.

▷ PROBLEM 30 (PARTITION):

Input: A tuple (A, s) of a finite set $A = \{1, 2, \ldots, n\}$ with $n \geqslant 1$ and a weight $s(i) \in \mathbb{Z}^+$ for each $i \in A$.

Question: Is there a subset $A' \subseteq A$ such that $\sum_{i \in A'} s(i) = \sum_{i \notin A'} s(i)$?

▷ LEMMA 62: MCBMN-U is coNP-complete.

Proof. We show that the complement of MCBMN-U is in NP. If there is a terminal bipartition with more than one min-cut, then the machine guesses this bipartition and verifies the existence of multiple min-cuts belonging to it. This can be done by finding the two extremal min-cuts and verifying that they are different. To see that there is no contraction-based mimicking network of size m, the machine guesses $m+1$ nodes and for each pair of these $m+1$ nodes a terminal bipartition. Then, it is verified that for each pair u, v among the guessed $m+1$ nodes, the unique min-cut of the guessed terminal bipartition separates u and v – hence making it impossible to merge any nodes of the $m+1$ guessed ones. Concluding, there is no CBMN of size m.

For hardness, we describe a log-space reduction from PARTITION to the complement of MCBMN-U. Given a PARTITION instance (A, s) with $|A| = n$, create a bipartite network N with terminals $\mathcal{T} = \{t_1, t_2, \ldots, t_{n+2}\}$ on one side and two non-terminals u, v on the other – a total of $n+4$ nodes. For $i \in \{1, \ldots, n\}$ there are edges connecting t_i with u and with v of capacities $c(t_i, u) = c(t_i, v) = 4 \cdot s(a_i)$. Moreover, $c(t_{n+1}, u) = 1$ and $c(t_{n+2}, v) = 2$. These edges make up the edge set E of the constructed network. Every edge now has a positive whole number as capacity. The MCBMN instance is then defined as $(N, n+3)$ – the question whether there is a pair of nodes that can be merged.

In the following, we analyze the capacities of cuts. For every $T \subseteq \mathcal{T}$, let $A(T) = \{ a_i \mid t_i \in T, i \in \{1, \ldots, n\} \}$. Further, let $Z = \sum_{a_i \in A} s(a_i)$ and $Z_T = \sum_{a_i \in A(T)} s(a_i)$. For every non-trivial terminal subset $T \subset \mathcal{T}$, there are four candidates for being a T-min-cut. Due to symmetry, we may assume that $Z_T \leqslant Z/2$. Their capacities are

$$\begin{aligned} c(T) &= 8 \cdot Z_T + f(T \cap \{t_{n+1}, t_{n+2}\}, \emptyset), && (110) \\ c(T \cup \{u\}) &= 4 \cdot Z + f(T \cap \{t_{n+1}, t_{n+2}\}, \{u\}), && (111) \\ c(T \cup \{v\}) &= 4 \cdot Z + f(T \cap \{t_{n+1}, t_{n+2}\}, \{v\}), \text{ and} && (112) \\ c(T \cup \{u, v\}) &= 8 \cdot Z - 8 \cdot Z_T + f(T \cap \{t_{n+1}, t_{n+2}\}, \{u, v\}), && (113) \end{aligned}$$

where $f : 2^{\{t_{n+1}, t_{n+2}\}} \times 2^{\{u,v\}} \to \{0, 1, 2, 3\}$ is a function that adds an according value to the cut capacity in dependence of whether the edges (t_{n+1}, u) or (t_{n+2}, v) are cut. It is defined as $f(A, B) = \sum_{e \in E_{A,B}} c(e)$ where

$$\begin{aligned} E_{A,B} = \{ & e \in \{t_{n+1}, t_{n+2}\} \times \{u, v\} \mid \\ & \text{exactly one endpoint of } e \text{ is in } A \cup B\}. && (114) \end{aligned}$$

If $Z_T < Z/2$, then the value of f does not influence which cut is the min-cut – T is the unique T-min-cut which does not separate u and v. If $Z_T = Z/2$, then the cut with the lowest value for function f is the unique min-cut. If $t_1 \in T$, but $t_2 \notin T$, then this min-cut is $T \cup \{u\}$ and separates u and v.

Note that independent of whether $(A, s) \in$ PARTITION, all min-cuts in the network created by the reduction function are unique. If there is no PARTITION solution to (A, s), then $Z_T < Z/2$ for all considered $T \subset \mathcal{T}$. If A' is a PARTITION solution to (A, s), and if T' is a set such that $A(T') = A'$, then $2 \cdot Z_{T'} = Z = Z - 2 \cdot K_T$. Hence, it depends on the value of the term $c(T \cap \{t_{n+1}, t_{n+2}\}, \{u, v\})$ which cut capacity is minimum.

We are now ready to show the correctness of the reduction. So assume $(A, s) \in$ PARTITION and let $A' \subset A$ be a solution. Now, consider the terminal subset $T = \{t_i \mid a_i \in A'\} \cup \{t_{n+1}\}$ and let $\overline{T} = \mathcal{T} - T$ be the remaining terminals. There are 4 potential candidates for the T-min-cut, namely T, $T \cup \{u\}$, $T \cup \{v\}$, and $T \cup \{u, v\}$ with capacities

$$\begin{aligned} c(T) &= Z + 1, && (115) \\ c(T \cup \{u\}) &= Z, && (116) \\ c(T \cup \{v\}) &= Z + 2, \text{ and} && (117) \\ c(T \cup \{u, v\}) &= Z + 1. && (118) \end{aligned}$$

Hence, $T \cup \{u\}$ is the source side of the unique T-min-cut which separates u and v. This forbids to merge u and v. It remains to be shown that no other nodes can be merged – this could be pairs consisting of a terminal and a non-terminal. As there is a solution to the PARTITION instance, we know that each single element of A that is not a solution has a weight of truly less than $Z/2$. This implies for each $i \in \{1, \ldots, n\}$ and $x \in \{u, v\}$ that $c(t_i, x) < c(x, \mathcal{T} - \{t_i\})$. Hence, the unique t_i-min-cut is $\{t_i\}$. We deduce that $(N, n+3) \notin$ MCBMN.

If $(N, n+3) \notin$ MCBMN, then there is a $T' \subset \mathcal{T}$ such that all T'- min-cuts separate u, v. T' contains exactly one node of $\{t_{n+1}, t_{n+2}\}$, since otherwise the situation for u, v is completely symmetric and u, v are not separated. This implies $c\big(\{u\}, T' - \{t_{n+1}, t_{n+2}\}\big) = c\big(\{v\}, T' - \{t_{n+1}, t_{n+2}\}\big)$. The set A' that corresponds to $T' - \{t_{n+1}, t_{n+2}\}$ is a solution to the original PARTITION instance. □

The idea behind the just described reduction is illustrated in Figure 18. Multiplying the edge capacities with factor 4 and using different edge capacities for (t_{n+1}, u) and (t_{n+2}, v) might seem a little tedious. In fact, there is

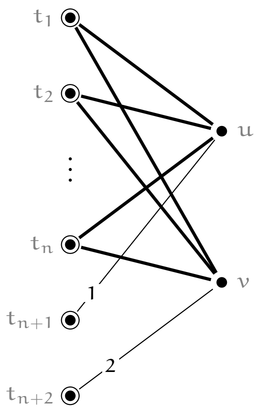


Figure 18: This figure illustrates the reduction from a PARTITION instance (A, s) to the MCBMN-U instance described in Lemma 62. Every thick edge incident to t_i has capacity $4 \cdot s(a_i)$. Additionally, there are the two edges between t_{n+1} and u and between t_{n+2} and v of capacity 1 and 2, respectively. These capacities are so low in comparison to the capacities of the thick edges such that a T'-min-cut splits u and v if, and only if, the weights of the elements in A corresponding to the terminals in $T' \subset \mathcal{T}$ add up to half of the total weight.

an easier reduction to show coNP-completeness for MCBMN-U. However, we wanted to emphasize that the success of the reduction does not depend on creating multiple min-cuts at some point but solely on whether two nodes can be merged or not. This way the hardness result can immediately be applied to MCBMN as well.

▷ COROLLARY 63: MCBMN is coNP-hard.

This corollary naturally raises the question whether MCBMN is in coNP as well. Unfortunately, the technique of guessing $m + 1$ nodes that cannot be merged pairwise does not work – it fails due to the existence of multiple min-cuts. An example of such a situation can be seen in Figure 19.

The polynomial hierarchy introduced by Stockmeyer [77] seems promising to provide a fitting complexity class for MCBMN. We are able to prove membership for MCBMN in Σ_2^P.

▷ LEMMA 64: MCBMN $\in \Sigma_2^P$.

Proof. An $\mathrm{NP}^{\mathrm{coNP}}$ machine repeatedly guesses a pair of nodes of the given network N and merges them until m nodes are left. It remains to be verified that the network N' obtained this way is a mimicking network of N. This can be done by checking universally for all possible terminal bipartitions that the min-cut capacities are identical. □

For MCBMN it is unclear whether this upper bound can be improved or whether the lower bound of Corollary 63 can be lifted up. Another possibility is that the complexity of MCBMN is best captured by Δ_2^P. The precise complexity of MCBMN remains open. The membership part of the proof of Lemma 62 does not work anymore if min-cuts do not have to be unique. Matching this fact, MCBMN $\in$ coNP seems unlikely given that the proof of Lemma 62 works also for the more restricted problem asking whether there is a pair of nodes that can be merged in a bipartite network – with *unique* min-cuts.

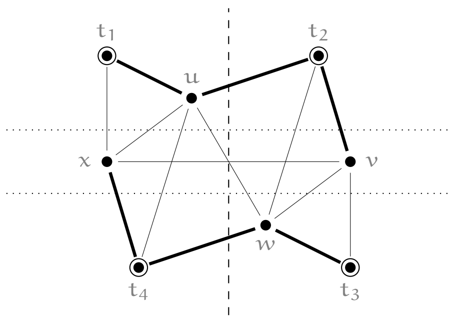


Figure 19: Every thin edge has capacity 1, every thick edge capacity 2. All $\{t_1, t_2\}$-min-cuts are drawn as dotted lines, all $\{t_1, t_4\}$-min-cuts as dashed lines. All other min-cuts do not separate any non-terminals and all t_i-min-cuts for $t_i \in \mathcal{T}$ make it impossible to merge any nodes with a terminal. Merging (u, x) as well as merging (v, w) yields a CBMN. However, merging both pairs does not. Therefore, there are two different minimum CBMNs of size 7. It can be observed that for every set of 7 nodes, there is a pair of nodes that can be merged. This shows that the strategy used in the proof of Lemma 62 does not work to show MCBMN $\in$ coNP.

The problem to compute the minimum size of a CBMN for a given network N corresponds to a language that is the intersection of a language in Σ_2^P and one in Π_2^P. To actually construct a minimum CBMN a machine in Δ_3^P suffices. Note that for MCBMN-U the complexity drops down to Δ_2^P.

In order to show Σ_2^P-completeness, a reduction from a Σ_2^P-hard problem (for a compendium of such problems see [72]) is needed that constructs an MCBMN instance making smart use of the possibility to have several min-cuts for each terminal bipartition. The problem of constructing such networks is the lack of understanding which combinations of min-cuts are possible. It is the same lack of understanding that is reflected in the exponential gap between lower and upper bounds in the size for minimum CBMNs and the same lack of understanding which set systems correspond to a multi-terminal network (see Chapter 3). The correlation between these questions is also expressed by noting that MCBMN is equivalent to MIN-CUT MINIMUM TYPE SELECTION defined on page 43 in the previous chapter.

▷ THEOREM 65: MCBMN = MIN-CUT MINIMUM TYPE SELECTION.

Proof. $(N, m) \in$ MCBMN is equivalent to the fact that there is a choice of min-cuts γ such that R_γ contains at most m equivalence classes. When representing min-cuts by min-cut vectors as done for MIN-CUT MINIMUM TYPE SELECTION, then the columns of the matrix composed row-wise of the min-cut vectors belonging to chosen min-cuts correspond to nodes. Two columns are identical if, and only if, they correspond to nodes that are not separated by any chosen min-cut, i. e., two columns are identical if, and only if, the corresponding nodes belong to the same equivalence class. In total, there is a choice of min-cuts for N such that the according matrix contains at most k distinct columns if, and only if, there is a CBMN for N of size k. □

Let us now switch to the parameterized world and analyze the fixed-parameter tractability of these problems. For this, p-MCBMN and p-MCBMN-U shall be the parameterized variants of MCBMN and MCBMN-U, respectively, where the parameter is the number of terminals. The algorithm presented by Khan and Raghavendra [52] shows that p-MCBMN is fixed-parameter tractable when limited to input networks with unique min-cuts. The status of p-MCBMN remains unclear. Theorem 59 and Theorem 61 do not suffice to create a kernel. Interestingly, the following problem remains fixed-parameter tractable even if the unique min-cuts condition is dropped.

▷ PROBLEM 31 (p-NODES-MERGE):

Input: A tuple (N, ℓ) of an undirected multi-terminal network N and a natural number $\ell \geqslant 2$.

Parameter: The number of terminals k in N.

Question: Is there a set of ℓ nodes in N that can be merged to a single node to obtain a mimicking network of N?

If all min-cuts are known to be unique, then the algorithm from Khan and Raghavendra [52] proves the problem to be in FPT. In our more general setting, however, a different strategy is needed which we describe next.

▷ THEOREM 66: p-NODES-MERGE $\in$ FPT.

Proof. For all $B \in \mathcal{B}$, compute the two extremal B-min-cuts. Then, iterate over all resulting $2^{2^{k-1}-1}$ choices of min-cuts and accept if, and only if, there is an equivalence class of size ℓ for some choice of min-cuts.

The two extremal min-cuts are unique for each terminal-bipartition due to the submodularity of min-cuts and can be determined efficiently by standard methods.

Certainly, this algorithm never accepts instances in the complement of p-NODES-MERGE. For the other direction, let $(N, \ell) \in$ p-NODES-MERGE with a set L of ℓ nodes that can be merged, but the algorithm rejects. This implies that there is a terminal bipartition, for which both extremal min-cuts separate nodes within L. If so, however, then all min-cuts separate nodes within L – a contradiction. □

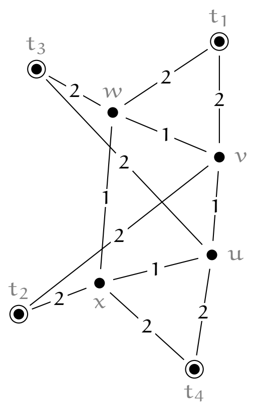
The last result sheds some light on the structure of these networks. For a network N, let $M(N)$ be the *merge graph* of N that is defined over the same set of nodes. Two nodes u, v are joined by an edge if, and only if, they can be merged to obtain a mimicking network of N. Knowledge of the merge graph does not allow to reconstruct the network since information is missing. In particular, merging two nodes may or may not remove other edges of the merge graph. However, all nodes of a clique can always be merged to a single node as is now next.

▷ LEMMA 67: A set L of nodes in a network N can be merged to a single node if, and only if, L is a clique in $M(N)$.

Proof. If all nodes of L can be merged, then there is a choice of min-cuts such that there is an equivalence class containing L. Since all nodes of an equivalence class can be merged pairwise, L is a clique in $M(N)$.

Assume there is a clique L in $M(N)$ and pick an arbitrary terminal bipartition B. If the B-min-cut with smallest source side contains a node $u \in L$ and if there is a node $v \in L$ not contained in the source side of the min-cut with the largest source side, then every B-min-cut separates u, v. Since there is an edge (u, v) in $M(N)$, this is a contradiction. Therefore, the min-cut with the smallest source side does not contain any nodes of L or the min-cut with the largest source side contains all nodes of L in the source side. In either case, all nodes of L can be merged. □

When considering the set of merge graphs for all networks with a fixed number of terminals, then Theorem 66 together with Lemma 67 imply that



All T-min-cuts for	
$T = \{t_1, t_2\}$	$\{t_1, t_2, v\}$
	$\{t_1, t_2, v, w\}$
	$\{t_1, t_2, v, w, x\}$
$T = \{t_1, t_3\}$	$\{t_1, t_3, w\}$
	$\{t_1, t_3, v, w\}$
	$\{t_1, t_3, u, v, w\}$

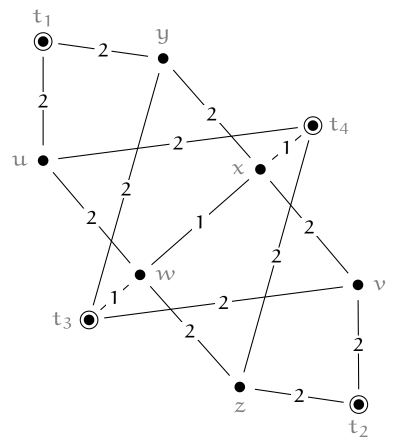
Figure 20: The displayed network is an example of a case where no choice of extremal min-cuts leads to a minimum CBMN. All terminal bipartitions not listed above do not have any min-cuts separating any non-terminals. Hence, we only need to consider the listed min-cuts. The only choice of these min-cuts that leads directly to the minimum CBMN is the two non-extremal ones. In this case, v and w as well as u and x can be merged. Other choices may lead to a local minimum. For example, there is a choice of min-cuts that allows to merge u and w – but from then on, no more nodes can be merged. There are, however, choices of extremal min-cuts that neither lead to the minimum CBMN directly, nor to a local minimum. See Figure 21 for an example where every choice of extremal min-cuts leads to a local minimum.

cliques of any size can efficiently be detected in these graphs. Under standard complexity theoretic assumptions, the problem of finding a clique of size ℓ in an arbitrary graph is fixed-parameter *intractable* when parameterized over ℓ as it is $W[1]$-complete [23]. The merge graph of such networks is thus subject to structural restrictions that make the problem considerably easier. It is unclear whether these restrictions allow an FPT algorithm for p-MCBMN as well. Unfortunately, the proof idea of Theorem 66 does not work for p-MCBMN as there are networks where the choice of min-cuts needs to include a min-cut that is not extremal, see Figure 20 and Figure 21 for such networks.

