Approximation Algorithms for Complex Network Flow Over Time Problems

vorgelegt von
Diplom-Informatiker
Martin Groß
geboren in Duisburg

Von der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Tag der wissenschaftlichen Aussprache: 28. Januar 2014

Berlin 2014
D83

Preface

About four years ago, I started working as scientific assistant and PhD student in the group of Martin Skutella at TU Berlin. Now, my thesis has taken shape and I wish to thank the people who have supported me and made this possible.

First of all, I want to thank my advisor Martin Skutella for his support, encouragement and supervision of my thesis. In particular, I want to thank him for introducing me to the field of network flows over time which now have become the main focus of my research. I am very grateful for the independence he allowed me in my work and research, and the opportunity to participate in various international conferences and workshops. In this regard, I also want to thank the DFG and the MATHEON for the funding they provided me for this. I am also very grateful to Gerhard Woeginger for his interest and willingness to take the second assessment of this thesis.

A special thanks goes to all the people I did research with, in particular, the co-authors of my papers, Ashwin Arulselvan, Jan-Philipp Kappmeier, Daniel Schmidt, Melanie Schmidt and Martin Skutella. It is a pleasure to work with you. I am especially grateful to Melanie Schmidt for her support during my PhD time and for reading my thesis very diligently on short notice. Her careful reading and comments were invaluable for me in order to give my thesis the final polish.

I also wish to thank Dorothea Kiefer for helping me with all the administrative aspects of my work, Ralf Hoffman for solving all my computer-related problems and Wiebke Höhn for sharing her experiences of the formalities involved in writing a thesis at TU Berlin.

Furthermore, I want to thank all the members of COGA for creating a great atmosphere for research. Without you, work and research here would not have been half as nice. In particular, I want to Mohsen Rezapour for being my office mate and having the patience to put up with my sometimes seemingly endless CoMa-related work.

Finally, I want to thank my family and friends for supporting me and just being there!

Contents

1 Introduction

Networks are omnipresent in our current world. We live in houses which are connected by water pipelines, power lines and telecommunication cables. Once we step out on the street, we have a network of roads at our disposal, either as a pedestrian, car driver or by using public transportation systems[1], which form large networks of their own in many cities. On a larger scale, we have trains, ships or airplanes available for travelling, and their connections again form networks. Aside from transportation and infrastructure, networks can appear in many other settings. For example, they appear as phylogenetic trees in biology, as molecular graphs in chemistry, as financial networks in economy or as molecular topology in pharmacology. Even in our leisure time, networks have found their way into many popular board games – for example, we can model the boards of popular games like the Settlers of Catan or Kingdom Builder as graphs.

Since many of these networks are part of our everyday lives, we use them on a regular basis and usually, there is some kind of objective involved. If we travel, we usually want to minimize the time necessary for that – perhaps because we want to waste as little time as possible, or to avoid the dreaded "Are we there yet?" questions of children. On the other side, the providers of transportation systems want to attract customers to make profit. Therefore, they attempt to offer a good selection of connections which can be operated with economical pricing. Using telecommunication networks, we want to send or receive data as quickly as possible – having an important email arrive minutes or hours later, or watching a movie stream that lags all the time can be quite annoying. The telecommunication companies on the other hand should try to satisfy their customers as profitable as possible. In financial networks, the goal would be to use assets to generate profit – the list of networks could go on and on.

Many of these problems have in common that people, objects or some kind of abstract commodity need to be transported, and often, many things need to be transported at once. Such transportation problems gave rise to the field of *network flows*. From a mathematical point of view, a network (or *graph*) consists of a set of *nodes* which represent street crossings, computers, houses or other kinds of – potentially abstract – locations, and *arcs* (or *edges*) which model connections between nodes, like roads or cables. Additionally, the arcs have *capacities* that limit the amount they can transport. This model is highly useful for many applications and has been the focus of a lot of research, see for example Ahuja, Magnanti and Orlin [2]. Network flows can be seen as a special case of linear programming and be solved as such, but specialized combinatorial algorithms exist for many network flow problems (see [2]).

However, many of these problems have in common that two of their aspects are not

[1]If we happen to live in a city which actually has a decent public transportation system.

captured by (classic) network flows at all. The first of these aspects is the *variance* of flow over time. For example, traffic is going to be much more intense during the rush hours than in the middle of the night. In telecommunication networks, there can be spikes in traffic due to streaming services, as movies are usually watched in the evening. Similar effects can occur if a new version of a popular software becomes available for download. Planning aircraft, train or passenger transport schedules can also benefit from not having to use exactly the same routes every hour, in order to accommodate demands and legal restrictions like night flight bans. Power management sees daily variation as well as seasonal variation – normally, the demand for energy is higher in winter, due to increased lighting and heating needs. Logistic service providers will usually have highly varying requests, as most people will not get the same letter or parcel every day.

The second aspect is the *delay* between the departure and arrival of flow. For instance, no one would expect cars, busses, trains, air planes or parcels to arrive instantaneously. Whenever some kind of physical distance needs to be overcome, there is a natural speed limit imposed by the speed of light. Normally, this speed seems so high that it is inconsequential, but sending data from a point on earth to its antipode would already take 66.8 ms if a direct optical fiber cable were available between these two spots. 67 ms might seem to be an insignificant amount of time, but for time-sensitive applications like distributed computing, this can already be very noticeable. And this assumes a perfect infrastructure between the two points, which we will generally not have. Also, most applications involve passengers or physical objects which can travel only far more slowly, which makes the delays highly significant. This gets even intensified by the fact that most applications involve deadlines. Arriving at work late, receiving a priority package after a month or waiting on a delayed air plane is not what we want.

Thus, both variance and delays need to be incorporated into the flow model. This was first done by Ford and Fulkerson over half a century ago by introducing the concept of *dynamic flows*. These flows allow for a different flow rate to be specified at any point in time, instead of a single flow value that classic flows would use. Furthermore, they added transit times to the arcs to allow delays to be specified – any flow entering an arc leaves the arc only after the specified transit time. Finally, they added a *time horizon* to the problem, which acts as a deadline for the flow: all flow needs to have arrived at its destination within the time horizon.

Recently, the term *flows over time* has been introduced for dynamic flows in order to prevent misunderstandings, as dynamic graph algorithms in theoretical computer science (e. g., Eppstein, Galil and Italiano [28]) refer to problems where the input changes after solving a problem, and the solution needs to be "repaired" to reflect the change in the input.

Sometimes, capacities, transit times and a time horizon are not sufficient. Consider for instance the examples above, where we want to accomplish something as cheaply or as profitably as possible. This requires the adding of *costs* for arcs to our model. The cost of an arc specifies a fee that needs to be paid for the usage of the arc. Usually, costs can be negative to model profits. Another aspect that requires an additional attribute is the scenario if flow increases or decreases during the traversal of an arc. For example, in a financial flow, we can generate interest on our flow – thus, we might get more money out of an arc than we send into it. Or, we might want to model currency exchanges. This can be modelled by *gains*, which depending on their value can also model losses that could occur through evaporation or theft. Finally, there is the aspect of modelling the origins and destinations of the flow, as well as the amount of flow that we can (or need to) send. Classically, there would be a single origin, the *source*, and a single destination, the *sink*, with a *supply* and a *demand* specifying the amount of flow that can or needs to be sent. The supply could for example be an amount of money that we can invest, or an amount of a good that we have to send. The demand could represent a bill that needs to be paid, or an order of a good that needs to be delivered.

This can be generalized to a set of sources and sinks, where flow from any source can be sent to any sink. This is a realistic assumption, if all our flow is of a single type – like money, for example. However, in applications like logistics where we have many different parcels to send, we require multiple types of flow. Such types of flow are usually referred to as *commodities*. Thus, if we have different types of flow we say that our flow has *multiple commodities*.

A peculiarity of flows over time with multiple commodities is that letting flow wait can allow us to send flow faster. This might seem counterintuitive at first glance, but if we consider a road trip for example, we would probably not be surprised if a route which involves some waiting might still be faster than its alternatives.