4.4 AN ALGORITHM FOR THE 4-TERMINAL CASE

A necessary condition for p-MCBMN to be in FPT is that every slice is in P, i. e., every language consisting of all problem instances with equal parameter can be solved in polynomial time. To see if this necessary condition is met, we analyze the problem for up to 4 terminals. If this problem can be solved in polynomial time, then p-MCBMN possibly is in FPT. But if it is not, then this possibility is ruled out.

This section describes an algorithm for p-MCBMN restricted to up to 4 terminals which is a surprisingly challenging task. While the presented algorithm runs in polynomial time, it can only guarantee to find contraction-based mimicking networks at most two nodes greater than the minimum one. In the following, we will go through all possible cases for minimum CBMNs to networks with at most 4 terminals. The cases that cannot be decided correctly by the algorithm in polynomial time are promising candidates to proof NP-hardness of this slice of p-MCBMN.



All T-min-cuts for	
$T = \{t_1, t_3\}$	$\{t_1, t_3, y\}$
	$\{t_1, t_3, v, x, y\}$
	$\{t_1, t_3, u, v, w, x, y\}$
$T = \{t_2, t_3\}$	$\{t_2, t_3, v\}$
	$\{t_2, t_3, v, x, y\}$
	$\{t_2, t_3, v, w, x, y, z\}$

Figure 21: In this example, any choice of extremal min-cuts leads to a local minimum when trying to find a minimum CBMN. Again, all terminal bipartitions not listed do not have any min-cuts separating non-terminals and can hence be ignored. Note that w and x are not separated by any extremal min-cut. Merging them, however, yields a local minimum.

For up to three terminals, every terminal bipartition separates a single terminal from the remaining terminals. Theorem 59 (page 86) allows us to pick the best min-cut for such a terminal bipartition in polynomial time and hence a minimum CBMN can be found in polynomial time for any network with up to 3 terminals.

We refer to the problem p-MCBMN with exactly 4 terminals to 4-MCBMN. The number of terminal bipartitions that needs to be considered in a 4-terminal network grows to 7. Four of these are terminal bipartitions isolating a single terminal. For these, Theorem 59 can still be applied allowing us to merge a maximum number of non-terminals with terminals. Hence, if there is a CBMN with terminals only, then we can find it in polynomial time – and certainly it is a minimum CBMN. The question $(N, 4) \in$ MCBMN can thus be answered in polynomial time.

If not all non-terminals can be merged with terminals, then we would like to know whether they can all be merged with each other to obtain a CBMN of size 5. This is the case if, and only if, we can find for each of the three remaining terminal bipartitions a min-cut that does not separate any non-terminals (compare to Lemma 60 on page 86). If there is such a min-cut, then it is an extremal min-cut. Since these can be found in polynomial time, we can determine in polynomial time whether the minimum CBMN is of size 5.

If there is no minimum CBMN of size 5, then it still is a good idea to choose extremal min-cuts that do not separate any non-terminals. For each of the remaining three terminal bipartitions, we hence check if there is such a min-cut and in the positive case choose it. This leaves us with at most three terminal bipartitions for which we have to choose one min-cut each splitting the set of non-terminals into as few sets as possible. For notational ease, we will assign colors to these three terminal bipartitions – the red, the green, and the blue one. A min-cut of the corresponding terminal bipartition will from now on be referred to as a red, green, or blue min-cut.

What we have achieved so far is that we either found a minimum CBMN of size at most 5 or we know that every minimum CBMN has size at least 6, and no non-terminals can be merged with any terminals. Moreover, none of

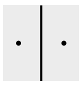
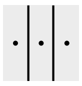
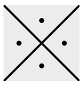
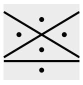
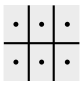
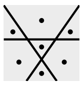
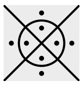
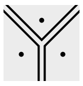
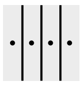
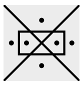
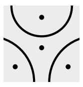
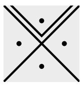
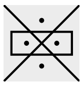
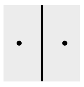
	Number of non-terminals in minimum CBMN						
Case	2	3	4	5	6	7	8
a)							
b)							
c)							
d)							
e)							

Table 2: This table lists all classes of cases how the chosen min-cuts of a 4-terminal network can split the set of non-terminals into 2 to 8 non-terminals. A line denotes a min-cut, a dot a non-terminal. Terminals are not shown. Each of the 7 terminal bipartitions of a 4-terminal network demands the choice of a corresponding min-cut. Theorem 59 and Lemma 60 allow to make this choice optimally in polynomial time for at least 4 of them. At most three terminal bipartitions are left. If just one or two min-cuts are depicted, then either there are only one or two terminal bipartitions left or multiple min-cuts are equivalent with respect to the non-terminals (congruent min-cuts). The case of 0 and 1 non-terminals are not listed.

the remaining terminal bipartitions allow a min-cut not separating any non-terminals. Choosing an arbitrary min-cut for each of the remaining terminal bipartitions leads to a minimum CBMN with a total of at most 12 nodes – 4 terminals plus 2^3 non-terminals. The task is to find out which case applies.

minimum cbmn of size 6. If there is a minimum CBMN of size 6, then min-cuts can be chosen such that the set of non-terminals is split into exactly two sets. This is indicated by case 2a) of Table 2 that is also depicted on the right. If there is only one min-cut left to be chosen, then any min-cut will suffice. If, however, several need to be chosen, then we look for min-cuts belonging to different terminal bipartitions that separate the set of non-terminals in the exact same way. We call such min-cuts *congruent min-cuts* as defined in the following.



▷ definition 8 (congruent min-cuts): Let $B_1, B_2 \in \mathcal{B}$ be two different terminal bipartitions. A B_1-min-cut (S_1, T_1) and a B_2-min-cut (S_2, T_2) are *congruent* if $S_1 \cap (V - \mathcal{T}) = S_2 \cap (V - \mathcal{T})$ or $T_1 \cap (V - \mathcal{T}) = S_2 \cap (V - \mathcal{T})$.

In the following, we will see that congruent min-cuts can be identified for a set of terminal bipartitions in polynomial time – even though there may be up to exponentially in $|V|$ many min-cuts.

▷ lemma 68: There is an algorithm deciding for any network N in time $O\left(p(|N|) \cdot 2^{|\mathcal{B}|}\right)$, where p is some fixed polynomial, whether there is a choice γ of min-cuts such that all chosen min-cuts are pairwise congruent, and outputting γ, if there is one.

Proof. For each terminal bipartition $B \in \mathcal{B}$, we compute a corresponding max-flow f_B. Next, we compute the residual network N_{f_B} which is composed of strongly connected components by Lemma 6 on page 32. For every terminal t_i, we connect all neighbors of t_i with each other and then delete all edges incident to t_i. This way, we remove terminals from strongly connected components without creating new components of non-terminals.

Next, we consider two different orientations for the edges of each residual network N_{f_B}: Either all edges remain the way they are, or the direction of each edge is reversed. For each of the $2^{|\mathcal{B}|}$ emerging possibilities, called *global orientation*, we copy the edges of all residual networks into a new graph consisting of the nodes $V - \mathcal{T}$. We claim that there is a global orientation such that the resulting graph decomposes into at least 2 strongly connected components if, and only if, there is a choice of congruent min-cuts.

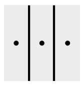
If V' is a maximal strongly connected component in this graph, then for each $B \in \mathcal{B}$, it follows that $B \cup V'$ or $B \cup (V - \mathcal{T} - V')$ is a B-min-cut. Since creating these networks and checking the number of strongly connected components can be done in the claimed time, we are left to prove correctness.

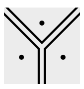
So assume there is a global orientation such that the resulting graph G contains at least 2 strongly connected components with V' being one of them. All edges in G with exactly one endpoint in V' either leave or enter V'. If they leave V', then all edges enter $V - \mathcal{T} - V'$. Let X be the set entered by all edges with exactly one endpoint in X. Then, for every $B \in \mathcal{B}$, $B \cup X$ is a B-min-cut of capacity 0 in N_{f_B} and thus a B-min-cut in N by Lemma 5 on page 32.

If for every global orientation, G is a single strongly connected component, then for every $V' \subset V - \mathcal{T}$, there are edges leaving V' and edges entering V'. This means that for every V' there is a $B \in \mathcal{B}$ such that $B \cup V'$ is not a B-min-cut. □

Clearly, if the number of terminals is fixed, like in our case to 4, then we can even find congruent min-cuts in polynomial time. Hence, $(N, 6) \in$ 4-MCBMN can be decided in polynomial time as well. For our purposes, we extend the notion of crossing min-cuts to min-cuts belonging to different terminal bipartitions. So let $B_1, B_2 \in \mathcal{B}$ be two distinct terminal bipartitions, (S_1, T_1), and (S_2, T_2) a B_1- and B_2-min-cut, respectively. Then, these two min-cuts are crossing min-cuts if, and only if, $(S_1 - S_2) \cap (V - \mathcal{T}) \neq \emptyset$ and $(S_2 - S_1) \cap (V - \mathcal{T}) \neq \emptyset$.

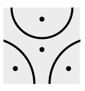
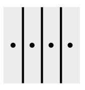
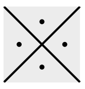
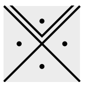
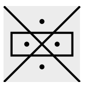
MINIMUM CBMN OF SIZE 7. If there is a minimum CBMN of size 7, then it contains 3 non-terminals that are separated pairwise by min-cuts. Note that none of these min-cuts may be crossing min-cuts since this would immediately yield 4 non-terminals. Every min-cut that is left to be chosen by now separates the set of non-terminals into two sets, let's say (S_1, S_2). Another min-cut may be congruent to the first or separate one of the two sets into two more sets. A third min-cut may be congruent to one of the first two or separate the nodes in a new way not introducing any new sets. The discussed possibilities belong to one of two cases which are represented by the two little images to the right.





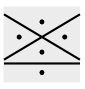
If the minimum CBMN belongs to the class represented by the upper image, then to at least one of the displayed two min-cuts, there is no congruent min-cut that is also chosen. Note, that this min-cut can be replaced by an appropriate extremal min-cut of the same color, since the other min-cuts only separate nodes on one side of it. Thus, for each color and each of its extremal min-cuts, we check if after choosing it we can find two congruent min-cuts of the other colors not crossing the chosen extremal min-cut.

If the minimum CBMN belongs to the lower image, then the presented techniques do not suffice. By trying all extremal min-cuts, however, we will find a CBMN of size at most 8 – the chosen min-cuts will then look like case 4c) in Table 2. We hence find in polynomial time a CBMN at most one node larger than a minimum CBMN.

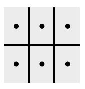
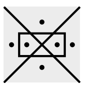






MINIMUM CBMN OF SIZE 8. If there is a minimum CBMN of size 8, then min-cuts can be chosen accordingly to one of the five images to the right. We can distinguish two main cases: Either there are crossing min-cuts or there are none. If there are none, then the three min-cuts separate the non-terminals as indicated in the first or second image on the right, if there are some, then it corresponds to one of the three remaining images.

If there are min-cuts crossing each other, then we already have separated the set of non-terminals into 4 sets. The third cut either is congruent to an already chosen one (third image) or it separates only nodes that have already been separated, but is not congruent to any other chosen min-cut (fourth and fifth image). Cases belonging to the third image can be detected quite easily by finding two colors which allow to choose two congruent min-cuts. For the remaining color, any min-cut that can be chosen crosses these two congruent ones as there is no CBMN of size 7. Cases belonging to the fourth image and fifth image cannot be correctly detected. At least, for the fourth image, by trying all extremal min-cuts, we are able to choose min-cuts as depicted in case 5a) in Table 2. For the fifth image, this technique always leads to chosen min-cuts as depicted in case 6b). Hence, the algorithm finds in polynomial time a CBMN at most two nodes larger than a minimum CBMN.

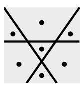


MINIMUM CBMN OF SIZE 9. If there is a minimum CBMN of size 9, then min-cuts can be chosen that split the non-terminals into 5 sets. Among these, there need to be two crossing min-cuts as otherwise, there would be a smaller CBMN. These two crossing min-cuts yield four sets of non-terminals. The third min-cut splits one of these sets into two new sets as indicated in the image on the right. This third min-cut can be replaced by an extremal min-cut of the same color. After this, the two crossing min-cuts can also be replaced by extremal min-cuts of the same color. Putting this together, by trying all possibilities to choose extremal min-cuts, we are able to detect this case in polynomial time.

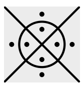



MINIMUM CBMN OF SIZE 10. In this case, again, there are two min-cuts crossing each other yielding four sets of non-terminals. The third min-cut needs to split two of these sets into two new ones each. This may happen as depicted in the two images on the right. Cases belonging to the upper image can be detected by the algorithm by again trying all choices of extremal min-cuts. The cases belonging to the lower image again are difficult. However, the min-cut represented as the rectangle in the middle may be replaced by an extremal min-cut such that this rectangle becomes smaller. That case still looks like the lower image. Hence, we may assume that this min-cut is an extremal min-cut. In the image, there are four regions in that rectangle two of which are empty. By choosing extremal min-cuts for the remaining two colors, one will find a choice of min-cuts where at most one of these two regions is not empty anymore. Hence, we will find a CBMN at most one node larger than a minimum CBMN.

MINIMUM CBMN OF SIZE 11. If a minimum CBMN has 7 non-terminals, then the min-cuts split the nodes as depicted on the right. There are two min-cuts crossing each other creating four sets of non-terminals. The third min-cut splits three of these four sets into 2 new sets each. This case, too, can be detected by trying all possible choices of extremal min-cuts.



MINIMUM CBMN OF SIZE 12. If none of the previous cases applies, then clearly all minimum CBMNs have size 12 and have 8 non-terminals. No matter which three min-cuts are chosen, they split the set of non-terminals as depicted on the right.



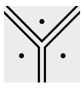
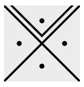
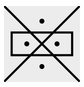
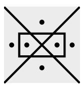
This concludes the description of the algorithm. All these considerations lead to the following theorem.

▷ THEOREM 69: There is a deterministic polynomial time algorithm that on input of a 4-terminal network N outputs a CBMN N' for N whose size is at most two greater than a minimum CBMN for N.

Proof. We check each of the just described cases beginning with the smallest number of non-terminals and work our way up. As soon as we can confirm that a case applies, we output the appropriate size and stop. □

Even if this algorithm succeeded at finding the minimum CBMN in all cases in polynomial time, the degree of the polynomial dominating the run-time highly depends on the number of terminals. If we further were able to generalize this algorithm to an arbitrary number of terminals, then we would rather expect an XP algorithm for p-MCBMN and not an FPT algorithm.

It is well possible that 4-MCBMN is not in P, but rather NP-complete. Easily, the problem is in NP as we can guess the right choice of min-cuts leading to a CBMN of size at most m. Verifying the solution can be done in polynomial time as we have a fixed number of terminal bipartitions. To show NP-hardness, the above case analysis yields promising candidates for a reduction. These are case 3b), case 4d), case 4e), and case 6b) of Table 2, i. e., the four images on the right. For each of these cases, it may be possible to augment our algorithm to correctly handle it in polynomial time, or it may be possible to show that this case cannot be decided in polynomial time. The first of these cases, 3b), corresponds to the next problem.

▷ PROBLEM 32: 4-MCBMN-7

Input: A tuple (N, m) of an undirected 4-terminal network N and a natural number m.

Question: Is there a CBMN of size 7 for N?

If this problem turns out to be NP-hard, then clearly p-MCBMN $\notin$ FPT. If, however, 4-MCBMN-7 $\in$ P, then hopes are up to fit the other cases in P as well.

Following the idea of Dahlhaus et al. [17], we try to leverage the submodularity to solve 4-MCBMN-7 in polynomial time. Grötschel, Lovász, and Schrijver [38] have found that any submodular set function $f : 2^U \to \mathbb{R}$ that can be evaluated in polynomial time can also be minimized in polynomial time, i. e., a set $X \subseteq U$ can be found in polynomial time with $f(X) < f(Y)$ for all $Y \subseteq U$. Recall that f is submodular if $f(X) + f(Y) \geqslant f(X \cap Y) + f(X \cup Y)$ for all $X, Y \subseteq U$.

If we could define a submodular set function such that it is evaluable in polynomial time, and the set for which it hits its minimum hints us at the solution to 4-MCBMN-7, then we could find the minimum in polynomial time. Let us define a function f_N over the non-terminals of N as follows.