Depending on our setting, flows over time can have different objectives. The classic objective is to send as much flow as possible from a given source to a given sink within a time horizon – this is the *maximum flow over time problem*. Related to this is the setting where specific amounts of supplies and demands are given, and we need to find the minimal time needed to fulfill them. This is referred to as the *quickest flow* or *quickest transshipment problem*. We already mentioned the task of sending a given amount of flow as cheaply as possible within a given time – this is called the *minimum cost flow over time problem*. Sending as much flow as possible in a network with gains is referred to as the *generalized maximum flow over time problem*.

Another application of flows over time is their usage for evacuation planning. In this setting, we need to send evacuees to safety as quickly as possible. There are many scenarios which can require an evacuation, ranging from a burning building, or a natural disaster like a hurricane or a wildfire, to a terrorist attack. In these settings, we usually do not know exactly beforehand how much time we have at our disposal. Thus, an evacuation plan that is optimal independently of the time available would be perfect. This objective can be captured by *earliest arrival flows*, which are exactly what we want – they send a maximal amount of flow at any point in time, making them well suited for evacuation problems. They were introduced by Gale [41] under the name of *universal maximal flows* in 1959. However, while earliest arrival flows do always exist in networks with a single sink, networks with multiple sinks do not permit earliest arrival flows in general. Then, we have to settle for *time-* or *value-approximative earliest arrival flows*. The concept of time-approximative earliest arrival flows was studied by Fleischer and Skutella [30] and Baumann and Köhler [10], whereas value-approximative earliest arrival flows were first studied by Hoppe and Tardos [66] and by Groß, Kappmeier, Schmidt and Schmidt [50]. These approximative flows still make a guarantee about the flow value sent at any point in time, but a weaker one. An α-time-approximative earliest arrival flow sends at any point in time θ at least as much as could have been sent until time θ/α, whereas a β-value-approximative flow sends at any point in time θ at least $1/\beta$ of what could have been sent until time θ.

An aspect of flows that we have neglected so far is the direction of flows and the underlying network. For example, roads can theoretically be used in any direction – it is just that traffic becomes much safer for humans if there is a concept of lanes and directions imposed on them. In some problems, like evacuations on a larger scale, we are able to change these directions in order to accelerate the evacuation. This type of problem is usually referred to as the *contraflow problem* or the *lane reversal problem*.

The concept of flows over time has found widespread acceptance in applications, and we will mention several of them in later chapters. Examples can also be found in the surveys of Aronson [4] and Powell, Jaillet and Odoni [92]. Alternatively, searching Google Scholar or similar search engines produces thousands of results[2]. Of course, having applications and a model is only the first step, as we still need to find optimal solutions in the model.

Computing flows over time that are optimal for some objective harbors the problem that there are many degrees of freedom – at every point in time, we can choose a different flow rate for an arc. On the one hand, we need this freedom in order to capture the desired variance of flow over time. On the other hand, we need to bound the freedom somehow so that we can compute optimal solutions efficiently. Aside from introducing flows over time, Ford and Fulkerson also developed two approaches for dealing with this problem. The first approach is to search for *temporally repeated* structures – if there are optimal solutions which show a repetitive behaviour that we can exploit, this reduces our freedom and helps us to find an optimal solution efficiently. However, this is only possible in the special case of problems where we do not use the full amount of variance over time that we are allowed to. The second approach is to perform a *time expansion*, which consists of copying the underlying network for every point in time and linking the copies according to the transit times of the arcs. Thus, if we had two nodes A and B with an arc between them with a transit time of two hours, we would for instance have nodes $(A, 1\,\text{pm})$ and $(B, 3\,\text{pm})$ which would be connected in the time expansion. This gets rid of the temporal dimension altogether, but at the expense of a huge, non-polynomial blow-up in network size, as the number of copies of the network is determined by the time horizon. Since the time horizon is given by a number in the input, this leads to a pseudo-polynomial running time. However, this technique can be applied to most flow over time problems, making it very powerful even though it is not efficient.

Thus, after Ford and Fulkerson's seminal work on flows over time, there were algorithms available to solve them, but the algorithms for anything other than the maximum flow over time problem were unsatisfactory in the sense that they were not efficient and at the same time, there was no indication that efficient algorithms could not be found. In fact, about three decades passed, until progress was made in form of an NP-hardness

[2]When searching for "flow over time" or "dynamic flow" together with "graph" or "network", which seems to produce results in the area of research we are interested in.

result for minimum cost flows over time by Klinz and Woeginger [70, 71], which triggered several further results in this area. If a problem is known to be NP-hard, finding an exact efficient algorithm is exceedingly unlikely, as it would imply P = NP which is almost unanimously believed to be not the case.

As a consequence, efficient *approximation algorithms* replaced exact algorithms as a research goal for those problems for which NP-hardness was established. In terms of flows over time, an approximation algorithm has an approximation guarantee c that can refer to the flow value, the cost or the time horizon, depending on the type of approximation. If it refers to the flow value, then the approximation algorithm produces a solution that is at least as good as an optimal exact solution, with the caveat that it only needs to send $1/c$ of the flow. We call this a *flow approximation.* Similarly, a *cost approximation* yields a solution that has a cost of at most c times the cost of an optimal solution but still sends everything in time, and a *time approximation* results in a solution at least as good as the optimal solution in terms of flow sent or cost, but with c times the time horizon. In some cases, an approximation algorithm can approximate two or more of these parameters at once – in this case, we speak of a *multicriteria* approximation.

Preferably, we would have approximation algorithms whose approximation guarantees are as good as possible. Ideally, this would mean that we find an algorithm A_ε with an approximation guarantee of $1 + \varepsilon$ for any $\varepsilon > 0$. A family of algorithms $(A_\varepsilon)_{\varepsilon>0}$ such that the running time of any A_ε is polynomial in the input size of the problem is called a *polynomial-time approximation scheme* (PTAS). Notice that the running time can be exponential in ε^{-1} for a PTAS. If all of the algorithms A_ε have a running time that is polynomial in the input size *and* ε^{-1}, then $(A_\varepsilon)_{\varepsilon>0}$ is called a *fully polynomial-time approximation scheme* (FPTAS).

An important result for developing approximation algorithms for flows over time was the work by Fleischer and Skutella [30, 31, 32], who introduced an approximation technique based on time expansion which is called *condensed time-expanded networks.* We will also refer to this as *time condensation* for short. This technique is based on rounding transit times and scaling them down, which reduces the size of the time expansion. If the rounding is done carefully, this yields a time-approximation FPTAS for several flow over time problems. However, this technique has the drawback that its approximation guarantee fails for flow over time problems where optimal solutions might need to send flow in cycles. This might seem counter-productive at first, but can actually be necessary for problems involving costs, gains or multiple commodities which are not allowed to wait.

For problems where we might need to send flow in cycles, we can use *sequence condensation* (Groß and Skutella [52]). This technique uses linear programs which are based on variables for the flow values of sequences of arcs, instead of individual arcs. We then round the transit times of these sequences instead of the transit times of single arcs, which produces a much better approximation guarantee in the case of flow running in cycles. However, the sequence-based linear program has exponentially many variables, and solving it requires the separation of its dual problem. Luckily, we can give an FPTAS for the separation problem, which is good enough for our purposes. This generalizes time condensation to quite a few more problems.

Generally, flow over time algorithms can be classified along the approaches that Ford and Fulkerson used: those that use time expansion and those that work on the normal network and instead rely on temporal repetition or similar structures. In this thesis, we will follow this classification and split our new algorithmic results in the Chapters 4, 5 and 6, which are based on time expansion (Chapters 4 and 5) and temporal repetition (Chapter 6), respectively. Our new results in the area of computational complexity can be found in Chapter 7. Finally, we discuss the setting where we are allowed to orient a graph for a flow over time problem in Chapter 8. A detailed overview of the structure of this thesis and its new contributions can be found in the following.