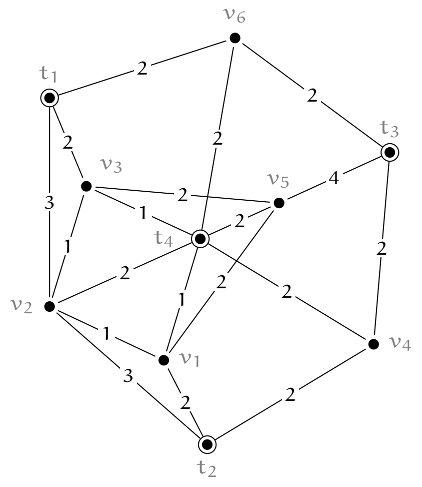
$$f_N(X) = \begin{cases} 1 & \text{if } \{t_1, t_2\} \cup X \text{ is not a } \{t_1, t_2\}\text{-min-cut or} \\ & \{t_1, t_2\} \cup X \text{ is a } \{t_1, t_2\}\text{-min-cut but choosing} \\ & \{t_1, t_2\} \cup X \text{ as } \{t_1, t_2\}\text{-min-cut} \\ & \text{does not allow a CBMN of size } 7\,; \\ 0 & \text{otherwise}\,. \end{cases} \tag{119}$$

Function f_N can indeed be evaluated in polynomial time. It is easy to decide whether $\{t_1, t_2\} \cup X$ is a $\{t_1, t_2\}$-min-cut or not. Moreover, it can be decided efficiently whether the choice of $\{t_1, t_2\}$-min-cut as $\{t_1, t_2\}$-min-cut still allows the choice of other min-cuts in order to obtain a CBMN of size 7. For this, a slightly modified approach of the one described in Lemma 68 suffices. However, it does not always define a submodular set function. The network N of Figure 22 is an example of a network for which f_N is not submodular. To find this counterexample, we started off at a set of min-cuts that causes a network to exhibit the desired property making f_N non-submodular. Since it is unclear whether for such a system of set systems a network exists at all, we formulated a linear programming forcing the desired min-cuts to be min-cuts and all other cuts to be non-minimal. As this leads to an exponential sized linear program, we made use of a technique called *column generation* as described by Grötschel, Lovász, and Schrijver [39].

Hence, we have further confined the truly hard cases – these are networks N for which f_N is not submodular. Further following the idea of Dahlhaus et al. [17], we tried to construct a reduction from an NP-complete problem using the network of Figure 22 as NP-completeness gadget – unfortunately unsuccessfully.

Concluding to this chapter, we are left to find that the problem of determining the complexity of MCBMN is very challenging. Even though we have pinpointed the difficulty to a few cases, we are unable to completely answer it but rather give close upper and lower bounds on the complexity in the first levels of the polynomial hierarchy.

Another main contribution is to show that the unique min-cuts assumption, which is quite common in this area of research, is not only unrealistic in practical settings where there is only a small set of distinct edge capacities, but that enforcing this uniqueness by slightly perturbing edge capacities may exponentially increase the size of a minimum CBMN. Moreover, the unique min-cuts assumption made in previous research yields wrong expectations concerning the worst-case complexity of constructing a minimum CBMN. Furthermore, we showed that the approximability by contraction-based vertex sparsifiers does not behave smoothly.



Terminal Bipartition T	All T-min-cuts	Capacity
$T = \{t_1, t_2\}$	$T \cup \{v_2\}$	12
	$T \cup \{v_1, v_2\}$	
	$T \cup \{v_2, v_3\}$	
	$T \cup \{v_1, v_2, v_3\}$	
$T = \{t_1, t_3\}$	$T \cup \{v_6\}$	13
	$T \cup \{v_3, v_5, v_6\}$	
$T = \{t_2, t_3\}$	$T \cup \{v_4\}$	13
	$T \cup \{v_1, v_4, v_5\}$	

Figure 22: The depicted network N is a network for which the function f_N defined in (119) is not submodular. All terminal isolating min-cuts contain exclusively that terminal. All other min-cuts are listed above. N allows a minimum CBMN of size 7. For this, there are two possible choices of min-cuts: 1) $\{t_1, t_2, v_1, v_2\}$, $\{t_1, t_3, v_3, v_5, v_6\}$, $\{t_2, t_3, v_4\}$, and 2) $\{t_1, t_2, v_2, v_3\}$, $\{t_1, t_3, v_6\}$, $\{t_2, t_3, v_1, v_4, v_5\}$. All other choices lead to larger CBMNs. Hence, we have $f_N(\{v_1, v_2\}) = f_N(\{v_2, v_3\}) = 0$ and $f_N(\{v_2\}) = f_N(\{v_1, v_2, v_3\}) = 1$ implying that f_N is not a submodular set function.

Without the unique min-cuts assumption, there may be several min-cuts for each bipartition of terminals to select from. A wrong choice and thus a merging of less suitable nodes may prevent reaching a minimum CBMN. For the complexity of deciding whether a CBMN of a given size exists we have established lower and upper bounds in the first levels of the polynomial hierarchy. We also presented some partial results concerning fixed-parameter tractability. The exact complexity remains open.

We conjecture that p-MCBMN is not fixed-parameter tractable, but at best in XP. Moreover, we conjecture that MCBMN is not in coNP, but rather Σ_2^P-complete.

5 ONLINE ANALYSIS OF DYNAMIC STORAGE NETWORKS

This chapter introduces our model of dynamic storage networks. We identify request sequences that are the most difficult and devise a general technique to find the best strategy of how to use the storage sites in a network. Moreover, worst case networks belonging to certain classes are constructed.

In the following, Section 5.1 briefly presents several ways in which the basic model of flow networks has been generalized. We also give a short introduction to the world of online problems. Section 5.2 then introduces our model which is a certain kind of a dynamic network in the context of online problems. In Section 5.3, a technique is developed for the competitive analysis of arbitrary networks. Finally, Section 5.4 applies this technique and presents bounds for the competitivity of certain network classes.

5.1 DYNAMIC NETWORKS AND ONLINE PROBLEMS

This section gives a short overview over the two topics of dynamic networks and of online problems before the next section introduces our model of dynamic storage networks that combines these two worlds.

DYNAMIC NETWORKS. The motivation to consider flow networks and min-cuts in them originated from military purposes [75]. In particular, it was of interest to find a minimum number of railway tracks that need to be cut to separate certain parts of a territory. The usefulness of the new approach stimulated to model more and more aspects of railway systems and soon many more applications. Many of these generalized flow network models are grouped under the term *dynamic networks*. In general, such networks contain some kind of temporal component. For example, it may take time for flow to pass an edge, flow may be delayed at nodes, and edge capacities may change over time. This may include the deletion or addition of edges to a network at some point of time. In the following, we present a number of problems from the realm of dynamic networks and present techniques useful for our needs.

One of the earliest such networks has been considered by Ford and Fulkerson [32]. In their setting, flow does not instantly traverse an edge, but takes a certain amount of discrete time steps. The objective is to find flow that maximizes the amount of flow reaching sink nodes within a given number of time units. For this, they develop the technique of *time-expanded networks*, which is a conventional s-t flow network. To obtain the time-expanded network, the original network is copied for the needed number of time steps. An original edge from u to v that takes k time steps to be traversed is then represented in the time-expanded network by edges from u of every i^{th} copy to v of the $(i+k)^{\text{th}}$ copy. Now, we simply use conventional max-flow techniques to find a max-flow. Figure 24 on page 109 is an example for a time-expanded network.

This techniques also helps for more sophisticated models and problems and has been adapted to different needs. A drawback is that the size of the time-expanded network may be exponential in the length of the original problem. Fleischer and Skutella [30] developed the *condensed time-expanded* *network* applicable in some cases resulting in a blow-up in size that is polynomially bounded.

All kinds of problems from the realm of static networks and static flows, like multi-commodity flow problems or transshipment problems, have been defined with different temporal components. Problems in the dynamic network setting may be modeled using continuous time or discrete time, may involve storage nodes, and may allow the change of some parameters over time. This allows many dimensions in which new models and problems can be found. Several surveys have been published that give an overview over this field of research, these include the ones by Aronson [6], Kotnyek [54], Lovetskii and Melamed [59], Powell, Jaillet, and Odoni [70], and Skutella [76].

We specifically want to address one important feature of some dynamic networks: the ability to store flow at nodes. In the literature, this feature has been given explicitly or implicitly. In the latter case, storing flow at a node can be simulated by looping flow at that node in an edge with unit traversal time and an appropriate capacity. In the explicit case, the node conservation constraint may be relaxed of some nodes such that they may store flow. The amount of flow that can be stored may be limited or unlimited.

Fleischer and Skutella [30] show that for the minimum cost flow problem with continuous time and fixed traversal times and arc capacities, storage nodes are not needed for any optimal solution. Minieka [63] also addresses storage, but then assumes that no node is capable of storing flow to avoid difficulties in the analysis. Halpern [42] looks for a max-flow in a model with discrete time, fixed traversal times, but varying arc capacities, where nodes may store flow. The solution uses an augmenting path algorithm. Fleischer and Orlin [29] look for a minimum cost quickest integral transshipment in a network model involving storage nodes. Recapitulatory, storage nodes may be used but do not play a very prominent role in any of these solutions. Mostly, this is due to the fact that solutions usually employ some kind of time-expanded network which quite effortlessly takes storage nodes into account. This will change in our online setting, which will be introduced soon.

ONLINE PROBLEMS. We encounter online problems all the time. Should you hold on to your stocks or sell them? Should you get a monthly public transport commuter ticket or pay each time? Should you gas up your car now or wait until tomorrow?

All of these questions share a similarity – information is missing to find the globally optimal answer. But what strategy is optimal with respect to incomplete information? How well does the strategy work to get a monthly commuter ticket every time it would have paid off last month?

Online problems deal with just these kinds of questions. They model problems where the input is given one piece at a time, and upon receiving the input, the online algorithm must take action in response to it. Such an action is afflicted with a cost or a payoff, and the total amount of it needs to be minimized or maximized, respectively. To every online algorithm solving such problems, there is an offline algorithm that gets the entire input at once. As part of the *competitive analysis* the performance of an online algorithm is compared to the performance of an optimal offline algorithm.

A well-studied setting is the famous *ski rental problem*. Suppose you are going skiing for the first time, and you are asking yourself whether to buy or to rent skis. Let us say buying skis costs $\$10$ while renting them costs

\$1 a day. If you knew how many times you are going skiing, the choice would be clear. However, this is exactly the information missing that turns the problem into an online problem. Indeed, regardless of your strategy it can be shown that there always is a scenario that makes you pay at least twice of what you could have paid. If your strategy tells you to rent skis for j days and then buy them if you are still up for it, then you pay \$$(j+10)$. The minimum you could have paid is \$$\min\{j, 10\}$ of course. An adversary, that tries to make you pay as much as possible, will then be able to make you pay twice as much as you optimally would. For this, it waits until you buy and then makes sure you never ski again.

On the other hand, there is a quite simple strategy ensuring that you do not pay more than twice as much either. It makes you buy as soon as you have skied 10 times. In total, this makes the problem *2-competitive*. The formal definitions of these notions follow later.

What we have just learned also applies to the mentioned problem of deciding whether to buy a monthly commuter ticket or not. The suggested strategy to buy such a ticket once it would have paid off for the last month does not ensure any bounds. The ski rental strategy, however, instructs us to get a monthly commuter ticket as soon as we have spent as much on single tickets as a monthly ticket costs. This way, we will certainly never pay more than twice as much as we optimally would have.

It is worth to mention that there exists a randomized algorithm for the ski rental problem that performs better in terms of an *expected* cost. This is based on the assumption that the adversary does not know the outcome of the coin flips of the randomized algorithm ahead. Karlin et al. [49] show that there is a randomized algorithm with an expected cost of $e/(e-1) \approx 1.58$, and that there is no better randomized algorithm.

For a more detailed introduction to these topics, refer to the textbook *Online Computation and Competitive Analysis* by Borodin and El-Yaniv [9], or the surveys by Albers [3] as well as by Fiat and Woeginger [28].

5.2 DYNAMIC STORAGE NETWORKS – THE MODEL

In this section, we introduce our model that combines dynamic networks and online problems. The power grid is modeled as a multi-terminal network where power is treated as a good that is instantaneously routed from one node of the network to another one. In each time step, there is a certain amount of supply and demand which vary over time. The terminals are assigned the role of a source, a storage node, or a sink. Edges are assigned a capacity very similar to power grids: A power line has a power capacity; an upper bound of energy that can be transmitted in a certain amount of time. To avoid power outages, the amount of power transmitted via such a line needs to stay below that bound. The nodes of the grid obey the flow conservation – or act as supply, storage, or demand node.

▷ DEFINITION 9 (DYNAMIC STORAGE NETWORK): A *dynamic storage network* $M = (V, E, S, R, D, c)$ consists of an undirected and connected graph (V, E) (again modeled by a directed but symmetric edge relation), the disjoint and non-empty sets of *supply nodes* S, *storage nodes* R, and *demand nodes* D such that $S \uplus R \uplus D \subseteq V$, and finally a (symmetric) capacity function $c : V \times V \to \mathbb{R}_0^+$.

The supply, storage, and demand nodes will be referred to as *terminals* $\mathcal{T}$, all other nodes as *non-terminals*. We again assume that some arbitrary but fixed total order is imposed over the terminals that we need in the context of external flows.

▷ DEFINITION 10 (REQUEST SEQUENCE): A *request sequence* $\bar{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_l)$ (or simply $\bar{\sigma} = \sigma_1\sigma_2\ldots\sigma_l$) of length $l \in \mathbb{N}^+$ is a sequence of functions $\sigma_i : S \cup D \to \mathbb{R}_0^+$ called *requests* that assign a non-negative value $\sigma_i(v)$ to each supply and demand node. Each request σ_i has a *duration* $\tau(\sigma_i) \in \mathbb{R}^+$.

We will use the notation $\big(\sigma_i(s_1), \ldots, \sigma_i(s_{|S|}) \,|\, \sigma_i(d_1), \ldots, \sigma_i(d_{|D|})\big)^{\tau(\sigma_i)}$ for a request σ_i of duration $\tau(\sigma_i)$ (or $\sigma_i^{\tau(\sigma_i)}$). The vertical line separates the supplies from the demands for a better readability.

The operation of *concatenating two request sequences* is defined as the natural concatenation of the two sequences and yields a new request sequence. We also define $\sigma^{\tau_1}\sigma^{\tau_2} = \sigma^{\tau_1+\tau_2}$. This has implications on what a prefix is. A request sequence $\bar{\sigma}_p$ is a *prefix* of another request sequence $\bar{\sigma}$ if there is a request $\bar{\sigma}_s$ with $\bar{\sigma}_p\bar{\sigma}_s = \bar{\sigma}$. A request sequence $\bar{\sigma}_p$ is a prefix of a set of request sequences ρ if $\bar{\sigma}_p$ is a prefix of each request sequence in ρ. Moreover, $\bar{\sigma}_p$ is a *maximal prefix* for a set of request sequences ρ if $\bar{\sigma}_p$ is a prefix for every request sequence in ρ and there is no other prefix $\bar{\sigma}_p'$ for ρ such that $\bar{\sigma}_p$ is a prefix of $\bar{\sigma}_p'$. Note that request sequences cannot be empty and hence there is no empty prefix.

Since request sequences can be represented in different ways, we make assumptions concerning their representation. The difference lies in the durations of the requests. While $\sigma^{\tau_1}\sigma^{\tau_2}$ and $\sigma^{\tau_1+\tau_2}$ are by definition the same request sequences, it does make a difference for the online algorithm which will be presented one request at a time. In order to make the representation of request sequences unique, we assume that the duration of every request is as long as possible unless for a shorter duration we obtain a maximal prefix for some subset of ρ with at least two sequences. For example, when the two request sequences σ_1^2 and σ_1^3 are in the same set, they are represented as σ_1^2 and $\sigma_1^2\sigma_1^1$. This assumption can always be fulfilled by changing the way the request sequences are represented.

For a request σ and a supply node (demand node) v_j, we call $\tau(\sigma) \cdot \sigma(v_j)$ the *supply* (*demand*) of v_j. The supply (demand) of a request is the sum of all its supplies (demands), and the supply (demand) of a request sequence is the sum of the supplies (demands) of all its requests.

▷ DEFINITION 11 (SEQUENCE OF FLOWS): A *sequence of flows* $\bar{f} = (f_1, f_2, \ldots, f_l)$ with respect to a request sequence $\bar{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_l)$ is a sequence of functions $f_i : V \times V \to \mathbb{R}$ where each f_i satisfies

$$\begin{aligned}
&\text{the } \textit{capacity constraint} & & f_i(e) \leqslant \tau(\sigma_i) \cdot c(e) \text{ for all } e \in V \times V, && (120)\\
&\text{the } \textit{skew symmetry} & & f_i(u,v) = -f_i(v,u) \text{ for all } (u,v) \in V \times V, && (121)\\
&\text{the } \textit{flow conservation} & & f_i(v) = 0 \text{ for all } v \in V - \mathcal{T}, && (122)\\
&\text{the } \textit{request constraints} & & 0 \leqslant -f_i(s) \leqslant \tau(\sigma_i) \cdot \sigma_i(s) \text{ for all } s \in S \text{ and} && (123)\\
& & & 0 \leqslant f_i(d) \leqslant \tau(\sigma_i) \cdot \sigma_i(d) \text{ for all } d \in D\text{, and} && (124)\\
&\text{the } \textit{storage constraint} & & -f_i(v) \leqslant r_i(v) \text{ for all } v \in R, && (125)
\end{aligned}$$

where

$$r_i(v) = \begin{cases} \sum_{j=1}^{i-1} f_j(v) & \text{if } i > 1\,; \\ 0 & \text{if } i = 1\,. \end{cases} \qquad (126)$$

Intuitively, the request constraints do not allow any supply node to send out more flow as it can supply according to the request, and no demand

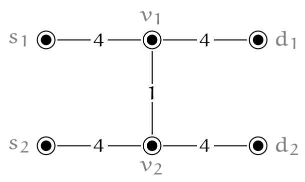


Figure 23: This dynamic storage network consists of two supply nodes s_1, s_2, two storage nodes v_1, v_2, and two demand nodes d_1, d_2. As before, ◉ denotes a terminal. Supply nodes are named s_i, storage nodes v_i, and demand nodes d_i. Consider the request $\sigma_1 = (1,0\,|\,0,0)^4$, that allows any algorithm to store four units of flow in an arbitrary way at the storage nodes. Now let $\sigma_a = (0,0\,|\,4,0)$ and $\sigma_b = (0,0\,|\,0,4)$ be two more requests. If $\rho = \{\sigma_1^4\sigma_a, \sigma_1^4\sigma_b\}$, then any optimal offline algorithm will store the four flow units of σ_1 at v_1 if the current request sequence is $\sigma_1^4\sigma_a$ and at v_2 otherwise. An optimal online algorithm needs to find a compromise between these two extremes and stores two flow units at each storage node. This way, one flow unit will remain in the dynamic storage network by the end of either request sequence. Hence, the competitiveness is $4/3$.

node may consume more flow than it demands according to the request. Moreover, the request constraints demand that no supply node may have a positive net flow, and no demand node may have a negative net flow.

The duration of a request could be modeled by simply repeating the request for a certain number of times. That way, however, only durations that are whole numbers can be modeled and the representation could become quite large. It turns out to be an elegant solution to simply scale up and down each edge's capacity with the request's duration. This happens in the capacity constraint (120).

A storage node is modeled as a node with relaxed flow conservation constraint. It stores flow if it has a positive net flow, and it releases stored flow if it has a negative net flow. While it may store arbitrary amounts of flow, it can only release as much flow as it has stored earlier in the sequence of flows. The value $r_i(v)$ is the amount of flow stored at v during the requests 1 through $i-1$ and thus available at the beginning of request σ_i. Initially, we have $r_1(v) = 0$ for all $v \in R$.

Having defined a sequence of flows with respect to a request sequence, we now define a flow for a dynamic storage network M.

▷ DEFINITION 12 (FLOW IN A DYNAMIC STORAGE NETWORK): A *flow in a dynamic storage network* is a function $f : V \times V \to \mathbb{R}$ complying to (120), (121), and (122) with the additional properties that $f(s) \leqslant 0$ for all $s \in S$ and $f(d) \geqslant 0$ for all $d \in D$.

If we view a dynamic storage network M as a conventional multi-terminal network N, then f is a flow in M if it also is a flow in N, supply nodes have a non-positive net flow, and demand nodes a non-negative net flow. For a dynamic storage network M and a flow f in M, there is a request sequence such that a corresponding sequence of flows may contain f. The external flow of a flow is defined analogously to multi-terminal networks – if f is a flow in a dynamic storage network M, then $\vec{f} = \big(f(v_1), f(v_2), \dots, f(v_{|\mathcal{T}|})\big)$ is the corresponding external flow. Now, $F_M = \{\vec{f} \mid f \text{ is a flow in } M\}$ is the external flow pattern of M (compare to page 18). Again, while a flow uniquely describes its external flow, an external flow may describe several flows evoking the same net flows. The *value of a flow* in a dynamic storage network is the sum of all positive (or all negative) net flows as it has been defined before.

▷ DEFINITION 13 (DYNAMIC STORAGE SYSTEM): A *dynamic storage system* is a tuple $\mathcal{M} = (M, \rho)$ of a dynamic storage network M and a non-empty set of request sequences ρ. An *unrestricted dynamic storage system* is a dynamic storage system where ρ contains all possible request sequences for M.

Let $\Phi_{M,\bar{\sigma}}$ be the set of all possible flow sequences with respect to a dynamic storage network M and a request sequence $\bar{\sigma}$. We will often neglect the M in the subscript if clear from the context. Now, we may define on- and offline algorithms.

▷ DEFINITION 14 (ONLINE AND OFFLINE ALGORITHM): An *offline algorithm* for $\mathcal{M} = (M, \rho)$ is a function $\mathcal{A}$ such that $\mathcal{A}(\bar{\sigma}) = \bar{f}$ where $\bar{f} \in \Phi_{\bar{\sigma}}$ and $\bar{\sigma} \in \rho$. An *online algorithm* has the additional requirement that it outputs the same flow sequence for the same prefix of different request sequences.

We measure the performance of an algorithm with a profit function in contrast to a cost function. Let $\Phi_M = \{\Phi_{M,\bar{\sigma}} \mid \bar{\sigma} \in \rho\}$ be the set of all possible flow sequences in response to all possible request sequences. The profit of a flow sequence $\bar{f} = (f_1, f_2, \ldots, f_l)$ is a function $\text{profit} : \Phi_M \to \mathbb{R}$. The profit of $\mathcal{A}$ on $\bar{\sigma}$ is the profit of the flow sequence produced by $\mathcal{A}$. We define

$$\text{opt}(\bar{\sigma}) = \max_{\bar{f} \in \Phi_{\bar{\sigma}}} \left\{ \text{profit}(\bar{f}) \right\} \tag{127}$$

as the optimal profit achievable by an offline algorithm. An offline algorithm is optimal if its profit on $\bar{\sigma}$ equals $\text{opt}(\bar{\sigma})$ for every $\bar{\sigma} \in \rho$. An optimal offline algorithm will also be denoted by opt. Even though there may not be a unique optimal offline algorithm, we often refer to *the* optimal offline algorithm. As the performance is measured only by the profit function, we do not care about other aspects of these offline algorithms. We will only consider request sequences $\bar{\sigma}$ with $\text{opt}(\bar{\sigma}) > 0$.

In the following, we consider the profit function

$$\text{profit}(\bar{f}) = \sum_{i=1}^{l} \sum_{v \in D} f_i(v) \tag{128}$$

where the profit of a flow sequence is the sum of all flow units reaching demand nodes.

▷ DEFINITION 15 (COMPETITIVENESS): An online algorithm $\mathcal{A}$ for a dynamic storage system $\mathcal{M} = (M, \rho)$ is called c-*competitive* if for each $\bar{\sigma} \in \rho$ we have $\text{opt}(\bar{\sigma}) \leqslant c \cdot \text{profit}(\mathcal{A}(\bar{\sigma}))$. (M, ρ) is c-competitive if there is a c-competitive online algorithm $\mathcal{A}$ for (M, ρ). M is c-competitive if (M, ρ) is c-competitive for every ρ. We define $\text{comp}(\cdot)$ to be a function that assigns a value c to a network, a dynamic storage system, or an online algorithm if it is c-competitive, but not c'-competitive for any $c' < c$. This value c will also be referred to as *competitive ratio*. An online algorithm $\mathcal{A}$ is *optimal for* (M, ρ) if $\text{comp}(M, \rho) = \text{comp}(\mathcal{A})$.

The best competitiveness is 1 in which case the optimal offline and optimal online algorithm send the same amount of flow to demand nodes. If the competitiveness is worse, i. e., greater than 1, then for every online algorithm there are request sequences for which it fails to send as much flow to demand nodes as the optimal offline algorithm can do. See Figure 23 for an example demonstrating these concepts.

5.3 A TECHNIQUE FOR THE ONLINE ANALYSIS

The driving question of this section is: Given a dynamic storage network, what is its competitivity? We will derive a general technique to answer such questions. For this, we are first interested in finding a small set of request sequences for a given dynamic storage network still maintaining the same competitiveness. Clearly, we have $\operatorname{comp}(M, \rho') \leqslant \operatorname{comp}(M, \rho)$ if $\rho' \subseteq \rho$. We aim for a small set ρ' with $\operatorname{comp}(M, \rho') = \operatorname{comp}(M, \rho)$.

▷ DEFINITION 16 (HARD SET OF REQUEST SEQUENCES): For an unrestricted dynamic storage system (M, ρ), we call any $\rho' \subseteq \rho$ a *hard set of request sequences* if $\operatorname{comp}(M, \rho') = \operatorname{comp}(M, \rho)$. In particular,

$$\rho_{\mathcal{A}} = \left\{ \bar{\sigma} \in \rho \mid \operatorname{opt}(\bar{\sigma}) = \operatorname{comp}(\mathcal{A}) \cdot \operatorname{profit}\left(\mathcal{A}(\bar{\sigma})\right) \right\} \tag{129}$$

is a hard set of request sequences for an optimal online algorithm $\mathcal{A}$.

Analogously to the external flow pattern for multi-terminal networks, defined in Chapter 2 on page 18, we now give the corresponding definition for a dynamic storage network.

▷ DEFINITION 17 (EXTERNAL FLOW PATTERN): For a dynamic storage network M, the *external flow pattern* is $F_M = \left\{ \vec{f} \mid f \text{ is a flow in } M \right\}$.

Even though the definitions look identical, they differ by the definition of a flow. For example, a dynamic storage network does not allow any positive net flow at a supply node.

▷ LEMMA 70: For any dynamic storage network M, the external flow pattern F_M is a $|\mathcal{T}| - 1$ dimensional, convex polytope.

Proof. Hagerup et al. [41] have shown that for multi-terminal networks, a vector $(x_1, \dots, x_k)$ with $k = |\mathcal{T}|$ is an external flow if, and only if,

$$\sum_{i=1}^{k} x_i = 0 \quad \text{and} \tag{130}$$

$$\sum_{v_i \in T'} x_i \leqslant c_{M,T'}, \quad \text{for all } T' \subseteq \mathcal{T}, \tag{131}$$

where $c_{M,T'}$ is the capacity of a T'-min-cut in M.

Feasible flows in a dynamic storage network have the additional restriction to have non-positive net flows at supply nodes and non-negative net flows at demand nodes. Hence, we add

$$x_i \leqslant 0 \quad \text{for all } 1 \leqslant i \leqslant |S| \quad \text{and} \tag{132}$$

$$x_i \geqslant 0 \quad \text{for all } |S| + |R| + 1 \leqslant i \leqslant |\mathcal{T}|. \tag{133}$$

These linear constraints describe the external flow pattern which hence is a convex polytope.

Concerning the dimension, note that due to (130), the polytope is not fully dimensional. Moreover, there are $k - 1$ linearly independent external flows found as follows: For each of the first $k - 1$ terminals denoted by u, find some terminal v of different kind than u. Let $f_{u,v}$ be a flow from u to v of positive value if u is a supply or storage node and from v to u if v is a supply or storage node, and let $\vec{f}_{u,v}$ be the corresponding external flow. □

This lemma implies that for two arbitrary external flows $\vec{f}_1, \vec{f}_2$, the external flow $\lambda \cdot \vec{f}_1 + (1 - \lambda) \cdot \vec{f}_2$ represents a flow for any $0 \leqslant \lambda \leqslant 1$. Such a flow lies somewhere in between two other flows, so to say. This is not only true for flows, but also for sequences of flows. If $\lambda \cdot \bar{f}$ denotes the sequence of flows $(\lambda \cdot f_1, \lambda \cdot f_2, \ldots)$, then we may equally define a sequence of flows that lies in between other sequences of flows. This is an important insight because the strategy chosen by an optimal online algorithm lies in between the flows chosen by an optimal offline algorithm.

By continuously thinning out the hard set of request sequences, we seek to find a smallest one allowing us to focus on the core reason that no online algorithm may perform better than the competitiveness of the network allows. While $\rho_{\mathcal{A}}$ is defined in dependence of a certain online algorithm $\mathcal{A}$, we are interested in finding a smallest hard set of request sequences. This should be independent of any online algorithm $\mathcal{A}$ as the competitiveness is a property of the network.

From the definition of hard sets of request sequences it is immediate that the union of the hard set of sequences $\rho_{\mathcal{A}}$ defined in (129) for all optimal online algorithms $\mathcal{A}$ is a hard set of request sequences for every optimal online algorithm. The next lemma states that this applies to the intersection as well.

▷ LEMMA 71: Let (M, ρ) be any dynamic storage system and A the set of all optimal online algorithms for (M, ρ). Then

$$\rho_h = \bigcap_{\mathcal{A} \in A} \rho_{\mathcal{A}} \tag{134}$$

is a hard set of request sequences.

Proof. Let $\text{comp}(M, \rho) = c$. We construct an online algorithm $\mathcal{A}_\varnothing$ that outputs an *average flow sequence* $\bar{f}_\varnothing(\bar{\sigma}) = (f_{\varnothing,1}, f_{\varnothing,2}, \ldots, f_{\varnothing,l})$ for each request sequence $\bar{\sigma}$ with

$$f_{\varnothing,i}(e) = \frac{1}{|A|} \sum_{\mathcal{A} \in A} f_{\mathcal{A},i}(e) \tag{135}$$

for all $e \in V \times V$ where $f_{\mathcal{A},i}$ is the i^{th} flow of $\mathcal{A}(\bar{\sigma})$. This truly is a flow sequence for $\bar{\sigma}$ as a consequence of Lemma 70.

For the performance of $\mathcal{A}_\varnothing$ with respect to a given request sequence $\bar{\sigma}$, we have $\text{opt}(\bar{\sigma}) = c \cdot \text{profit}\left(\mathcal{A}_\varnothing(\bar{\sigma})\right)$ if, and only if, $\text{opt}(\bar{\sigma}) = c \cdot \text{profit}\left(\mathcal{A}(\bar{\sigma})\right)$ for every $\mathcal{A} \in A$ and $\text{opt}(\bar{\sigma}) < c \cdot \text{profit}\left(\mathcal{A}_\varnothing(\bar{\sigma})\right)$ otherwise. If $\rho_H = \emptyset$ then $\text{comp}(\mathcal{A}_\varnothing) < c$ which contradicts the fact that $\text{comp}(M, \rho) = c$. Hence, ρ_h is not empty and a hard set of request sequences. □

This allows us to talk about a hard set of request sequences for a dynamic flow system M independent of an optimal online algorithm. For this set we know that if an online algorithm performs better than $\text{comp}(M)$ on any of its request sequences, then this algorithm cannot be optimal. Moreover, we have $\rho_h = \rho_{\mathcal{A}_\varnothing}$. Still, this set is not a smallest hard set of request sequences. We will address the reasons for this soon.

A dynamic storage network involves several differences in comparison to classical flow networks – the less exciting being the fact that it may have several sources and sinks. Additionally, we consider these dynamic storage networks over the course of time, allow storage nodes, and have variable supplies and demands at the sources and sinks. Still, for a dynamic storage network M and a given request sequence $\bar{\sigma}$, we can construct a *time-expanded network* that is an s-t flow network. This time-expanded network is useful

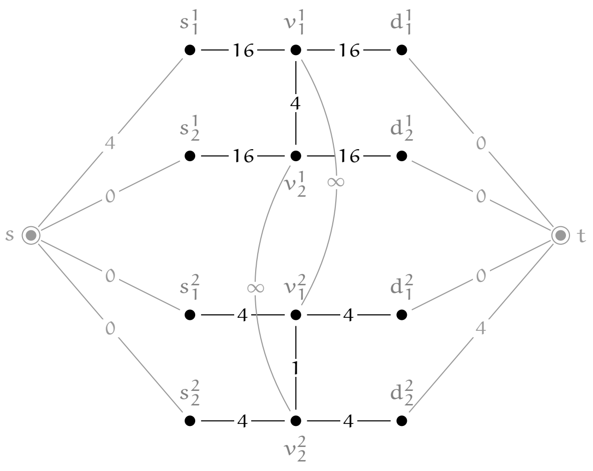


Figure 24: Let M be the dynamic storage network of Figure 23 and $\bar{\sigma} = (1,0\,|\,0,0)^4(0,0\,|\,0,4)$. The above pictured network is then the corresponding time-expanded network. It is easily seen, that a max-flow is a path-flow along s-s_1^1-v_1^1-v_2^1-v_2^2-d_2^2-t of value 4. The restriction of this flow to edges in M tells us what an optimal offline algorithm does in response to $\bar{\sigma}$. Since four flow units are sent along the edge (v_2^1, v_2^2), the optimal offline algorithm stores four flow units at v_2 in M in response to $(1,0\,|\,0,0)^4$.

for determining the maximum amount of flow that can be sent to the demand nodes of M with respect to the request sequence $\bar{\sigma}$. For this, we copy the network l times where l is the length of $\bar{\sigma}$. Then, all capacities of the i^{th} copy of M are scaled by $\tau(\sigma_i)$. Next, we introduce a source s and a sink t, then connect s to the i^{th} copy of s_j via an edge of capacity $\tau(\sigma_i) \cdot \sigma_i(s_j)$. Finally, each storage node of the i^{th} copy, for $1 \leqslant i < l$ is connected to its own copy of the $(i+1)^{\text{th}}$ copy with an edge of infinite capacity. Then, finding an s-t max-flow yields a flow f that is also a flow of maximum profit for M with respect to $\bar{\sigma}$. See Figure 24 for an example.

Request sequences for most online problems can be organized in a *prefix forest*. We define $\mathcal{P}_\rho = (V_\rho, E_\rho)$ to be the prefix forest of a set of request sequences ρ. We neglect the subscript if clear from the context.

The prefix forest $\mathcal{P}_\rho$ is defined as a graph that contains the nodes

$$V_\rho = \rho \cup \{\, \bar{\sigma} \mid \bar{\sigma} \text{ is a maximal prefix of at least two request sequences in } \rho \,\}. \tag{136}$$

There is an edge (u, v) in $\mathcal{P}_\rho$ if, and only if, u is the longest prefix of v in V. An online algorithm can be seen as a function assigning a flow to each node in V_ρ such that for every path from the root to a leaf we have a flow sequence with respect to the request sequence belonging to the leaf. If P is a tree in the prefix forest $\mathcal{P}_\rho$, then $\rho(P)$ denotes the set of request sequences corresponding to the leaves of P and hence $\rho(P) \subseteq \rho$ for every $\mathcal{P}_\rho$.

▷ LEMMA 72: For every dynamic storage system (M, ρ), there is a single tree P in $\mathcal{P}_\rho$ such that $\text{comp}(M, \rho) = \text{comp}(M, \rho(P))$.