Chapter 2: Preliminaries. In this chapter, we introduce our definitions and notations. We begin with definitions of undirected and directed graphs, as well as important special graphs like paths and cycles, followed by the attributes of their nodes, edges and arcs. After that, we define classical static flows and important concepts for them, like decomposition results and residual networks. We then give definitions of flows over time and objectives for them as well as a discussion of the time models that we can use for flows over time. Finally, we define the time- and value-approximation concepts for flow over time objectives.

Chapter 3: Flow over Time Problems. In the third chapter, we list the flow over time problems that we consider in this thesis and give a survey of the state of the art for these problems, as well as a more detailed overview of our contributions to these problems. Additionally, we discuss the peculiarity of flows over time with multiple commodities that letting flow wait in nodes can allow us to send flow faster. It has been an open question how much faster we can get by allowing waiting. We show that there are instances where we can be a factor of 2 faster, a result due to Groß and Skutella. This improves upon a decade old result of Fleischer and Skutella [30], who showed that this factor must be between $\frac{4}{3}$ and 2, and together, these results give a tight bound of 2 on the benefit of waiting.

Chapter 4: Time Expansion. In this chapter, we give a detailed introduction into time expansion. The concept of time expansion was developed by Ford and Fulkerson [39], and we describe time expansion in the context of gains and multiple commodities. This allows us to use time expansion for generalized maximum multi-commodity flow over time problems.

Furthermore, we study the problem of finding approximative earliest arrival flows in networks where earliest arrival flows do not exist – this is the case in most networks with multiple sinks and is therefore a rather common scenario. Groß, Kappmeier, Schmidt and Schmidt [50] analyzed how good the approximation guarantees for time- and value-approximations can be in arbitrary networks and gave tight bounds for almost all cases. Here, we will describe an algorithm based on iterative time expansion, which computes 2-value-approximative earliest arrival flows, which is best possible for arbitrary networks. This is also the first algorithm of its kind and has been published in [50] as well.

Chapter 5: Time Condensation. In this chapter, we introduce the concept of time condensation, as well as of our contribution of sequence condensation [52]. As for time expansion, we consider a more general approach for time condensation than used by Fleischer and Skutella [32], as we describe it with gains and multiple commodities, allowing us to use it for generalized maximum multi-commodity flow over time problems. This has first been published in Groß and Skutella [51]. Sequence condensation allows us to give an FPTAS for the minimum cost multi-commodity flow over time problem without storage, as well as for the multi-commodity flow over time problems without storage. The best previously known results were by Fleischer and Skutella [32], who gave $2 + \varepsilon$ approximation algorithms for both of these problems and an FPTAS for the minimum cost flow over time problem with storage and conservative cost functions – i. e., cost functions without negative cycles. All these approximations are time-approximations, and the approximations for problems without waiting approximate both time and value. Generalized flows over time have not been considered before [51] to our knowledge, so no previous results were known.

Also, we continue our discussion of finding approximative earliest arrival flows in networks where earliest arrival flows do not exist. Here, we discuss how to use a special form of time condensation called *geometric time condensation* to construct an FPTAS for finding the best possible approximation guarantee for a given instance, which can be much better than what is possible in the general case. These results were also published in Groß, Kappmeier, Schmidt and Schmidt [50].

Chapter 6: Temporal Repetition. In this chapter, we study techniques which do not require time expansion, but work on the normal network instead.

In particular, we study the special case of generalized maximum flows over time, where we have only losses and no gains (which we also refer to as *lossy* gains) and the losses are proportional to the transit times. That means that on any arc with a transit time of τ, we lose the same fraction γ of flow. We describe a successive shortest path variant for this case, which works on the normal network, but requires a pseudo-polynomial number of iterations in the worst case. Building upon this algorithm, we devise an FPTAS for this problem, which is a value-approximation instead of a time-approximation. This is a rare feature for flow over time approximations, because usually, it is the temporal dimension which introduces the complexity which has to be dealt with. This FPTAS has the second special property that the error ε appears only logarithmically as $\log \varepsilon^{-1}$ in the runtime of the FPTAS – this makes it extremely cheap to obtain very accurate approximations. This was published in Groß and Skutella [51].

Chapter 7: Complexity. In this chapter, we will study techniques to prove the NP-hardness of flow over time problems.

We will present an NP-hardness result for generalized maximum flows over time with arbitrary or lossy gains that builds on the reduction of Klinz and Woeginger [71]. In fact, this reduction even yields a non-approximability result in terms of flow value for them. This result has been published in Groß and Skutella [51] and was the first hardness result for this particular problem.

Finally, we study the special case of generalized flows over time with lossy gains, that are also proportional to transit times. Here, we show a connection to earliest arrival flows – a maximum generalized flow over time without intermediate storage fulfills the earliest arrival property. This becomes significant in conjunction with recent results by Disser and Skutella [25], who showed that earliest arrival flows are NP-hard. The mentioned connection to earliest arrival flows was first published in Groß and Skutella [51].

Chapter 8: Orientations and Flows over Time. In the final chapter, we investigate the problem of orienting an undirected graph such that the orientation is favorable for a subsequent flow over time computation.

We consider the setting where we can choose the direction of arcs in a preprocessing step, and are not allowed to change it later on. This is reasonable in the sense that we might not have the resources necessary to change the direction of lanes during an evacuation. Of course, there will be settings where we do have the resources to do so, but we will have to leave that question to further research. Also, we assume that we have to orient each arc exclusively into one direction, splits are not possible. But, if we want to orient the lanes of a road individually, we can still do so by modelling the road by using an arc for each of its lanes.

We will focus predominately on the question of how much the orientation of the arcs in a preprocessing step costs us. In order to define this cost, we compare flows over time in an oriented network with a flow over time in an undirected graph. This flow over time in the undirected network can be seen as something we could do, if all cars were controlled by a single artificial intelligence (AI) – then we would not need lanes or directions, because the AI could react and plan fast enough so that this would become unnecessary. Cars controlled by humans need directed lanes, however, which is bound to use the roads less efficiently than the AI could accomplish. Here, we are interested in how much worse than the AI we can get, if we use our model of orientation. More formally, we introduce a *flow price of orientation* which is based on the quotient of flow that the AI can send divided by what we can send in the most favorable orientation, for a given time horizon. Analogously, the *time price of orientation* is defined as the ratio of the amount of time we need to send a given amount of flow in a favorable orientation compared to what the AI can do.

It follows from Ford and Fulkerson's work [39] that both prices of orientation are 1 if we have a single source and sink. For the case of multiple sources and / or sinks, we are able to show tight bounds for the flow price of orientation, which turns out to be 2 if we have either multiple sources or multiple sinks, and 3 if we have both. We give an algorithm for finding orientations with a flow price of orientation of at most 3 (2 for a single source or sink). The algorithm reduces the problem to the single source and sink case, by simulating supplies and demands through capacities of auxiliary arcs. This simulation is not precise, however, and we show how to obtain good capacities to obtain the 2 and 3 bounds by using an iterative approach that uses Brouwer fixed-points [17]. For the time price of orientation, we give lower bounds that are linear in the network size. Furthermore, we show that not even approximative earliest arrival flows exist in this setting, if we want to maximize over all orientations. We conclude this chapter by considering orientations in conjunction with multi-commodity flow over time problems and show that computing the price of orientation is NP-hard and sometimes even non-approximable in this case. All of this is a – as of yet unpublished – work by Arulselvan, Groß and Skutella [7].

Concluding remarks. This thesis is supposed to be mostly self-contained, however, we assume our reader to be familiar with the basic concepts of approximation algorithms, combinatorial optimization, complexity theory, linear programming and classical network flow theory.

For an introduction to these topics, see for example the books by Grötschel, Lovász and Schrijver [53], Korte and Vygen [75], Nemhauser and Wolsey [81] and Schrijver [102]. For network flow theory in particular, the book by Ahuja, Magnanti and Orlin [2] provides a comprehensive overview. A thorough introduction into complexity theory can also be found in the books by Arora and Barak [6] and Garey and Johnson [43]. An introduction into approximation algorithms in combinatorial optimization can for example be found in the books by Hochbaum [64], Vazirani [114], or Williamson and Shmoys [119]. Alternative introductions to network flows over time can for example be found in Hoppe [65] or Skutella [104].

As a final remark, notice that we will only discuss *fractional flow over time problems* in this thesis, i. e., problems where we do not force the flow rates to be integral. This is due to the fact that just adding the temporal dimension adds many challenges on its own, which we want to focus on and having integrality constraints would dilute that to a certain extent. Of course, this does not mean that integral flows over time would not be interesting, quite on the contrary. However, since even fractional flows over time are not completely understood, it seems not unreasonable to focus on understanding them first.

2 Preliminaries

This chapter focuses on providing a formal introduction into the concepts that are required for the later chapters. The organisation of this chapter is as follows:

- Chapter 2.1 defines directed and undirected graphs and our notations for them. In addition, we highlight important types of subgraphs like cycles and paths and specialized notations for them. Finally, we introduce the attributes (e. g., capacities, costs, gains, etc.) for arcs, edges and nodes that we will study in this thesis.
- In Chapter 2.2 we define classical static network flows and introduce important techniques – namely flow decomposition and residual networks – that will be used in the later chapters.
- In Chapter 2.3 we define flows over time (sometimes also called dynamic flows), which are static flows with an added temporal dimension. These flows will be the main focus of this thesis and will be discussed in detail in the later chapters.
- Finally, in Chapter 2.4 we discuss what kind of approximations are possible for flow over time problems.

2.1 Graphs

In this chapter, we define our basic notations for graphs. We begin with directed graphs, as classical flows are usually defined for them. This is reasonable, as flows are often used to solve logistical and related problems – and in these problems, there is usually a concept of direction in the underlying networks. However, these directions are usually not inherent in the network but only enforced by rules applied to the network. For example, we could use street lanes in any direction – it would just be prone to cause many accidents if everyone would choose a direction by themselves. But even if using streets without any rules is a bad idea, there are settings where influencing the rules is within our possibilities. For instance, we might be able to change the direction of street lanes in a planning stage. For such problems, we define undirected graphs and orientations of them.

2.1.1 Definitions

In the literature, graphs are sometimes referred to as *networks* and vice versa. The distinction between these two terms is not always clear. Usually, a network refers to a directed graph, a directed graph with capacities or a directed graph with multiple attributes like capacities, costs and transit times. In contrast to this, a graph can be either undirected or directed, and may not have any attributes. We will use the terms graph and network interchangeably and we will take care to make always clear whether our graphs are undirected or not, and which attributes are used.

Directed graphs. A *directed graph* G consists of a finite set of *nodes* $V(G)$ and a finite set of *arcs* $A(G)$, with an arc $a \in A(G)$ being a pair of nodes (v, w), $v, w \in V(G)$. We set $n := |V(G)|$ and $m := |A(G)|$. For an arc $a = (v, w) \in A(G), v, w \in V(G)$, we refer to v as its *start node* and w as its *end node*. Arcs $a, a' \in A(G)$, $a \neq a'$ are called *parallel*, if they have the same start and end node, i. e., $a = (v, w)$ and $a' = (v, w)$ for some $v, w \in V(G)$. A *loop* $a \in A(G)$ is an arc with the same start and end node, i. e., $a = (v, v)$ for some $v \in V(G)$. We assume that our graphs have neither parallel arcs nor loops – this no significant restriction, as we can split parallel arcs or loops by introducing additional nodes. A sketch of these constructions can be seen in Figure 2.1. Due to this assumption, we can identify arcs by their start and end node.

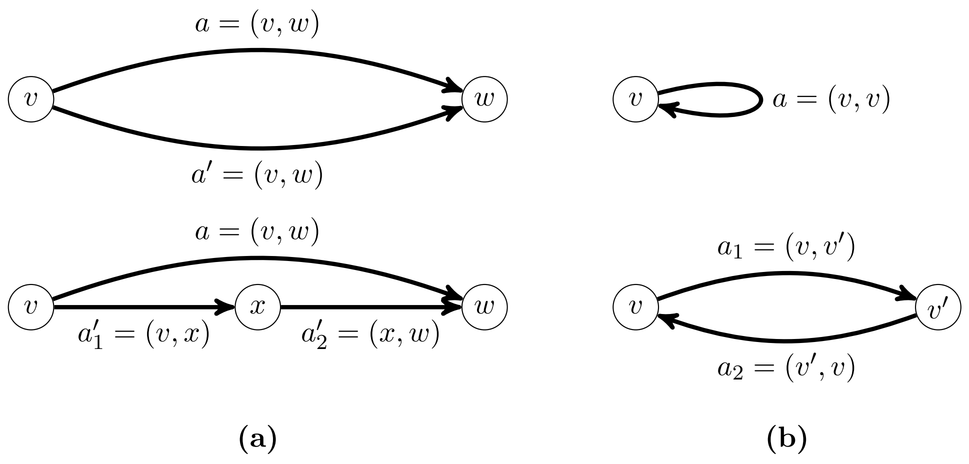


Figure 2.1: (a) Parallel arcs and a construction to break them up. (b) A loop and a construction to break it up.

We define $\delta^+_G(v)$ as the set of arcs of G leaving a node $v \in V(G)$, which we refer to as *outgoing arcs*. Similarly, we define $\delta^-_G(v)$ as the set of arcs of G entering a node $v \in V(G)$, which we also refer to as the set of *incoming arcs*, and $\delta_G(v)$ as the union of the two:

$$\begin{aligned}
\delta^+_G(v) &:= \{a \in A(G) \mid a = (v, w) \text{ for some } w \in V(G)\}\,,\\
\delta^-_G(v) &:= \{a \in A(G) \mid a = (u, v) \text{ for some } u \in V(G)\}\,,\\
\delta_G(v) &:= \delta^+_G(v) \cup \delta^-_G(v).
\end{aligned}$$

We call the arcs $a \in \delta_G(v)$ *incident* to v, and we refer to nodes $v, w \in V(G)$ for which an arc $a = (v, w) \in A(G)$ exists as *adjacent*. If the graph G can be inferred from the context, we just write $\delta^+(v)$, $\delta^-(v)$ and $\delta(v)$.

Undirected graphs and orientations. An *undirected graph* G consists of a finite set of nodes $V(G)$ and a finite set of *edges* $E(G)$. An edge $e \in E(G)$ is a set $\{v, w\}$ containing two nodes $v, w \in V(G)$. Similarly to directed graphs, we call edges $e, e' \in E(G)$, $e \neq e'$ parallel, if $e = \{v, w\}$ and $e' = \{v, w\}$ for some $v, w \in V(G)$, and call edges $\{v, w\}$ with $v = w$ loops. Analogously to the directed case, we assume that our graphs do not contain parallel edges, allowing us to identify edges by their sets of nodes. For undirected graphs, we define $m := |E(G)|$.

We define $\delta_G(v)$ as the set of edges of G that contain a node $v \in V(G)$:

$$\delta_G(v) := \{e \in E(G) \mid e = \{v, w\} \text{ for some } w \in V(G)\}.$$

Again, edges $e \in \delta_G(v)$ are called incident to v, and nodes $v, w \in V(G)$ are called adjacent, if an edge $e = \{v, w\} \in E(G)$ exists. We write $\delta(v)$ instead of $\delta_G(v)$, if G can be inferred from the context.

An *orientation* of an undirected graph G is a directed graph $\overrightarrow{G}$ with $V(\overrightarrow{G}) := V(G)$ and $A(\overrightarrow{G})$ satisfying

$$e = \{v, w\} \in E(G) \Leftrightarrow \text{either } (v, w) \in A(\overrightarrow{G}) \text{ or } (w, v) \in A(\overrightarrow{G}).$$

2.1.2 Subgraphs: Sequences, paths, cycles and more

For the purpose of analyzing graphs and flows, it can be helpful to split a graph into simpler, easier to understand subgraphs. We define a *subgraph* of a directed graph G to be a directed graph G' that consists of a subset of nodes and arcs of the original graph: $V(G') \subseteq V(G)$ and $A(G') \subseteq A(G)$. Subgraphs of undirected graphs are defined analogously. We are now interested in simple subgraphs which consist of a single, connected sequence of arcs.