Proof. Suppose this was not the case such that for any tree P in the prefix forest $\mathcal{P}_\rho$, there is an optimal online algorithm $\mathcal{A}_P$ with $\text{comp}(\mathcal{A}_P) < \text{comp}(M, \rho)$. Then, an online algorithm $\mathcal{A}$ that simulates $\mathcal{A}_P$ if the received request sequence belongs to P performs better than the competitiveness allows. □

We call such a tree a *hard tree*. We may infer that all request sequences of a smallest hard set of request sequences start with the same request. We may wonder how small this set might get. If $\mathrm{comp}(M, \rho) > 1$, then every hard set of request sequences needs at least 2 elements starting with the same request since otherwise, an online algorithm is not subject to any additional restrictions in comparison to the optimal offline algorithm.

When taking a closer look at ρ_h, we notice that the prefix forest belonging to ρ_h may consist of many trees instead of just a single one. This is due to several reasons that are explained in the next few lemmas. For the first lemma, we define the *scaling of a request sequence* $\bar{\sigma}$ with a factor $q \in \mathbb{R}^+$ denoted with $q \cdot \bar{\sigma}$ as the request sequence $\bar{\sigma}$ where the duration of each request is multiplied with q. We further define $q \cdot \rho = \{ q \cdot \bar{\sigma} \mid \bar{\sigma} \in \rho \}$ as a *scaled set of request sequences*.

▷ LEMMA 73 (SCALING LEMMA): (M, ρ) is c-competitive if, and only if, $(M, q \cdot \rho)$ is c-competitive.

Proof. Let $\mathcal{P}_\rho$ and $\mathcal{P}_{q\cdot\rho}$ be the prefix forests corresponding to the request sequence sets ρ and $q \cdot \rho$. There is a bijection between the trees of these forests where each tree corresponds to its scaled version.

Let P be any tree in $\mathcal{P}_\rho$ and $P_{q\cdot\rho}$ the corresponding scaled one. If there is an online algorithm $\mathcal{A}$ for $(M, \rho(P))$ with $\mathrm{comp}(\mathcal{A}) = c$, then we can derive an online algorithm $\mathcal{A}_q$ for $(M, P_{q\cdot\rho})$ that is c-competitive as well. On input $q \cdot \bar{\sigma}$, this algorithm simulates $\mathcal{A}$ on $\bar{\sigma}$ to obtain the flow sequence $\bar{f}$. It then scales this flow sequence with q to obtain the flow sequence $q \cdot \bar{f} = (q \cdot f_1, q \cdot f_2, \dots)$.

This suffices as for the other direction, the above argument works with the scaling factor $1/q$. Hence, (M, ρ) is c-competitive if, and only if, $(M, q \cdot \rho)$ is c-competitive. □

As the duration of a request is nothing else but the scaling of the network's capacities, this lemma also implies that if a network is c-competitive, then the same network with equally scaled edge capacities is also c-competitive.

Moreover, from the Scaling Lemma it follows that for an unrestricted dynamic storage system, ρ_h is an infinite set of request sequences containing infinitely many scaled copies of a single tree. But even a single hard tree might contain more request sequences than necessary. For example, a set of hard request sequences might contain request sequences that supply more flow than even an optimal offline algorithm can send to the demand nodes. Intuitively, it is clear that lowering the supply such that all flow can be sent to the demand nodes does not hurt the offline algorithm and should not help any online algorithm and hence yields another hard set of request sequences. The situation is quite similar with request sequences where the demand is so high that not even an optimal offline algorithm can fully satisfy it. Here, too, the demand can be lowered to still obtain a hard set of request sequences. This is formally stated in the next lemma.

▷ LEMMA 74: Let (M, ρ) be an unrestricted dynamic storage system and

$$\rho' = \{ \bar{\sigma} \in \rho \mid \mathrm{supply}(\bar{\sigma}) = \mathrm{opt}(\bar{\sigma}) = \mathrm{demand}(\bar{\sigma}) \}. \tag{137}$$

Then, $\mathrm{comp}(M, \rho) = \mathrm{comp}(M, \rho')$.

Proof. Since $\rho' \subseteq \rho$, we have $\mathrm{comp}(M, \rho') \leqslant \mathrm{comp}(M, \rho)$. In particular, this implies $\mathrm{comp}(M, \rho') = 1$ if $\mathrm{comp}(M, \rho) = 1$. For the remainder of the proof, we hence assume that $\mathrm{comp}(M, \rho) > 1$.

We say $\bar{\sigma} \sqsubseteq \bar{\sigma}'$ if $\bar{\sigma}$ and $\bar{\sigma}'$ are of the same length l, $\text{opt}(\bar{\sigma}) = \text{opt}(\bar{\sigma}')$, and for all $i \in \{1, 2, \dots, l\}$ we have $\tau(\sigma_i) = \tau(\sigma_i')$ as well as $\sigma_i(\nu) \leqslant \sigma_i'(\nu)$ for all $\nu \in S \cup D$. Now consider the set $\rho'' = \{\bar{\sigma}'' \sqsubseteq \bar{\sigma} \mid \bar{\sigma} \in \rho \text{ and } \bar{\sigma}'' \in \rho'\}$. By definition, $\rho'' \subseteq \rho'$. However, since for every request sequence $\bar{\sigma}' \in \rho'$, there is a request sequence $\bar{\sigma} \in \rho$ with $\bar{\sigma}' \sqsubseteq \bar{\sigma}$, we have $\rho'' = \rho'$.

Let $P = (V, E)$ be a hard tree of the prefix forest $\mathcal{P}_\rho$. Then, there is an isomorphic tree $P' = (V', E')$ of sequences in ρ' such that for each pair $(\bar{\sigma}, \bar{\sigma}')$ of isomorphic nodes with $\bar{\sigma} \in V$ and $\bar{\sigma}' \in V'$, the two request sequences satisfy $\bar{\sigma}' \sqsubseteq \bar{\sigma}$. If there is an online algorithm $\mathcal{A}'$ for (M, ρ') with $\text{comp}(\mathcal{A}') < c$, then we can use $\mathcal{A}'$ to construct an online algorithm $\mathcal{A}$ for (M, ρ) with $\text{comp}(\mathcal{A}) = \text{comp}(\mathcal{A}')$. When $\mathcal{A}$ receives the request sequence $\bar{\sigma}$, it identifies the isomorphic request sequence $\bar{\sigma}'$ of an isomorphic tree P' and simulates $\mathcal{A}'$ on it. This, however, yields $\text{comp}(\mathcal{A}) < \text{comp}(M, \rho)$ – a contradiction.

We may conclude that for an unrestricted dynamic storage system, in order to find a hard set of request sequences it suffices to consider those request sequences for which an optimal offline algorithm can send as much flow to the demand nodes as is supplied and demanded. □

This allows us to assume that all request sequences $\bar{\sigma}$ considered from now on satisfy $\text{supply}(\bar{\sigma}) = \text{opt}(\bar{\sigma}) = \text{demand}(\bar{\sigma})$.

We have now shown a number of ways to thin out a hard set of request sequences, but still have not arrived at the smallest one yet. In the following, we will show how to obtain a certain set of request sequences ρ_M for a given dynamic storage network M. We will later see that it is almost a smallest hard set of request sequences.

We can now argue that there always is a hard set of request sequences all beginning with a request of zero demand.

▷ LEMMA 75: Let (M, ρ) be an unrestricted dynamic storage system with $\text{comp}(M, \rho) > 1$. Then, there is a hard set of request sequences that all begin with a request of zero demand.

Proof. For the sake of contradiction, we assume to the contrary such that all hard sets of request sequences start with a request of positive demand. Let P be a hard tree of request sequences. By Lemma 74, we may assume that an optimal offline algorithm can send all supplied flow to sinks and satisfy all demand.

Let σ_1 be the first request of all sequences belonging to P. By Lemma 74, we have $\text{demand}(\sigma_1) \leqslant \text{supply}(\sigma_1)$. Moreover, if $\text{demand}(\sigma_1) = \text{supply}(\sigma_1)$, then P certainly is not a hard tree. Hence, $x = \text{supply}(\sigma_1) - \text{demand}(\sigma_1) > 0$ is the amount of flow that can be stored during the first request.

Now let us replace the first request of each sequence with a new request of zero demand and a total supply of x flow units. In combination with the duration, this request can be created in a way that allows any algorithm to arbitrarily store the x flow units at storage nodes. The remaining requests are left unchanged. Let us denote the corresponding tree by P'.

The sequences in P allow any online algorithm to send flow from the sources directly to the sinks in the first request as it has a positive demand. Up to this point, the competitiveness is 1. For an adversary seeking a high competitiveness, this can only pay off if the online algorithm is hindered to store flow in a better way which it could if there was no demand. However, as the optimal offline algorithm can still fully satisfy all demand, Lemma 70 implies that a compromise that lies in between the sequence of flows chosen by the optimal offline algorithm, is possible as well. Hence, the adversary can only lose by demanding flow at some demand nodes during the first request. □

Our search for a smallest hard set of request sequences can hence be focused on those requests that all start with a request just supplying flow. Due to the Scaling Lemma, we may assume that during this first request, exactly 1 unit of flow is supplied.

The next request of each sequence is thus a request draining the storage nodes. A request might force some storage node to release more flow than another. As soon as a storage node runs empty, while there still is a need to release more flow from it, the competitive ratio raises. In this case, the adversary continues requesting the same storage node to release flow which further increases the competitive ratio.

The adversary is hence looking for request sequences that stress the storage nodes in as different ways as possible. This search can be narrowed down to the vertices of the external flow pattern. These vertices correspond to flows, however, the adversary can only control the requests. These dictate a net flow at source nodes and at sink nodes. It is left to the online and offline algorithms to choose compliant flows. Depending on this choice, the net flow at the storage nodes can vary. Still, some requests can only be served by extracting a large amount of flow from a specific storage node. We now introduce some notions to formalize these ideas.

▷ DEFINITION 18 (REQUEST SPACE): The request space of a dynamic storage network M is

$$\begin{aligned}\sigma(M) = \{ & (f_1, f_2, \dots, f_{|S|}, \\ & f_{|S|+|R|+1}, f_{|S|+|R|+2}, \dots, f_{|S|+|R|+|D|}) \mid \vec{f} \in F_M \}. \end{aligned} \tag{138}$$

The request space is thus the external flow pattern restricted to those components which can be controlled by request sequences: the net flows of the supply nodes and the net flows of the demand nodes. By appropriately interpreting requests, that formally are functions over $S \cup D$, as vectors, we see that the request space is made up of all requests that could potentially be served.

▷ DEFINITION 19 (VERTEX REQUEST): A request $\sigma \in \sigma(M)$ for a dynamic storage network M is a *vertex request* if σ is a vertex in $\sigma(M)$, i.e., there are no two distinct requests in $\sigma(M)$ such that σ is a convex linear combination of these two. A *draining vertex request* is a vertex request for which the net flows of the demand nodes is greater than the net flows of the supply nodes. A *normalized draining vertex request* is a draining vertex request with a duration such that the demand is exactly 1 unit greater than the supply.

A draining vertex request can only be served by taking flow from the storage nodes. There always are draining vertex requests since every dynamic storage network has at least one storage node.

▷ DEFINITION 20 (FILLING REQUEST): For a dynamic storage network M, let $c_{\min}$ be the minimum capacity over all positive edge capacities. Then, the request $\sigma_{M,\text{fill}}$ with $\sigma_{M,\text{fill}}(s) = c_{\min}$ for a fixed supply node $s \in S$, $\sigma_{M,\text{fill}}(v) = 0$ for all $v \in S \cup D - s$, and $\tau(\sigma_{M,\text{fill}}) = 1/c_{\min}$ is the *filling request* of M.

A filling request has no demand, but a supply so low that it can be routed through any edge. This request has a duration such that the total amount of flow entering the network is 1. This 1 flow unit may be stored in an arbitrary way at the storage nodes, since the underlying graph is connected.

In the following, we link a flow f in M to the request σ_f of M by saying that a flow f is *compliant* to a request σ_f if $\tau(\sigma_f)\sigma_f(v) = |f(v)|$ for all $v \in S \cup D$. There may be several flows that are compliant to a given request.

▷ DEFINITION 21 (DRAINING VERTEX REQUEST SEQUENCES): Let M be a dynamic storage network. If ρ_d is the set of all normalized draining vertex requests for M, then we define

$$\rho_M = \left\{ \sigma_{M,\text{fill}}\sigma_f \mid \sigma_f \in \rho_d \right\} \tag{139}$$

as *the set of all vertex request sequences for* M.

Due to the previous discussions, we have $\text{comp}(M, \rho_M) = \text{comp}(M, \rho)$ for any unrestricted dynamic storage system (M, ρ). Hence, we now analyze the competitiveness of the dynamic storage system (M, ρ_M). While an offline algorithm may store the flow during the filling request in exactly the way it will be needed for the next request, any online algorithm is forced to react to the first request of every sequence in ρ_M with the same flow and thus store the provided flow in the same way at the storage nodes.

▷ DEFINITION 22 (STORAGE VECTOR): Let $\mathcal{A}$ be an online or offline algorithm for a dynamic storage system (M, ρ), and let $(r_1, r_2, \dots, r_{l+1})$ be the sequence of storage functions created by $\mathcal{A}$ in response to a request sequence $\bar{\sigma} = (\sigma_1, \sigma_2, \dots, \sigma_l)$. Further, let $R_i = \sum_{v \in R} r_i(v)$ be the total amount of flow stored in M after the i^{th} request. Then, we define the i^{th} *storage vector* $\vec{\alpha}_i = \big(r_i(v_1), r_i(v_2), \dots, r_i(v_{|R|})\big)$ and the *normalized* i^{th} *storage vector* to be $\vec{\alpha}_i/R_i$ if $R_i \neq 0$ and the 0-vector otherwise.

The storage vector describes the way in which $\mathcal{A}$ stores the flow at the storage nodes. The storage vector and normalized storage vector are equivalent after $\sigma_{M,\text{fill}}$ for optimal algorithms, as exactly 1 unit of flow is stored at the storage nodes.

While the optimal offline algorithm adapts the storage vector in response to $\sigma_{M,\text{fill}}$ depending on how the request sequence continues, an online algorithm always stores the flow provided during this first request in the same way. This may lead to situations where the online algorithm cannot satisfy all demands during the second request, see Figure 23. The flow not reaching any demand nodes is left at storage nodes by the end of the request sequence – it is easy to see that no online algorithm will ever have an advantage of not storing flow at storage nodes during the filling request. Moreover, for any draining vertex request, there is a flow independent of the current storage vector such that all flow supplied at supply nodes leaves the supply nodes. We may thus assume that all flow unable to be sent to demand nodes remains at storage nodes by the end of the request sequence. Let $\vec{\alpha}$ be some normalized storage vector and $M_{\vec{\alpha}}$ the dynamic storage network where the storage function corresponds to $\vec{\alpha}$. For any $\sigma_{M,\text{fill}}\sigma_f \in \rho_M$, let $d_{\vec{\alpha}}(\sigma_f)$ be the minimum amount of flow that is left at storage nodes if $\vec{\alpha}$ is the storage vector after $\sigma_{M,\text{fill}}$.