Sequences, paths and cycles. Let G be a directed graph. A *v-w-sequence* S in G is a sequence of arcs $(a_1, a_2, \ldots, a_{|S|})$, $a_i = (v_i, v_{i+1}) \in A(G)$ with $v = v_1$, $w = v_{|S|+1}$. A *v-w-path* is a v-w-sequence with $v_i \neq v_j$ for all $i \neq j$. A *v-cycle* is a v-v-sequence with $v_i = v_j$ implying $i = j$ or $i, j \in \{1, |S| + 1\}$. Thus, no arc or node occurs twice or more in paths and cycles (with the exception of $v_1 = v_{|C|+1}$ in cycles). Our paths and cycles are therefore *simple* arc sequences. A v-w sequence naturally induces a *subgraph* G' of G by $V(G') := \left\{v_1, \ldots, v_{|S|+1}\right\}$ and $A(G') := \{a_i \mid i = 1, \ldots, |S|\}$. We will sometimes identify a sequence with its induced subgraph.

We will treat sequences as sets, if the order of arcs is not important. Thus, we write $a \in S$ if an arc a is contained in a sequence S and likewise $v \in S$ if a node v is incident to one of the arcs contained in S. We refer to the number of occurrences of an arc a in a sequence S as $\#(a, S)$. We denote the subsequence of the arcs from the beginning of the sequence up to (but not including) the k-th occurrence of an arc $a \in A(G)$ by $S[\rightarrow (a, k)]$. Thus, for

$$S = (a_1, \ldots, \underbrace{a, \ldots,}_{k-1} a, \ldots, a_{|S|}),$$

we get

$$S[\to (a,k)] = (a_1, \dots, \underbrace{a, \dots}_{k-1}).$$

Here, the $a, \dots$ stands for arc a followed by zero or more arcs $a' \neq a$. Similarly, we define $S[\to i]$ for $i = 1, \dots, |S|$ as the subsequence of the arcs from the beginning of the sequence up to (but not including) the i-th element in the sequence: $S[\to i] := (a_1, \dots, a_{i-1})$. Furthermore, we extend the concept of incidence from arcs to sequences and write $S \in \delta^+_G(v)$ if $a_1 \in \delta^+_G(v)$ and $S \in \delta^-_G(v)$ if $a_{|S|} \in \delta^-_G(v)$. Figure 2.2 illustrates some of these notations. For undirected graphs G, we define a *v-w-sequence* S as a

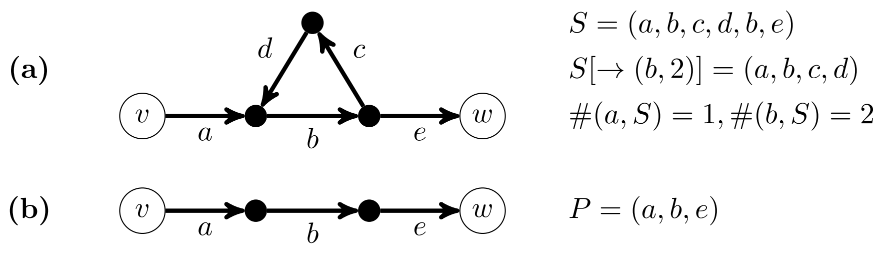


Figure 2.2: (a) A v-w-sequence $S = (a, b, c, d, b, e)$. (b) A v-w-path P.

sequence of edges $(e_1, e_2, \dots, e_{|S|})$, $e_i = \{v_i, v_{i+1}\} \in E(G)$ with $v = v_1$, $w = v_{|S|+1}$ and use the same notations as for sequences of arcs.

Sequence decomposition. Notice that a v-w-sequence S of arcs or edges can always be decomposed into a v-w-path and (zero or more) cycles. We formalize this in the following lemma.

Lemma 2.1. *A v-w-sequence S of arcs or edges can always be decomposed into a (possibly empty) v-w-path P and zero or more cycles $C_1, \dots, C_k$. This can be done in $O(|S|)$.*

Proof. In the following, we show the result for sequences in directed graphs. The proof for undirected graphs works analogously. Let $S = (a_1, \dots, a_{|S|})$, $a_i = (v_i, v_{i+1})$ with $v = v_1$, $w = v_{|S|+1}$ be our v-w-sequence that is to be decomposed. If S is simple, we are done by setting $P := S$, $k := 0$. Otherwise, there is a node x that occurs at least twice in S. Let v_i, v_j, $i < j$ be the first two occurrences of $x \in S$, and choose x so that j is minimal. Then $C := (a_i = (v_i, v_{i+1}), \dots, a_{j-1} = (v_{j-1}, v_j))$ is a x-cycle, as by choice of x, there cannot be two or more occurrences of a node other than x in C.

Now consider the sequence $S' = (a_1, \dots, a_{i-1}, a_j, \dots, a_{|S|})$ that is obtained from S by removing $(a_i, \dots, a_{j-1})$. Due to $x = v_i = v_j$, S' is a v-w-sequence with fewer arcs than S. Thus, we can apply the procedure outlined above again. This yields a decomposition in a path and cycles, and can be done in $O(|S|)$, as only a single iteration of the sequence is necessary to do this – we can essentially traverse the sequence and split of cycles while traversing. □

We refer to this decomposition of a sequence S as its *path-cycle-decomposition*. Notice that the algorithm is deterministic, so the path-cycle-decomposition is well defined.

Cycle-paths, path-cycles and bi-cycles. Aside from paths and cycles, we will need three structures that consist of paths and cycles. These are important to describe elementary generalized flows, which we will study later. A *path-cycle* in G consists of a path $P = (a_1, \dots, a_{|P|} = (v_{|P|}, v_{|P|+1}))$ and a $v_{|P|+1}$-cycle C, such that P and C are arc-disjoint, i. e., $A(P) \cap A(C) = \emptyset$. A *cycle-path* in G consists of a v_1-cycle C and a path $P = (a_1 = (v_1, v_2), \dots, a_{|P|})$ such that P and C are arc-disjoint. A *bi-cycle* in G consists of a v-cycle C_1, a w-cycle C_2 and a v-w-path P such that C_1, C_2 and P are pairwise arc-disjoint. Like sequences, these structures induce subgraphs of G and we will identify these structures with their subgraphs. Figure 2.3 illustrates these structures.

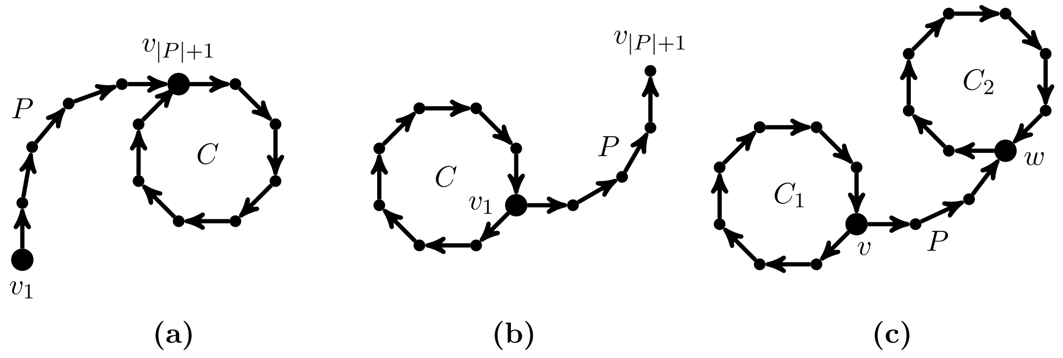


Figure 2.3: (a) A path-cycle, (b) a cycle-path and (c) a bi-cycle.

2.1.3 Attributes of arcs, edges and nodes

In this section, we introduce attributes for arcs, edges and nodes that are necessary to model various real-world phenomena.

We begin by defining attributes for directed graphs and arcs and discuss their meaning. For undirected graphs, we are mainly interested in how attributes of edges can be carried over to arcs in orientations, as we will work mostly on naturally directed graphs or orientations of undirected graphs.

Capacities. *Capacities* are given by a function $u : A(G) \to \mathbb{R}_0^+ \cup \{\infty\}$ and are used to limit the flow that can be sent into an arc. The way capacities do this depends on the setting. For static flows, capacities limit the *total amount* of flow on an arc – this is akin to limiting the total number of cars on a street, for example. For flows over time, capacities limit the *rate* at which flow can enter an arc at any point in time, i. e., like the number of lanes on a street restricts the number of cars that can enter it at the same time.

Transit times. *Transit times* are given by a function $\tau : A(G) \to \mathbb{R}_0^+ \cup \{\infty\}$ and specify the amount of time a unit of flow needs to traverse an arc, i. e., if flow enters an arc $a = (v, w)$ at time θ, it arrives at w at time $\theta + \tau(a)$. We will mostly focus on constant transit times, i. e., transit times that are not dependent on time, or the amount or rate of flow on the arc. Related to transit times, there is often a constant $T \in \mathbb{R}_0^+ \cup \infty$ called the *time horizon* specified in flow over time problems. This acts as a deadline, after which no flow may be left in the network.

Costs. *Costs* are given by a function $c : A(G) \to \mathbb{R} \cup \{-\infty, \infty\}$ and denote a price that is to be paid for sending flow through an arc. We use linear costs, i. e., if x_a units of flow are sent though arc a, the cost for sending them is $c(a)x_a$. This can be used to model shipping costs and similar fees.

Gains. *Gains* are given by a function $\gamma : A(G) \to \mathbb{R}_0^+ \cup \{\infty\}$ and can be used to model that the amount of flow changes while traversing through an arc. This can model taxes, interest, theft, evaporation and much more. If x_a units of flow enter an arc a, there will be $\gamma(a)x_a$ flow units leaving the arc. Therefore, gains < 1 are also called *losses* and a graph G with $\gamma(a) \leq 1$ for all $a \in A(G)$ is called *lossy*.

Supplies and demands. For nodes, we have *supplies and demands* of multiple *commodities* as attributes. As the name suggests, these denote some (potentially abstract) goods that need to be shipped through a network. Each commodity has nodes with its supplies from where the goods need to be sent to the nodes with its demands. Notice that using supplies of one commodity to satisfy demands of another commodity is not possible. Formally, we assume that we have k commodities $1, \ldots, k$ from a set $K := \{1, \ldots, k\}$. For each commodity $i \in K$, we have supplies and demands (sometimes also referred to as *balances*) given by a function $b^i : V(G) \to \mathbb{R} \cup \{-\infty, \infty\}$. Notice that we allow supplies and demands to be infinite – this is because we want to model problems like the classical maximum s-t-flow problem with supplies and demands as well. This will allow us to use the same notations for maximum flow and transshipment problems (these problems will be described in detail in Chapter 3). We accumulate all these supplies and demands under a single function $b : V(G) \times K, b(v, i) := b^i(v)$. For brevity, we write b for b^1 in the single commodity case. Furthermore, we set $b_v^i := b^i(v)$ for $i \in K, v \in V(G)$ and $b_v := (b_v^1, \ldots, b_v^k)^T$.

For a commodity $i \in K$, nodes $v \in V(G)$ with $b_v^i > 0$ are called *sources* of commodity i and their b_v^i is referred to as *supply*. Nodes $v \in V(G)$ with $b_v^i < 0$ are called *sinks* of commodity i, and their b_v^i is referred to as *demand*. All other nodes, i. e., nodes $v \in V(G)$ with $b_v^i = 0$ are called *intermediate nodes* of commodity i, in contrast to the sources and sinks, which are summarized under the term *terminals*. We define the set of sources of commodity i by $S_+^i := \left\{ v \in V(G) \mid b_v^i > 0 \right\}$ and the set of sinks of commodity i by $S_-^i := \left\{ v \in V(G) \mid b_v^i < 0 \right\}$. Furthermore, we write $S_+ := \bigcup_{i \in K} S_+^i$ for the set of the sources of all commodities and $S_- := \bigcup_{i \in K} S_-^i$ for the set of the sinks of all commodities. Finally, we define $B_+^i := \sum_{v \in S_+^i} b_v^i$ to be the sum of all supplies of commodity i, and $B_-^i := \sum_{v \in S_-^i} b_v^i$ to be the sum of all demands of commodity i. Likewise, we have $B_+ := \sum_{i \in K} B_+^i$ as the sum of all supplies and $B_- := \sum_{i \in K} B_-^i$ as the sum of all demands.

If all supplies and demands are finite, we assume that $\sum_{v \in V(G)} b_v^i = 0$ holds for all commodities $i \in K$, unless specified otherwise. In this case, $B_+^i + B_-^i = 0$ holds for all $i \in K$, which means that all supplies can be used and all demands can be fulfilled.

Attributes for undirected graphs. As we mentioned before, we are interested in carrying over attributes of edges to their corresponding arcs when orienting graphs. We accomplish this by defining *orientations* of edge attributes as follows. Let u, τ, c, γ be capacities, transit times, costs and gains for an undirected graph G. Then we define their oriented versions $\overrightarrow{u}, \overrightarrow{\tau}, \overrightarrow{c}, \overrightarrow{\gamma}$ for $\overrightarrow{G}$ by

$$\begin{aligned} \overrightarrow{u}_{a=(v,w)} &:= u_{e=\{v,w\}}, & \overrightarrow{\tau}_{a=(v,w)} &:= \tau_{e=\{v,w\}}, \\ \overrightarrow{c}_{a=(v,w)} &:= c_{e=\{v,w\}}, & \overrightarrow{\gamma}_{a=(v,w)} &:= \gamma_{e=\{v,w\}}. \end{aligned}$$

Since nodes are not affected by orientations, we do not have to modify supplies and demands.

Notations for arcs and sequences. For brevity, we define $u_a := u(a)$, $\tau_a := \tau(a)$, $c_a := c(a)$ and $\gamma_a := \gamma(a)$ for an arc $a \in A(G)$. Furthermore, we extend these arc attributes to sequences $S = (a_1, \dots, a_{|S|})$ by defining $\tau_S := \sum_{i=1}^{|S|} \tau_{a_i}$, $c_S := \sum_{i=1}^{|S|} c_{a_i}$ and $\gamma_S := \prod_{i=1}^{|S|} \gamma_{a_i}$. Thus, τ_S is the time needed to travel through S, c_S is the cost of doing so and γ_S is the factor by which the outflow depends on the inflow. Cycles C with $\gamma_C = 1$ are called *unit gain cycles*, cycles with $\gamma_C > 1$ *flow-generating cycles* and cycles with $\gamma_C < 1$ *flow-absorbing cycles*. Cycles C with $c_C < 0$ are called *negative cost cycles*. A cost function c is called *conservative* for a graph G, if G has no cycle C of negative cost.

Short notation for attributes. For brevity, we sometimes write a graph G and its associated attributes as a tuple $(G, u, \tau, \dots)$. The size of the tuple varies depending on the number of attributes associated with the graph. When using the tuple notation, u refers to the capacities, γ to the gains, c to the costs and b to the supplies and demands. We also add a time horizon to the tuple, if present, as well as terminals if they are important. This is usually the case when there is only a single source and sink. In this case, the single source and sink are sometimes called super-terminals. When using super-terminals, s usually refers to the super-source and t to the super-sink.

2.2 Static Flows

In this chapter, we give an introduction into classical static network flows that do not incorporate any concept of time. We begin by defining flows and continue by introducing flow decompositions and residual networks. We will define our flows for the case of multiple commodities and gains, and treat flows without gains or with a single commodity as special cases. We will also define flows for undirected networks by reducing them to flows in directed networks.