The competitiveness of the online algorithm $\mathcal{A}_{\vec{\alpha}}$ choosing $\vec{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_{|R|})$ as storage vector after $\sigma_{M,\text{fill}}$ is equal to the worst ratio over all requests in ρ_M of flow that can be sent to demand nodes by the offline algorithm divided by flow that can be sent to demand nodes by an optimal offline algorithm. More formally, we have

$$\text{comp}(\mathcal{A}_{\vec{\alpha}}) = \max\left\{ \frac{1 + \text{supply}(\sigma_f)}{1 + \text{supply}(\sigma_f) - d_{\vec{\alpha}}(\sigma_f)} \;\middle|\; \sigma_{M,\text{fill}}\sigma_f \in \rho_M \right\}. \tag{140}$$

To determine this competitiveness, we need to find the optimal storage ratio $\vec{\alpha}_{\text{opt}}$ that minimizes this term. This is

$$\begin{aligned}
\vec{\alpha}_{\text{opt}} &= \operatorname*{argmin}_{\vec{\alpha}} \operatorname{comp}(A_{\vec{\alpha}}) && (141)\\
&= \operatorname*{argmin}_{\vec{\alpha}} \max\left\{ \frac{1+\operatorname{supply}(\sigma_f)}{1+\operatorname{supply}(\sigma_f) - d_{\vec{\alpha}}(\sigma_f)} \;\middle|\; \sigma_{M,\text{fill}}\sigma_f \in \rho_M \right\} && (142)\\
&= \operatorname*{argmin}_{\vec{\alpha}} \min\left\{ \frac{1+\operatorname{supply}(\sigma_f) - d_{\vec{\alpha}}(\sigma_f)}{1+\operatorname{supply}(\sigma_f)} \;\middle|\; \sigma_{M,\text{fill}}\sigma_f \in \rho_M \right\} && (143)\\
&= \operatorname*{argmin}_{\vec{\alpha}} \min\left\{ 1 - \frac{d_{\vec{\alpha}}(\sigma_f)}{1+\operatorname{supply}(\sigma_f)} \;\middle|\; \sigma_{M,\text{fill}}\sigma_f \in \rho_M \right\} && (144)\\
&= \operatorname*{argmin}_{\vec{\alpha}} \max\left\{ \frac{d_{\vec{\alpha}}(\sigma_f)}{1+\operatorname{supply}(\sigma_f)} \;\middle|\; \sigma_{M,\text{fill}}\sigma_f \in \rho_M \right\}. && (145)
\end{aligned}$$

We take a closer look at $d_{\vec{\alpha}}(\sigma_f)$. It is defined as the minimum amount of flow that is left at the storage nodes after the request sequence $\sigma_{M,\text{fill}}\sigma_f$ beginning with a storage vector $\vec{\alpha}$. Since f may not be the only flow corresponding to σ_f, we define the *response space* $F(\sigma_f)$ of a draining vertex request σ_f as the set of flows compliant to the request σ_f. The vertices of this response space are exactly those draining vertex flows f' compliant to σ_f. Therefore,

$$d_{\vec{\alpha}}(\sigma_f) = \min\left\{ \sum_{v_i \in R} \max(\alpha_i - f(v_i), 0) \;\middle|\; f \in F(\sigma_f) \right\}. \qquad (146)$$

Since $\sum_{v_i \in R} \alpha_i = 1$ and $\sum_{v_i \in R} f(v_i) = 1$, we see that $\sum_{v_i \in R} \alpha_i - f(v_i) = 0$ which implies $\sum_{v_i \in R} \max(\alpha_i - f'(v_i), 0) = \sum_{v_i \in R} \frac{|\alpha_i - f'(v_i)|}{2}$. Hence,

$$d_{\vec{\alpha}}(\sigma_f) = \min\left\{ \sum_{v_i \in R} \frac{|\alpha_i - f'(v_i)|}{2} \;\middle|\; f' \in F(\sigma_f) \right\}. \qquad (147)$$

Thus, the objective is to find the α that minimizes

$$\max\left\{ \frac{\min\left\{ \sum_{v_i \in R} \frac{|\alpha_i - f'(v_i)|}{2} \;\middle|\; f' \in F(\sigma_f) \right\}}{1+\operatorname{supply}(\sigma_f)} \;\middle|\; \sigma_{M,\text{fill}}\sigma_f \in \rho_M \right\}. \qquad (148)$$

In order to simplify this, we make the assumption, that $|F(\sigma_f)| = 1$ for every $\sigma_{M,\text{fill}}\sigma_f \in \rho_M$. We arrive at

$$\operatorname*{argmin}_{\vec{\alpha}} \max\left\{ \frac{\sum_{v_i \in R} |\alpha_i - f'(v_i)|}{1+\operatorname{supply}(\sigma_f)} \;\middle|\; \sigma_{M,\text{fill}}\sigma_f \in \rho_M \right\}. \qquad (149)$$

To put some intuition into this, let us think about this geometrically. The objective is to find a vector $\vec{\alpha}$, whose sum of components is 1, such that the maximum distance to one of a set of given points is minimal. The set of given points corresponds exactly to the external flow patterns limited to the net flow at storage nodes of all normalized draining vertex requests. The distance to these points is measured with a scaled L_1 distance where each point has its own scaling factor. If $\vec{x}$ is a point in this space, then the distance of a draining vertex request σ_f to $\vec{x}$ is the L_1 distance between these two points scaled down by $1+\operatorname{supply}(\sigma_f)$ (actually by $1/2(1+\operatorname{supply}(\sigma_f))$ which can be neglected because we are only considering maxima and minima). The task thus is to find an $\vec{\alpha}$ that we call a *scaled center*.

The unit circle, that is the set of points reachable from the origin within a distance of 1, is a rotated rectangle for the L_1 distance. We rotate the axes by 45° in order to arrive at an axes parallel rectangle. This is achieved by transforming every point $\vec{x} = (x_1, x_2, \dots, x_{|R|})$ to the new point

$$\begin{aligned}\vec{x}' = \big(&x_1 + x_2 + x_3 + x_4 + \dots + x_{|R|}\,,\\ &x_1 - x_2 + x_3 + x_4 + \dots + x_{|R|}\,,\\ &x_1 + x_2 - x_3 + x_4 + \dots + x_{|R|}\,,\\ &x_1 + x_2 + x_3 - x_4 + \dots + x_{|R|}\,,\\ &\dots,\\ &x_1 + x_2 + x_3 + x_4 + \dots - x_{|R|}\big)\,. \end{aligned} \tag{150}$$

The L_1 distance now allows us to consider each dimension independently of the others to find for each dimension the two draining vertex requests such that their scaled center is a scaled center for all draining vertex requests in this dimension. So let us assume we have the two transformed vectors $\vec{x}'$ and $\vec{y}'$, the first with a scaling factor of $s_{x'}$, the second with a scaling factor of $s_{y'}$. Then, the scaled center for the i^{th} dimension is α'_i, where

$$\frac{1}{s_{x'}}\left|\alpha'_i - x'_i\right| = \frac{1}{s_{y'}}\left|\alpha'_i - y'_i\right|\,. \tag{151}$$

Just demanding the distances to be equal as in (151) allows two solutions due to the scaling. These are

$$\alpha'_i = \begin{cases} \frac{y'_i s_{x'} - x'_i s_{y'}}{s_{x'} - s_{y'}}\,; \\ \frac{y'_i s_{x'} + x'_i s_{y'}}{s_{x'} + s_{y'}}\,. \end{cases} \tag{152}$$

The upper solution may produce a scaled center not between the two given points. The second solution is the one producing a true center, i. e., a point such that the maximum scaled L_1 distance to the other points is minimal.

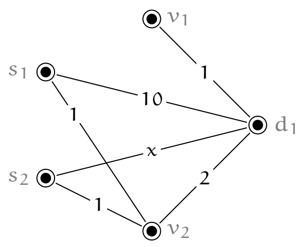
Therefore, we find α'_i for the i^{th} dimension by finding the scaled center for every pair of given points and then checking whether this scaled center is a scaled center for all points in that dimension.

The vector $\vec{\alpha}'$ obtained this way is then transformed by back-rotating the axes. There is an applied example of this technique in the proof of Lemma 80 on page 119.

Due to our assumption that $|F(\sigma_f)| = 1$, the competitive ratio obtained via this technique is only an upper bound. In the next section, we will analyze a given network via this technique for which the assumption is true and hence the derived competitive ratio is tight.

While this technique works well for small networks, its major drawback is that it needs to take into account all vertices of the request space. These may be up to $(|S| + |D|)!$ many – for each order of supply and demand nodes, we iteratively set the supplied or demanded amount to a maximum. Many of these vertices may be identical but in the worst case, they are distinct. Even if many of them are distinct, they still might not be relevant for the competitiveness of the network. Example 3 illustrates this difficulty of identifying a smallest hard set of request sequences.

▷ EXAMPLE 3: This example demonstrates the difficulty to identify those draining vertex requests that are solely responsible for the competitiveness of a network. For this, let $x \geqslant 1$, M the depicted dynamic storage network, and ρ_M the given set of all vertex request sequences (scaled up by a factor of 15 such that $\sigma_{M,\text{fill}} = (1,0\,|\,0)^{15}$).



ρ_M
$\bar{\sigma}_1 = \sigma_{M,\text{fill}}(0,0\,\|\,5)^3$
$\bar{\sigma}_2 = \sigma_{M,\text{fill}}(11,0\,\|\,14)^5$
$\bar{\sigma}_3 = \sigma_{M,\text{fill}}(0,1+x\,\|\,4+x)^5$
$\bar{\sigma}_4 = \sigma_{M,\text{fill}}(11,1+x\,\|\,13+x)^{15}$

It can be shown that the best online algorithm never stores less than 3 or more than 5 flow units at v_1, independent of x. Knowing this, we find the number of flow units left in the network by the end of each of the four request sequences. Let $\vec{\alpha}_{\text{opt}}$ be the optimal offline storage vector after $\sigma_{M,\text{fill}}$ and $\vec{\alpha} = (\alpha_1, \alpha_2)$ the storage vector of the online algorithm after $\sigma_{M,\text{fill}}$, where α_1 is the amount of flow stored at v_1 with $3 \leqslant \alpha_1 \leqslant 5$.

Request sequence	$\vec{\alpha}_{\text{opt}}$	Flow left in M depending on α_1	Competitive ratio depending on α_1
$\bar{\sigma}_1$	$(3/15, 12/15)$	$\alpha_1 - 3$	$\frac{15}{18-\alpha_1}$
$\bar{\sigma}_2$	$(5/15, 10/15)$	$5 - \alpha_1$	$\frac{70}{65+\alpha_1}$
$\bar{\sigma}_3$	$(5/15, 10/15)$	$5 - \alpha_1$	$\frac{20+5x}{15+5x+\alpha_1}$
$\bar{\sigma}_4$	$(15, 0)$	$15 - \alpha_1$	$\frac{195+15x}{180+15x+\alpha_1}$

The minimum of the maxima of these functions depends on x and, more importantly, the set of vertex request sequences that suffice to show the competitiveness depends on x. Moreover, two of these vertex request sequences always suffice to show competitiveness. For $x < 20$, it is σ_1 and σ_4. For $x > 20$, it is σ_1 and σ_2. It never is σ_3. For $x = 20$, it is σ_1 together with σ_2 or σ_4.

Value of x	Hard sets of request sequences
< 20	$\{\bar{\sigma}_1, \bar{\sigma}_4\}$
$= 20$	$\{\bar{\sigma}_1, \bar{\sigma}_4\}$ and $\{\bar{\sigma}_1, \bar{\sigma}_2\}$
> 20	$\{\bar{\sigma}_1, \bar{\sigma}_2\}$

5.4 COMPETITIVE RATIOS OF SOME NETWORK CLASSES

In this section, we analyze the competitive ratios of certain classes of dynamic storage networks. We group networks according to the number of supply, storage, and demand nodes. The class of (n_S, n_R, n_D) dynamic storage networks are those with exactly n_S supply nodes, n_R supply nodes, and n_D demand nodes.

The problem an online algorithm faces is the question where to store excess flow and which storage node to tap at times where the supply does not suffice. Hence, for any kind of network with just a single storage node, the competitivity is 1. Starting with 2 storage nodes, the competitiveness may become worse – and it actually does as we will show shortly. We consider the simplest type of dynamic storage networks with two storage nodes – the one with a single source and a single sink. We will identify a network with highest competitive ratio in this class, a *hard network* of $(1, 2, 1)$.

But before we start looking for such lower bounds on the competitive ratio, it pays off to think about upper bounds as well.

▷ LEMMA 76: Let M be a dynamic storage network with n_R storage nodes. Then, we have $\text{comp}(M) \leqslant n_R$.

Proof. An online algorithm that stores flow as evenly as possible at all storage nodes achieves the desired competitiveness. Suppose the request sequences start with a filling request in which n flow units enter the network. Further, assume that each storage node stores one unit of flow.

If a storage node then runs empty during a draining request and all other storage nodes cannot help, then the flow that has been stored at these other storage nodes remains in the network. These are $n-1$ flow units. In this case, the competitive ratio is at most

$$\frac{n}{n-(n-1)} = n\,. \tag{153}$$

If no online algorithm is able to store flow that evenly at the storage nodes, then no offline algorithm will be able to do it either. Therefore, this bound is an upper bound. □

Such a network may have an arbitrary number of non-terminals. We will put to use what we have learned from mimicking networks and show that there is a hard $(1, 2, 1)$ network that contains at most a single non-terminal. Similar to the concept of mimicking networks, we wish to find a dynamic storage network that mimicks the behavior of another dynamic storage network.

▷ DEFINITION 23 (MIMICKING DYNAMIC STORAGE NETWORK): Let M, M' be two dynamic storage networks. M' is a *mimicking dynamic storage network* of M if $F_M = F_{M'}$.

For a dynamic storage network $M = (V, E, S, R, D, c)$, let $N = (V, E, S \cup R \cup D, c)$ be the *underlying multi-terminal network*. For the opposite direction, we say that M *belongs* to N. Now let M' be a dynamic storage network belonging to a network that mimicks the underlying multi-terminal network of M. Then, it is quickly seen that M' is a mimicking dynamic storage network of M.

▷ LEMMA 77: Let M be a dynamic storage network and N the underlying multi-terminal network. If N' is a mimicking network of N, then M' belonging to N' is a mimicking dynamic storage network of M.

Proof. Let f be a flow in M with the external flow $\vec{f}$. Then, $\vec{f}$ is also an external flow in N. Since N' is a mimicking network of N, $\vec{f}$ is an external flow of N' as well. When reassigning the corresponding roles to the terminals to obtain M', $\vec{f}$ still is an external flow. The same arguments hold for the opposite direction. □

One motivation behind mimicking dynamic storage networks is that they have the same competitiveness as the network they mimick.

▷ LEMMA 78: Let M, M' be two mimicking dynamic storage networks. Then $\text{comp}(M) = \text{comp}(M')$.

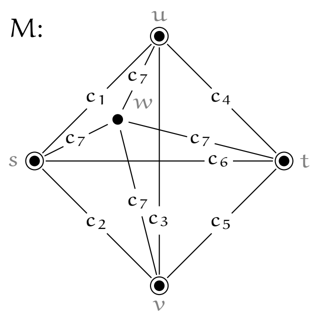


Figure 25: According to Chaudhuri et al. [12], there is a unrestricted mimicking network of size 5 for every network with 4 terminals such that the only non-terminal is connected to all four terminals with edges of equal capacity. This also holds true for dynamic storage networks and, as in this case, for $(1,2,1)$ networks.

Proof. Let us assume that M is c-competitive. Then there is a c-competitive online algorithm $\mathcal{A}$. We can use $\mathcal{A}$ to derive a c-competitive online algorithm $\mathcal{A}'$ for M'. For every request sequence that $\mathcal{A}'$ receives, it simulates the same request sequence with $\mathcal{A}$ on M. The sequence of flows generated by $\mathcal{A}$ for M is then transformed into a sequence of corresponding external flows. Then $\mathcal{A}'$ finds for each external flow of this sequence of flow. Such a flow exists since M and M' are mimicking dynamic storage networks. Algorithm $\mathcal{A}$ is c-competitive because the optimal offline algorithm cannot perform better on M' than on M and because $\mathcal{A}'$ sends the same amount of flow to demand nodes as does $\mathcal{A}$. □

The property of having the same competitiveness stretches across mimicking dynamic storage networks. Two networks that do not share the same number of storage nodes certainly can still share the same competitiveness. While the techniques learned from mimicking networks in the previous chapter do not allow to merge terminals with each other, the next lemma describes when this is possible for dynamic storage networks without affecting the competitiveness.

▷ LEMMA 79: Let M be a dynamic storage network and $v_1, v_2 \in R$ two distinct storage nodes. If $c_{M,\{v_1,v_2\}} \leqslant c_{M,\{v_1\}}$ then merging v_1, v_2 to v_3 yields a dynamic storage network M' with $\text{comp}(M) = \text{comp}(M')$.

Proof. Let (S,T) be a $\{v_1, v_2\}$-min-cut in M with $v_1, v_2 \in S$. As $c_{M,\{v_1,v_2\}} \leqslant c_{M,\{v_1\}}$, we know that any amount of flow that enters S from outside S can be either sent completely to v_1 or completely to v_2. The symmetric argument holds for flow leaving S. Hence, any optimal online algorithm $\mathcal{A}$ for M can be altered to never store any flow at v_1 without losing its competitiveness. If v_1 is never used as storage node, then we could simply demote it to be a non-terminal. If so, then Theorem 59 on page 86 tells us in combination with Lemma 78 that merging v_1, v_2 yields a mimicking dynamic storage network M' with $\text{comp}(M) = \text{comp}(M')$. □

So if, for example, two storage nodes are joined by an edge such that this edge dominates the two storage nodes, then at least one of them does not have to be ever used.

The described effect that a storage node is useless in the sense that all flow that is stored at it could just as well be stored at another storage node does not suddenly arise. If two storage nodes are joined by an edge whose

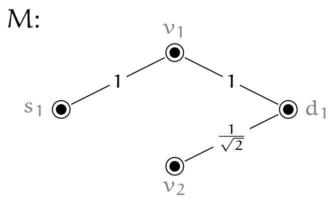


Figure 26: This network is a hard $(1, 2, 1)$ dynamic storage network. Any online algorithm needs to decide where to store the excess flow – at v_1 or at v_2. The storage node v_1 can be completely blocked by flow from s_1 to d_1. However, if d_1 demands $1 + 1/\sqrt{2}$ during a single time unit, then flow must have been stored at v_1. The best compromise is to store approximately 41% of the excess flow at v_1.

capacity is not large enough to dominate either one of the two nodes, then, still, the storage nodes can at least partly replace each other. This leads to the fact that up to a certain amount it does not matter where to store the excess flow. Under this perspective, it is not surprising that hard networks of a class usually do not contain any edges joining their storage nodes. Moreover, networks that do have edges joining their storage nodes are networks for which the response spaces to requests do not contain single flows.

COMPETITIVENESS OF $(1,n_R,1)$ NETWORKS. We show that any hard dynamic storage network with just a single supply node and a single demand node is $(1+\sqrt{2})/2$-competitive.

▷ LEMMA 80: There is a $(1+\sqrt{2})/2$-competitive $(1, 2, 1)$ dynamic storage network M.

Proof. Let M be the network of Figure 26. Then, $\rho_M = \{\bar{\sigma}_1, \bar{\sigma}_2\}$ is the set with

$$\bar{\sigma}_1 = \sigma_{M,\text{fill}}\left(0 \,|\, 1 + \frac{1}{\sqrt{2}}\right)^{2-\sqrt{2}}, \tag{154}$$

$$\bar{\sigma}_2 = \sigma_{M,\text{fill}}\left(1 \,|\, 1 + \frac{1}{\sqrt{2}}\right)^{\sqrt{2}}. \tag{155}$$

The first request, the filling request, allows the online algorithm to arbitrarily store 1 unit of flow at the storage nodes. The following draining vertex requests demand a total of 1 flow units to be released from the storage nodes.