2.2.1 Definitions

In order to define flows (and flows over time later on) for directed networks, we will make the assumption that sources do not have incoming arcs and sinks have no outgoing arcs. This makes defining flows easier and can be done without loss of generality, since we can slightly modify graphs to guarantee this property. We discuss this process in the following.

For a given directed graph G, we create a slightly modified directed graph G' whose sources do not have incoming arcs and whose sinks have no outgoing arcs. We define G' by introducing a new node for each source and sink. This new node is then connected to the terminal it was created for – this ensures that the desired property holds for G':

$$\begin{aligned} V(G') &:= V(G) \cup \{\, s_v \mid v \in S_+ \} \cup \{\, t_v \mid v \in S_- \}\,, \\ A(G') &:= A(G) \cup \{\, (s_v, v) \mid v \in S_+ \} \cup \{\, (v, t_v) \mid v \in S_- \}\,. \end{aligned}$$

Let u, c, γ and τ be capacities, costs, gains and supplies and transit times for G. We extend the attributes to the new arcs by choosing "neutral" values for them:

$$u'_a := \begin{cases} u_a & a \in A(G) \\ \infty & \text{else} \end{cases}, \quad c'_a := \begin{cases} c_a & a \in A(G) \\ 0 & \text{else} \end{cases} \quad \text{for all } a \in A(G'),$$

$$\gamma'_a := \begin{cases} \gamma_a & a \in A(G) \\ 1 & \text{else} \end{cases}, \quad \tau'_a := \begin{cases} \tau_a & a \in A(G) \\ 0 & \text{else} \end{cases} \quad \text{for all } a \in A(G').$$

Finally, if G has supplies and demands b, we shift these supplies and demands to the new terminals:

$$b'^{i}_{v'} := \begin{cases} b^i_v & v' = s_v \\ b^i_v & v' = t_v \\ 0 & \text{else} \end{cases} \quad \text{for all } v' \in V(G'), i \in K.$$

Since this creates at most $n = |V(G)|$ new nodes and arcs, we can do this in time $O(n)$. We will now use this assumption to define flows.

Flows in directed graphs. We begin by defining classical static flows for directed graphs. A (static) *generalized multi-commodity flow* x in a directed graph G with capacities u, costs c, gains γ and supplies and demands b of commodities $i \in K = \{1, \dots, k\}$ is a function $x : A(G) \times K \to \mathbb{R}$ that assigns a flow value $x^i_a := x(a, i)$ to every arc for every commodity, subject to the following constraints:

- *Flow conservation constraints* ensure that we neither use more supplies nor fulfill more demands of a commodity than a node has. The amount of used supplies and fulfilled demands depends of the difference between incoming and outgoing flow of a node:

$$\min(0, b^i_v) \le \sum_{a \in \delta^+_G(v)} x^i_a - \sum_{a \in \delta^-_G(v)} \gamma_a x^i_a \le \max(0, b^i_v) \qquad \text{for all } v \in V(G), i \in K.$$

If all supplies and demands are used, i. e.,

$$\sum_{a\in\delta_G^+(v)} x_a^i - \sum_{a\in\delta_G^-(v)} x_a^i = b_v^i \qquad \text{for all } v \in V(G), i \in K,$$

we say that *all supplies and demands have been fulfilled.*

- *Capacity constraints* make certain that the flow values assigned to the commodities for an arc do not violate its capacity:

$$\sum_{i\in K} x_a^i \le u_a \qquad \text{for all } a \in A(G).$$

- *Non-negativity constraints* are used to guarantee non-negativity of the flow:

$$0 \le x_a^i \qquad \text{for all } a \in A(G), i \in K.$$

This is sensible, as flow usually represents some kind of real world commodity, which often cannot take negative values.

There are two important special cases. The first case is when we have only a single commodity, i. e., $k = 1$. In this case, x is a *single commodity flow* and we call x just a *generalized flow*. The second case is if we have *unit gains*, i. e., $\gamma_a = 1$ for all arcs $a \in A(G)$. In this setting, we have "normal" flow conservation and refer to x just as a *multi-commodity flow*.

We refer to the flow values of a commodity $i \in K$ by $x^i : A(G) \to \mathbb{R}, x_a^i := x^i(a) := x(a, i)$. Each $x^i, i \in K$ can be interpreted as a single commodity flow. For notational brevity, we omit all the notations for commodities in the single commodity case and write x_a instead of x_a^1, and so on. Finally, if it is clear from the context that our flows are multi-commodity and / or generalized, we omit this and call the generalized / multi-commodity flows just flows.

The *value* $|x|$ of the flow x is defined as the amount of demands of all commodities that x fulfills:

$$|x| := \sum_{i\in K} |x^i|, \quad |x^i| := \sum_{v\in S_-^i} \sum_{a\in\delta_G^-(v)} \gamma_a x_a.$$

The *concurrent value* $|x|_{\text{conc}}$ of a flow x is defined as the minimal fraction of demands of a commodity that is fulfilled by x:

$$|x|_{\text{conc}} := \min_{i\in K} \frac{|x^i|}{|B_-^i|} \in [0, 1].$$

Thus, the commodity of which the smallest fraction of demands has been fulfilled, determines the concurrent value of a flow. The *cost* of a flow x is defined by

$$c(x) := \sum_{i\in K} c(x^i), \quad c(x^i) := \sum_{a\in A(G)} c_a x_a.$$

Finally, we call flows *circulations* in the special case of $b_v^i = 0$ for all $i \in K$, $v \in V(G)$.

Flows in undirected graphs. We define flows in undirected graphs by identifying them with flows in modified, directed versions of the undirected graphs. We do this as follows. Let G be an undirected graph with capacities u, costs c, gains γ and supplies and demands b. We modify it into a directed graph G' by replacing every edge $e = \{v, w\} \in E(G)$ with the construction depicted in Figure 2.4. More formally, we define G' by

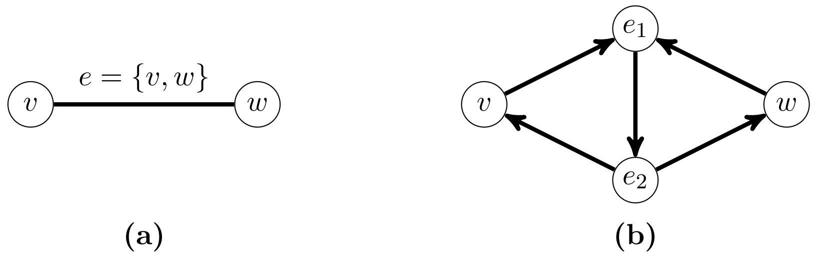


Figure 2.4: (a) An undirected edge and (b) a gadget of directed arcs as its replacement.

$$\begin{aligned} V(G') &:= V(G) \cup \{ e_1, e_2 \mid e \in E(G) \}, \\ A(G') &:= \{ (v, e_1), (w, e_1), (e_1, e_2), (e_2, v), (e_2, w) \mid e \in E(G) \}. \end{aligned}$$

We define capacities u', costs c' and gains γ' for all $a \in A(G')$ by

$$u'_a := \begin{cases} u_e & a = (e_1, e_2) \\ \infty & \text{else} \end{cases}, \quad c'_a := \begin{cases} c_a & a = (e_1, e_2) \\ 0 & \text{else} \end{cases}, \quad \gamma'_a := \begin{cases} \gamma_a & a = (e_1, e_2) \\ 1 & \text{else} \end{cases}.$$

Supplies and demands b' are defined for all $v \in V(G')$ and $i \in K$ by

$$b'^i_v := \begin{cases} b^i_v & v \in V(G) \\ 0 & \text{else} \end{cases}.$$

Thus, (e_1, e_2) inherits the attributes of the original edge and the other arcs and new nodes get neutral attributes. The purpose of this construction is to be able to send flow from v to w and w to v under the constraint that the sum of the flows sent in both directions does not exceed the capacity of the original edge. This is accomplished by the shared (e_1, e_2) arc. In this sense, the construction allows us to orient parts of the capacity of the edge in different directions, as would be possible with the lanes of a road, for example.