The storage vectors of an optimal offline algorithm are

$$\vec{s}_1 = \left(\frac{1}{1+\frac{1}{\sqrt{2}}}, \frac{\frac{1}{\sqrt{2}}}{1+\frac{1}{\sqrt{2}}}\right) \tag{156}$$

$$= \left(2-\sqrt{2}, \sqrt{2}-1\right) \approx (0.59, 0.41) \text{ and} \tag{157}$$

$$\vec{s}_2 = (0, 1) \tag{158}$$

for $\bar{\sigma}_1$ and $\bar{\sigma}_2$, respectively. With these storage vectors, an optimal offline algorithm is able to send all supplied flow to the demand nodes and satisfy all demand. The response spaces to the two draining vertex requests in ρ_M each contain a single flow. The external flow of these limited to the net flows at the terminals is exactly $\vec{s}_1$ and $\vec{s}_2$, respectively.

An optimal online algorithm uses the storage vector $\vec{\alpha}$ according to (149). We transform the vectors by an axes rotation and obtain

$$\vec{s}_1' = \left(1, 3 - 2\sqrt{2}\right) \text{ and} \tag{159}$$

$$\vec{s}_2' = (1, -1), \tag{160}$$

where the scaling factor of the first is 1 and of the second $1+\sqrt{2}$. Hence, by (152), we have

$$\alpha_1' = \frac{(1+\sqrt{2})+1}{2+\sqrt{2}} = 1 \text{ and} \tag{161}$$

$$\alpha_2' = \frac{(1+\sqrt{2})(3-2\sqrt{2})-1}{2+\sqrt{2}} \tag{162}$$

$$= 2\sqrt{2}-3. \tag{163}$$

Now, we need to back-rotate these vectors and hence, the sought vector $\vec{\alpha} = (\alpha_1, \alpha_2)$ is the one with

$$\alpha_1 + \alpha_2 = 1 \text{ and} \tag{164}$$

$$\alpha_1 - \alpha_2 = 2\sqrt{2}-3, \tag{165}$$

which is $\vec{\alpha} = (\sqrt{2}-1, 2-\sqrt{2}) \approx (0.41, 0.59)$. Knowing this, we deduce that for $sig\bar{m}a_1$, the algorithm leaves $|2-\sqrt{2}-(\sqrt{2}-1)| = 3-2\sqrt{2}$ flow units in the network. This leads to a competitive ratio of

$$\text{comp}(M) = \frac{1}{1-(3-2\sqrt{2})} = \frac{1+\sqrt{2}}{2}. \tag{166}$$

□

So there is a $(1,2,1)$ dynamic storage network M with a competitive ratio of $(1+\sqrt{2})/2$. For this network the optimal online algorithm stores approximately 41% of the flow supplied during the filling request at the one storage node (v_1 in case of Figure 26) and the remaining flow at the other storage node. But is there a dynamic storage network M' with a higher competitive ratio? Any smallest hard set of request sequences for such a network contains exactly two sequences since there are two draining vertex requests. Let us call them $\bar{\sigma}_1$ and $\bar{\sigma}_2$. Intuitively, one of them makes the optimal offline algorithm store a lot of flow at v_1, while the other makes the optimal offline algorithm store a lot of flow at v_2. This difference can only be achieved by supplying f_2 additional flow units during the draining request of $\bar{\sigma}_2$. The effect of this on the competitive ratio is maximized when the additional supply blocks the flow stored at v_1 completely. We thus assume that the optimal offline algorithm stores no flow units at v_1 during $\bar{\sigma}_2$ and x_1 units at v_1 during $\bar{\sigma}_1$.

The online algorithm stores α_1 units at v_1 and $1-\alpha_1$ at v_2 with $0 < \alpha_1 < 1$. By the end of $\bar{\sigma}_1$, $(1-\alpha_1)-(1-x_1)$ units are left at v_2 and none at v_1. By the end of $\bar{\sigma}_2$, α_1 units are left at v_1 and none at v_2. Hence, the competitive ratio is the minimum of

$$c_1 = \frac{1}{1-\big((1-\alpha_1)-(1-x_1)\big)} = \frac{1}{1+\alpha_1-x_1} \text{ and} \tag{167}$$

$$c_2 = \frac{1+s_2}{1+s_2-\alpha_1}. \tag{168}$$

If there is a $(1,2,1)$ network with a competitive ratio greater than $(1+\sqrt{2})/2$, then $c_1 > (1+\sqrt{2})/2$ and $c_2 > (1+\sqrt{2})/2$. This implies that

$$x_1 - 1 < \alpha_1 < \frac{\sqrt{2}x_1 + x_1 - \sqrt{2} + 1}{1+\sqrt{2}} \quad \text{and} \tag{169}$$

$$\frac{\sqrt{2}f_2 - f_2 + \sqrt{2} - 1}{1+\sqrt{2}} < \alpha_1 < f_2 + 1 \,. \tag{170}$$

Such α_1 may thus only exist if

$$(3-2\sqrt{2})f_2 < x_1 + 2(2\sqrt{2}-3)\,. \tag{171}$$

Note that during $\bar{\sigma}_1$ and $\bar{\sigma}_2$, different amounts of flow are sent to the demand node. This can only happen if they have different durations. If the durations of $\bar{\sigma}_1$ and $\bar{\sigma}_2$ are τ_1 and τ_2, respectively, then $f_2/\tau_2 = x_1/\tau_1$. The ratio of the two durations is the ratio of the flow that reaches the demand node and thus $1/1+f_2$. Hence, we have

$$x_1 = \frac{\tau_1}{\tau_2} f_2 = \frac{f_2}{1+f_2}\,. \tag{172}$$

Combining the two latter formulas yields

$$(3-2\sqrt{2})f_2 < \frac{f_2}{1+f_2} + 2(2\sqrt{2}-3)\,, \tag{173}$$

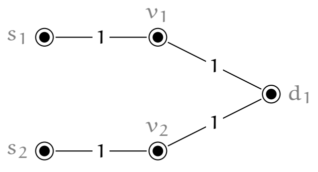
which is not fulfilled for any positive f_2. Is it fulfilled with equality instead of inequality for values that yield the network of Figure 26.

In fact, these considerations work for any dynamic storage network M with one supply and one demand node. The set of storage nodes can be partitioned into two sets of those nodes that need to store a lot of flow during $\bar{\sigma}_1$ or during $\bar{\sigma}_2$. This leads to the following theorem.

▷ THEOREM 81: For every $(1, n_R, 1)$ dynamic storage network M it holds that $\text{comp}(M) \leqslant \frac{1+\sqrt{2}}{2}$.

Note that this particular competitive ratio depends on irrational capacities in the network. From a computational point of view where the encoding matters, this is still of interest as it still as an upper bound. If the precision of the encoded numbers is increased to infinity, then the upper bound is met in the limit. In the following, we will see some more networks where ratios can only be achieved in the limit.

COMPETITIVENESS OF (2,2,1) NETWORKS. Let M be the following network together with the set of request sequences ρ.



ρ
$\bar{\sigma}_1 = (1,1\,\|\,0)(0,1\,\|\,2)^2$
$\bar{\sigma}_2 = (1,1\,\|\,0)(1,0\,\|\,2)^2$

For $\bar{\sigma}_1$, the optimal offline algorithm stores all flow at v_1 during the first request. During the following request, it can satisfy the demand of 2 by one unit from v_1 and one unit from the supply node. For $\bar{\sigma}_2$, the optimal offline algorithm stores all flow at v_2 during the first request and then behaves symmetrically.

The optimal online algorithm stores 1 unit at v_1 and 1 unit at v_2. During the draining requests, it will only be able to send 3 of the demanded 4 units of flow to the demand node.

The competitive ratio of M thus is $4/3$. This is a lower bound on the competitiveness of $(2,2,1)$ networks – possibly there is another $(2,2,1)$ network causing a higher competitive ratio. By Lemma 76, 2 is an upper bound on the competitive ratio.

COMPETITIVENESS OF $(n,n,1)$ NETWORKS. As we were able to find a network with a competitive ratio of $4/3$ for a $(2,2,1)$ network, a natural question is how this result can be generalized for $(n,n,1)$ networks. Let the network look like the one in the section before, but generalized for n supply and n storage nodes. Consider request sequences that allow the network to store n units of flow in an arbitrary way at the n storage nodes. Then, for every $0 \leqslant i < n$, there are $\binom{n}{i}$ sequences that continue with a request with a supply of i units and a demand of n units where there are exactly i sources with a supply of 1 unit. Such a request has a duration of $n/(n-i)$.

The optimal offline algorithm stores $n/(n-i)$ units of flow at those $n-i$ nodes that are not blocked by the additional supply during the draining request. The vectors describing the amount of flow that is drained from the storage nodes contain a 0 if the corresponding storage node is blocked or a n/i otherwise.

The maximum amount of flow that the optimal offline algorithm stores at a storage node is n if all other nodes are blocked $(i = n-1)$. The minimum is 0 if that node is blocked. So the optimal online algorithm has to do something in between and due to symmetry, the optimal online algorithm stores 1 unit at each of the storage nodes.

For i, we have the competitive ratio

$$c_i = \frac{n + \frac{n}{n-i} i}{n + \frac{n}{n-i} i - i} \tag{174}$$

$$= \frac{1}{1 - \left(\frac{i}{n} - \frac{i^2}{n^2}\right)} . \tag{175}$$

This is due to the following considerations. The optimal offline algorithm sends the n units of flow to the sink that have been stored at the beginning. It also sends the $i \cdot n/n-i$ units of flow to the sink that are supplied during the draining request. The optimal online algorithm, however, cannot send the amount of flow to the demand node that is stored at the blocked storage nodes – this is i units.

For a fixed n, the competitive ratio of the above network for the described request sequences is the maxima of c_i for $0 \leqslant i < n$. This is reached where $\frac{i}{n} - \left(\frac{i}{n}\right)^2$ reaches its maximum. For a given n, this is the case where the derivative equals zero, i. e.,

$$\frac{n-2i}{n^2} = 0 \tag{176}$$

$$\iff i = \frac{n}{2} . \tag{177}$$

So the hard set of request sequences contains those sequences where $n/2$ of the storage nodes are blocked by additional supply. In this case, the competitive ratio is

$$c_{n/2} = \frac{1}{1 - \left(\frac{n/2}{n} - \frac{(n/2)^2}{n^2}\right)} \tag{178}$$

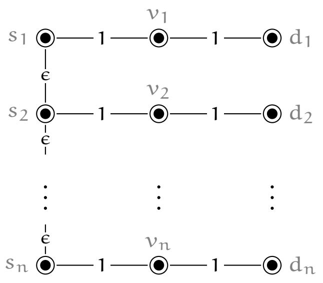
$$= \frac{1}{1 - \left(\frac{1}{2} - \frac{1}{4}\right)} \tag{179}$$

$$= \frac{4}{3}\,. \tag{180}$$

Hence, the networks with the described request sequences are $4/3$-competitive for all n, at least for all even n. For odd n it is slightly below $4/3$ but for n approaching positive infinity, the competitive ratio approaches $4/3$. To be exact, for odd n, the competitive ratio is $\frac{4n^2}{3n^2+4}$.

One might now expect that a smallest hard set of request sequences for M needs to contain $\binom{n}{\lfloor \frac{n}{2} \rfloor} \in \Theta\left(\frac{4^n}{\sqrt{n}}\right)$ request sequences. However, a mere two request sequences suffice where the one request sequence blocks the first half of the storage nodes and the other request sequence blocks the second half of the storage nodes.

COMPETITIVENESS OF (n,n,n) NETWORKS. Let (M, ρ) be the depicted dynamic storage system. We will analyze its competitive ratio for ϵ approaching 0.



ρ contains a $\bar{\sigma}_i$ for each $1 \leqslant i \leqslant n$
where $\bar{\sigma}_i = \sigma_{\text{fill}}\sigma_i$ with
$\sigma_{\text{fill}} = (\epsilon, 0, \ldots, 0)^{1/\epsilon}$ and
$\sigma_i(s_j) = 0$ for all $1 \leqslant j \leqslant n$ and
$\sigma_i(d_j) = 0$ for all $1 \leqslant j \leqslant n$, $j \neq i$ and
$\sigma_i(d_i) = 1$.

The request sequences allow the algorithms to store 1 unit arbitrarily in the network in a filling request. In a second request, there is no supply and demand of 1 at only one demand node. The competitive ratio is then

$$\rho = \frac{1}{1 - \left(1 - \frac{1}{n} - 2\epsilon\right)} \tag{181}$$

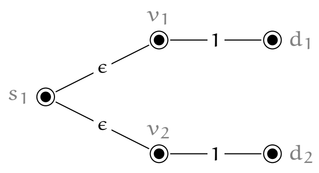
$$= \frac{n}{n - (n - 1 - 2\epsilon n)} \tag{182}$$

$$= \frac{n}{1 + 2\epsilon n}\,, \tag{183}$$

which approaches n for ϵ approaching 0.

Clearly, n is also an upper bound for the competitiveness of (n, n, n) networks. A smallest set of hard request sequences for (M, ρ) requires n sequences.

COMPETITIVENESS OF (1,2,2) NETWORKS. As for all networks with n_R storage nodes, the upper bound of the competitiveness is n_R. In this case, this upper bound is reached. The following network is a hard $(1, 2, 2)$ network.



ρ
$\bar{\sigma}_1 = (\epsilon \mid 0,0)^{1/\epsilon}(0 \mid 1,0)$
$\bar{\sigma}_2 = (\epsilon \mid 0,0)^{1/\epsilon}(0 \mid 0,1)$

Clearly, the optimal offline algorithm is able to fully satisfy the demand. The optimal online algorithm, however, splits the flow and stores 1/2 at each of the storage nodes. For each of $\bar{\sigma}_1$ and $\bar{\sigma}_2$, the optimal online algorithm can send $1/2 + \epsilon$ units of flow to the demand node. Hence, the competitive ratio is

$$\rho = \frac{1}{\frac{1}{2} + \epsilon} = \frac{2}{2\epsilon + 1}. \tag{184}$$

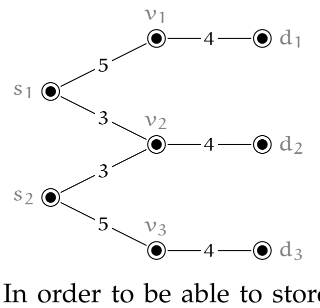
For ϵ approaching 0, the competitive ratio approaches 2.

COMPETITIVENESS OF $(1,n,n)$ NETWORKS. Let the dynamic storage network M look like the hard $(1,2,2)$ network generalized for $(1,n,n)$ networks. The request sequences begin with an identical filling request in which the algorithms may store 1 flow unit in an arbitrary way at the storage nodes. There are n request sequences. For each $1 \leqslant i \leqslant n$, there is a request sequence $\bar{\sigma}_i$ whose second request demands 1 unit for a duration of 1 at node d_i. There is no supply in this second request. The competitive ratio is then

$$\rho = \frac{1}{1 - \left(1 - \frac{1}{n} - \epsilon\right)} \tag{185}$$
$$= \frac{n}{1 + 1\epsilon}. \tag{186}$$

For ϵ approaching 0, the competitive ratio approaches n.

CONSTANT DEMAND AND CONSTANT SUPPLY. Even in the case that either the supply or the demand of a request sequence is constant and known ahead to the online algorithm, the competitive ratio can be greater than 1. Let $\mathcal{M}_d = (M_d, \rho_d)$ be a dynamic storage system where ρ_d has a constant demand of 4 in every request for every demand node, where M_d is the following dynamic storage network and ρ_d the following set of request sequences.



ρ_d
$\bar{\sigma}_1 = \big((7,6\vert 4,4,4)^5, (7,0\vert 4,4,4)\big)$
$\bar{\sigma}_2 = \big((7,6\vert 4,4,4)^5, (0,7\vert 4,4,4)\big)$

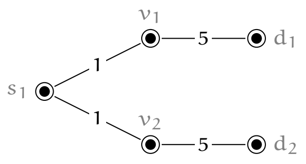
In order to be able to store flow during the first request, there needs to be more supply than demand. The storage functions of an optimal offline algorithm have to be different after this first request in order to be able to serve the different demands of the two sequences. During the first request, a total of five flow units can be stored in an arbitrary way at the storage nodes. In the last request of the first sequence, the supply is $(7,0)$. This blocks all flow that has prior been stored at v_1 and thus renders it useless. At the same time, at least one unit must have been stored at v_3 to fully satisfy the

Network class	Competitiveness
$(1,n,1)$	$= (1+\sqrt{2})/2$
$(2,2,1)$	$\geqslant 4/3$
$(n,n,1)$	$\geqslant 4/3$, even n
	$\geqslant 4n^2/(3n^2+4)$, odd n
(n,n,n)	$= n$
$(1,n,n)$	$= n$

Table 3: This table lists our results concerning the competitiveness of several network classes. By Lemma 76, every dynamic storage network with n_R storage nodes has a competitive ratio of at most n_R. This upper bound is met by networks that have n_R storage nodes, n_R demand nodes, but only one supply node. For $(n,n,1)$ dynamic storage networks, however, it seems not to be possible to construct a network with a competitive ratio above $4/3$.

demand. The same reasoning applies symmetrically to the second request sequence, where at least one unit of flow has to be stored at v_1. Therefore, the competitiveness is truly greater than 1.

For constant supply, let $\mathcal{M}_s = (M_s, \rho_s)$ be a dynamic storage system, where M_s and ρ_s are as depicted.



ρ_s
$\bar{\sigma}_1 = \big((1\,\vert\,0,0)^4, (1\,\vert\,5,0)\big)$
$\bar{\sigma}_2 = \big((1\,\vert\,0,0)^4, (1\,\vert\,0,5)\big)$

Due to symmetry, the best online strategy is to equally distribute flow at the storage nodes. However, it is easily seen that the competitive ratio is greater than 1 in this case. In fact, the competitive ratio becomes the greater the larger the capacities from v_i to d_i, the smaller the capacities from s_i to v_i, and the more storage nodes v_i are added following the same scheme and idea of M_s.

Concluding, we can sum up the results of this section concerning competitive ratios as in Table 3. While the upper bound of n can easily be reached for those network classes containing not only n storage nodes but also n demand nodes, this seems not possible for $(n,n,1)$ networks. In fact, we are only able to show a competitive ratio of $4/3$ for those networks, i. e., a constant competitive ratio. Similarly, with constant demand it seems to be harder to construct a network with a high competitive ratio than it is for constant supply. Hence, the demand seems to play a more important role for the competitive ratio than the supply.

6 CONCLUSION

In this thesis, we have developed the new model of dynamic storage networks and have derived a technique for the online analysis of them. Central to this technique are insights from the field of mimicking networks. A better understanding of them also deepens the understanding of dynamic storage networks.

We have pointed out drawbacks of the unique min-cuts assumption commonly made in conjunction with mimicking networks and have studied the complexity of finding a smallest contraction-based mimicking network without the unique min-cuts assumption. We have linked unresolved questions to the set of all min-cuts of a network.

To close in on these unresolved questions, we have studied the set of all min-cuts of a network. By redefining well-known NP-complete problems, whose inputs contain a set system, by replacing these set systems by flow networks, we have developed a new way of studying the restrictions imposed on set systems encoded by a flow network as the set of all min-cuts represented by the edges cut.

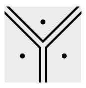
6.1 OUTLOOK

This section revisits some of the unanswered questions and offers some ideas how to approach them.

CHAPTER 3 – MIN-CUT SET SYSTEMS. One of the most important and yet unanswered questions of this chapter concerns the characterization of edge and node systems of multi-terminal networks. While we understand them quite well for the 2-terminal case, their dependencies in the multi-terminal case remain open. The lack of understanding these dependencies is the root for the lack of understanding other unanswered problems not only in this dissertation. In particular, a deeper understanding of this question is key to finding tighter bounds for the size of smallest mimicking networks and to finding an algorithm that computes a smallest contraction-based mimicking network for a given one.

CHAPTER 4 – MIMICKING NETWORKS. As shown, questions that remained open in this chapter are closely related to open questions of the previous chapter. Since MIN-CUT MINIMUM TYPE SELECTION and MCBMN are equal, settling the question of the complexity of MCBMN could yield valuable information about the min-cut set systems.

The algorithm presented at the end of Chapter 4 attempts to answer this question. However, it fails for a few instances making these instances promising candidates for instances that cannot be solved by any FPT algorithm. One of them can be depicted as on the right. It is remindful of the MULTICUT problem which asks whether there is a set of edges of total weight at most k such that deleting these edges pairwise disconnects a set of l given nodes. For $l \geqslant 3$, this problem is known to be NP-complete [17]. Even though we have been unsuccessful, this still might be a promising approach.



CHAPTER 5 – ONLINE ANALYSIS OF DYNAMIC STORAGE NETWORKS. An interesting addition to our model of dynamic storage networks would be non-zero transit times. In such a model, flow would not instantaneously travel through an edge, but take time to do so. Transit times are an aspect that has been examined in other contexts, and they are a generalization to our model making it applicable to more scenarios. For example, idling self-driving cars should idle at places that make them easily accessible even though future requests are unknown. Since they do not travel along streets at infinite speed, the augmented model fits better than its original version.

But also for many other kinds of generalizations that have been applied to the very basic model of flow networks, it is conceivable to apply them to our dynamic storage networks as well. Lossy transition or lossy storage sites could be used to more realistically model power grids.

Moreover, the profit function we used simply measures the amount of flow reaching demand nodes. One might want to enforce more fairness by, for example, defining a profit function that maps a flow to the minimum ratio of satisfied demand over all demand nodes.

BIBLIOGRAPHY

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. "Some Recent Advances in Network Flows." In: *SIAM review* 33.2 (June 1991), pp. 175–219. DOI: 10.1137/1033048.

[2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, And Applications*. Prentice-Hall, Inc., 1993.

[3] Susanne Albers. "Online Algorithms: A Survey." In: *Mathematical Programming, Series B* 97.1 (Aug. 2003), pp. 3–26. DOI: 10.1007/s10107-003-0436-0.

[4] Massoud Amin and John Stringer. "The Electric Power Grid: Today and Tomorrow." In: *Materials Research Society Bulletin* 33.4 (2008), pp. 399–407. DOI: 10.1557/mrs2008.80.

[5] Srinivasa R. Arikati, Shiva Chaudhuri, and Christos D. Zaroliagis. "All-Pairs Min-Cut in Sparse Networks." In: *Journal of Algorithms* 29.1 (Oct. 1998), pp. 82–110. DOI: 10.1006/jagm.1998.0961.

[6] Jay E. Aronson. "A survey of dynamic network flows." In: *Annals of Operations Research* 20.1 (1989), pp. 1–66. DOI: 10.1007/BF02216922.

[7] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[8] Béla Bollobás. *Modern Graph Theory*. Springer, 2013.

[9] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[10] Erin Chambers and David Eppstein. "Flows in One-Crossing-Minor-Free Graphs." In: *Journal of Graph Algorithms and Applications* 17.3 (2013), pp. 201–220. DOI: 10.7155/jgaa.00290.

[11] M. Charikar, T. Leighton, Shi Li, and A. Moitra. "Vertex Sparsifiers and Abstract Rounding Algorithms." In: *51st Annual IEEE Symposium on Foundations of Computer Science*. 2010, pp. 265–274. DOI: 10.1109/FOCS.2010.32.

[12] S. Chaudhuri, K. V. Subrahmanyam, F. Wagner, and C. D. Zaroliagis. "Computing Mimicking Networks." In: *Algorithmica* 26.1 (Jan. 2000), pp. 31–49. DOI: 10.1007/s004539910003.

[13] Julia Chuzhoy. "On Vertex Sparsifiers with Steiner Nodes." In: *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*. 2012, pp. 673–688. DOI: 10.1145/2213977.2214039.

[14] Vašek Chvátal. *Linear Programming*. New York, San Francisco: W. H. Freeman & Co., 1993.

[15] Eleonor Ciurea and Laura Ciupală. "Sequential and parallel algorithms for minimum flows." In: *Journal of Applied Mathematics and Computing* 15.1–2 (Apr. 2004), pp. 53–75. DOI: 10.1007/BF02935746.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. Third. The MIT Press, 2009.

[17] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. "The Complexity of Multiterminal Cuts." In: *SIAM Journal on Computing* 23.4 (1994), pp. 864–894. DOI: 10.1137/S0097539792225297.

[18] Deutsche Energie-Agentur GmbH (dena). *dena-Netzstudie II (Endbericht) – Integration erneuerbarer Energien in die deutsche Stromversorgung im Zeitraum 2015–2020 mit Ausblick auf 2025*. Nov. 2010.

[19] Reinhard Diestel. *Graph Theory*. Fourth. Springer, 2010.

[20] E. A. Dinic. "Algorithm for solution of a problem of maximum flow in networks with power estimation." In: *Soviet Math. Doklady* 11.5 (1970). English translation by Rinehart, R. F., pp. 1277–1280.

[21] Efim A. Dinitz, Alexander V. Karzanov, and Michael V. Lomonosov. "On the structure of the system of minimum edge cuts of a graph." In: *Studies in Discrete Optimization*. Ed. by A. A. Fridman. In Russian. Moscow: Nauka, 1976, pp. 290–306.

[22] Alan Dolan and Joan Aldous. *Networks and Algorithms: An Introductory Approach*. First. Wiley, 1994.

[23] Rod G. Downey and Michael R. Fellows. "Fixed-parameter tractability and completeness II: On completeness for $W[1]$." In: *Theoretical Computer Science* 141.1–2 (Apr. 1995), pp. 109–131.

[24] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer-Verlag London, 2013. DOI: 10.1007/978-1-4471-5559-1.

[25] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems." In: *Journal of the ACM* 19.2 (Apr. 1972), pp. 248–264. DOI: 10.1145/321694.321699.

[26] P. Elias, A. Feinstein, and C. E. Shannon. "A Note on the Maximum Flow Through a Network." In: *IRE Transactions on Information Theory* 2.4 (1957), pp. 117–119.

[27] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. "Vertex Sparsifiers: New Results from Old Techniques." In: *SIAM Journal on Computing* 43.4 (2014), pp. 1239–1262. DOI: 10.1137/130908440.

[28] Amos Fiat and Gerhard J. Woeginger. *Online Algorithms: The State of the Art*. Vol. 1442. Springer, 1998.

[29] Lisa Fleischer and James B. Orlin. "Optimal Rounding of Instantaneous Fractional Flows Over Time." In: *SIAM Journal on Discrete Mathematics* 13.2 (Apr. 2000), pp. 145–153. DOI: 10.1137/S0895480198344138.

[30] Lisa Fleischer and Martin Skutella. "Minimum cost flows over time without intermediate storage." In: *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete Algorithms*. 2003, pp. 66–75.

[31] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[32] L. R. Ford Jr. and D. R. Fulkerson. "Constructing Maximal Dynamic Flows from Static Flows." In: *Operations Research* 6.3 (June 1958), pp. 419–433. DOI: 10.1287/opre.6.3.419.

[33] L. R. Ford Jr. and D. R. Fulkerson. *Flows in Networks*. Santa Monica, California: RAND Corporation, Sept. 1962.

[34] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of* NP*-Completeness*. New York: W. H. Freeman & Co., 1979.

[35] Leslie M. Goldschlager, Ralph A. Shaw, and John Staples. "The maximum flow problem is log space complete for P." In: *Theoretical Computer Science* 21.1 (Nov. 1982), pp. 105–111. DOI: 10.1016/0304-3975(82)90092-5.

[36] R. E. Gomory and T. C. Hu. "Multi-Terminal Network Flows." In: *Journal of the SIAM* 9.4 (1961), pp. 551–570. DOI: 10.1137/0109047.

[37] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. New York, Oxford: Oxford University Press, 1995.

[38] M. Grötschel, L. Lovász, and A. Schrijver. "The Ellipsoid Method and Its Consequences in Combinatorial Optimization." In: *Combinatorica* 1.2 (1981), pp. 169–197. DOI: 10.1007/BF02579273.

[39] Martin Grötschel, Laszlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. 2nd ed. 2. Springer-Verlag Berlin Heidelberg, 1993. DOI: 10.1007/978-3-642-78240-4.

[40] Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. "Characterizations of k-Terminal Flow Networks and Computing Network Flows in Partial k-Trees." In: *Proceedings to the sixth Annual ACM-SIAM Symposium on Discrete Algorithms*. 1995, pp. 641–649.

[41] Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. "Characterizing Multiterminal Flow Networks and Computing Flows in Networks of Small Treewidth." In: *Journal of Computer and System Sciences* 57.3 (Dec. 1998), pp. 366–375. DOI: 10.1006/jcss.1998.1592.

[42] J. Halpern. "A generalized dynamic flows problem." In: *Networks* 9.2 (1979), pp. 133–167. DOI: 10.1002/net.3230090204.

[43] N. Hartmann, L. Eltrop, N. Bauer, J. Salzer, S. Schwarz, and M. Schmidt. *Speicherpotenziale für Deutschland*. Tech. rep. Zentrum für Energieforschung Stuttgart, 2012.

[44] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[45] N. Immerman. "Nondeterministic Space is Closed under Complementation." In: *SIAM Journal on Computing* 17.5 (1988), pp. 935–938. DOI: 10.1137/0217058.

[46] Dieter Jungnickel. *Graphs, Networks and Algorithms*. Heidelberg: Springer, 2013.

[47] David R. Karger. "Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm." In: *Proceedings to the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. 1993.

[48] David R. Karger and Clifford Stein. "A new approach to the minimum cut problem." In: *Journal of the ACM* 43.4 (July 1996), pp. 601–640. DOI: 10.1145/234533.234534.

[49] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. "Competitive Randomized Algorithms for Nonuniform Problems." In: *Algorithmica* 11.6 (July 1994), pp. 542–571. DOI: 10.1007/BF01189993.

[50] Richard M. Karp. "Reducibility among Combinatorial Problems." In: *Complexity of Computer Computations*. Ed. by R. E. Miller and J. W. Thatcher. The IBM Research Symposia Series. Springer US, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[51] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. "An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations." In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (2014), pp. 217–226.

[52] A. Khan and P. Raghavendra. "On mimicking networks representing minimum terminal cuts." In: *Information Processing Letters* 114.7 (July 2014), pp. 365–371. DOI: 10.1016/j.ipl.2014.02.011.

[53] Jon Michael Kleinberg. "Approximation Algorithms for Disjoint Paths Problems." PhD thesis. Massachusetts Institute of Technology, 1996.

[54] B. Kotnyek. *An annotated overview of dynamic network flows*. Tech. rep. INRIA, Sept. 2003.

[55] Robert Krauthgamer and Inbal Rika. "Mimicking Networks and Succinct Representations of Terminal Cuts." In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. 2013, pp. 1789–1799.

[56] Richard E. Ladner. "The circuit value problem is log space complete for P." In: *ACM SIGACT News* 7.1 (Feb. 1975), pp. 18–20. DOI: 10.1145/990518.990519.

[57] F. T. Leighton and A. Moitra. "Extensions and limits to vertex sparsification." In: *Proceedings of the fourty-second ACM Symposium on Theory of Computing*. New York: ACM, 2010, pp. 47–56. DOI: 10.1145/1806689.1806698.

[58] Thomas Lengauer and Klaus W. Wagner. "The binary network flow problem is logspace complete for P." In: *Theoretical Computer Science* 75.3 (Oct. 1990), pp. 357–363. DOI: 10.1016/0304-3975(90)90101-M.

[59] S. E. Lovetskii and I. I. Melamed. "Dynamic flows in networks." In: *Automation and Remote control* 48.11 (1987). Translated from Avtomatika i Telemekhanika, No. 11, pp. 7–29, pp. 1417–1434.

[60] V. M. Malhotra, M. P. Kumar, and S. N. Maheshwari. "An $O(|V|^3)$ algorithm for finding maximum flows in networks." In: *Information Processing Letters* 7.6 (1978), pp. 277–278.

[61] Peter J. Menck, Jobst Heitzig, Norbert Marwan, and Jürgen Kurths. "How basin stability complements the linear-stability paradigm." In: *Nature Physics* 9.2 (Jan. 2013), pp. 89–92. DOI: 10.1038/nphys2516.

[62] Karl Menger. "Zur allgemeinen Kurventheorie." In: *Fundamenta Mathematicae* 10.1 (1927), pp. 96–115.

[63] E. Minieka. "Dynamic Network Flows." In: *Networks* 4.3 (1974), pp. 255–265. DOI: 10.1002/net.3230040305.

[64] A. Moitra. "Approximation Algorithms for Multicommodity-Type Problems with Guarantees Independent of the Graph Size." In: *50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, Oct. 2009, pp. 3–12. DOI: 10.1109/FOCS.2009.28.

[65] A. Moitra. "Vertex Sparsification and Oblivious Reductions." In: *SIAM Journal on Computing* 42.6 (2013), pp. 2400–2423. DOI: 10.1137/100787337.

[66] Ankur Moitra. "Vertex sparsification and universal rounding algorithms." PhD thesis. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2011.

[67] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[68] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. 1982.

[69] Jean-Claude Picard and Maurice Queyranne. "On the structure of all minimum cuts in a network and applications." In: *Mathematical Programming Study* 13 (1980), pp. 8–16. DOI: 10.1007/BFb0120902.

[70] W. B. Powell, P. Jaillet, and A. Odoni. "Stochastic and dynamic networks and routing." In: vol. 8. Elsevier Science, 1995. Chap. 3.

[71] J. Scott Provan and Michael O. Ball. "The complexity of counting cuts and of computing the probability that a graph is connected." In: *SIAM Journal on Computing* 12.4 (1983), pp. 777–788. DOI: 10.1137/0212053.

[72] Marcus Schaefer and Christopher Umans. "Completeness in the Polynomial-Time Hierarchy: A Compendium." In: *SIGACT News* (Oct. 2008).

[73] Wolf-Peter Schill, Jochen Diekmann, and Alexander Zerrahn. "Stromspeicher: Eine wichtige Option für die Energiewende." In: *DIW-Wochenbericht* 82.10 (2015), pp. 195–205.

[74] Alexander Schrijver. "A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time." In: *Journal of Combinatorial Theory, Series B* 80.2 (Nov. 2000), pp. 346–355. DOI: 10.1006/jctb.2000.1989.

[75] Alexander Schrijver. "On the history of the transportation and maximum flow problems." In: *Mathematical Programming* 91.3 (Feb. 2002), pp. 437–445. DOI: 10.1007/s101070100259.

[76] Martin Skutella. "Research Trends in Combinatorial Optimization." In: Springer, 2009. Chap. An Introduction to Network Flows over Time, pp. 451–482. DOI: 10.1007/978-3-540-76796-1_21.

[77] Larry J. Stockmeyer. "The polynomial-time hierarchy." In: *Theoretical Computer Science* 3.1 (Oct. 1976), pp. 1–22. DOI: 10.1016/0304-3975(76)90061-X.

[78] Robert Szelepcsényi. "The Method of Forced Enumeration for Nondeterministic Automata." In: *Acta Informativa* 26.3 (Nov. 1988), pp. 279–284. DOI: 10.1007/BF00299636.

[79] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[80] Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer, 1999.