For the sake of completeness, we mention that there is another way to define flows in an undirected graph G. Recall that a generalized multi-commodity flow x can be split into single commodity flows x^i for each commodity. In this other way, x is a flow in G if there exist orientations for each commodity, such that x^i is a directed flow in its orientation. Additionally, the capacity constraints $\sum_{i \in K} x^i_e \leq u_e$ must hold for each edge $e \in E(G)$. For more details on this type of definition, see for example Schrijver [102].

The differences between the two definitions are that ours does not require orientations but allows flow to use $(v, e_1), (e_1, e_2), (e_2, v)$-sequences which have no counterpart in the undirected graph. This is unproblematic if running in these cycles has no benefit, which will be the case in our applications. Thus, we prefer the first definition, since it circumvents the need for orientations.

2.2.2 Flow Decompositions and Path-based Flows

We have now defined flows and will now look into their structure. Therefore, we will analyze the structure of flows in the following. In particular, we will focus on decomposing flows in arbitrary graphs into flows on very basic graphs. This will allow us to find new ways to describe flows. The process of decomposing a flow is referred to as *flow decomposition*.

In this section, we will restrict ourselves to single commodity flows. This is due to the fact that a multi-commodity flow with k commodities consists of k single commodity flows, which are independent of each other with the sole exception that they share the capacities of their graph. But since we are just interested in decomposing a given flow, we do not need to concern us with capacities – the flow values on each arc have to remain the same after the decomposition. Thus, we can decompose the k single commodity flows independently.

Historically, decomposition of single commodity flows was first done for flows without gains (or with unit gains, i. e., the case of $\gamma_a = 1$ for all arcs a). However, since this is a special case of flow decomposition with arbitrary gains, we will begin with the more general case first and show what simplifications can be made in the unit gain case later on.

Flows with arbitrary gains in a directed graph G can be decomposed into flows in subgraphs G_i, $i \in \mathbb{N}$ such that each subgraph G_i is either a path, a cycle, a path-cycle, a cycle-path or a bi-cycle. Figure 2.5 depicts these five types of subgraphs. We will formalize and prove this claim in the following theorem. Structurally, our proof follows Wayne [117] and Gondran and Minoux [49], where a version of this theorem with a single source and sink with infinite supplies and demands was shown.

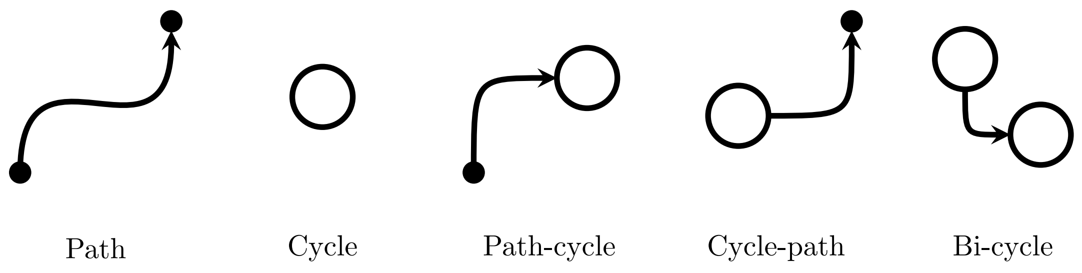


Figure 2.5: The five types of elementary generalized flows.

Theorem 2.2 (Generalized Flow Decomposition)**.** *Let x be a generalized flow in a directed graph G with capacities u, costs x, gains γ and supplies and demands b. Then there exists a family of subgraphs $\mathcal{G} := \{G_1, \ldots, G_j\}$ of G along with flows $x_1, \ldots, x_j$ in*

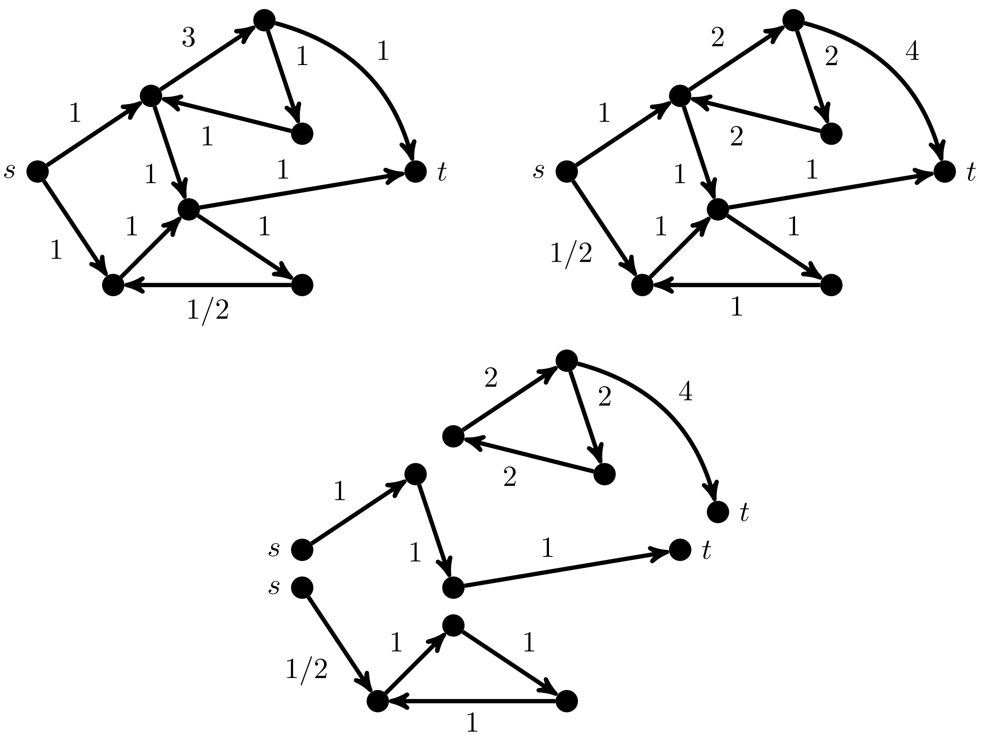


Figure 2.6: A network with gains (top left), a generalized flow in this network (top right) and a decomposition of this flow into elementary generalized flows (bottom).

G that are only positive in their respective subgraphs, i. e., $(x_i)_a = 0$ for $i = 1, \dots, j$ and $a \notin A(G_i)$. Furthermore, the sum of these flows yields the original flow:

$$x_a = \sum_{i=1}^{j} (x_i)_a \quad \text{for all } a \in A(G).$$

The subgraphs $G_1, \dots, G_j$ and flows $x_1, \dots, x_j$ can be chosen such that $j \leq n + m = |V(G)| + |A(G)|$ and G_i is of one of the following types: path, cycle, path-cycle, cycle-path or bi-cycle.

Proof. The idea of this proof is to use the flow conservation constraints to split a flow in an arbitrary graph into flow along the mentioned types of subgraphs. We show this result by induction over the number of arcs with positive flow value $|A'(x)|$, with $A'(x) := \{a \in A(G) \mid x_a > 0\}$. In every step of the decomposition, either $|A'(x)|$ or the number of nodes with finite, non-zero supply or demand $V'(b)$, $V'(b) := \{v \in V(G) \mid b_v \neq 0, -\infty < b_v < \infty\}$ will decrease. If $|A'(x)| = 0$, choosing $\mathcal{P} = \emptyset$ and $\mathcal{C} = \emptyset$ suffices to show the claim. We will show how we can subtract a flow along the mentioned types of subgraphs from x to obtain a new flow x' with new supplies and demands b' with $|A'(x')| < |A'(x)|$ or $|V'(x')| < |V'(x)|$. To this end, we distinguish the following cases.

Case 1: $A'(x)$ is acyclic. Then we can find a path P from a source s to a sink t due to flow conservation. We determine the *bottleneck capacity* u_P of $P = (a_1, \dots, a_{|P|})$ so that u_P is maximal under the constraints that

$$u_P \leq b_s, \quad u_P \gamma_P \leq |b_t|, \quad u_P \prod_{j=1}^{i-1} \gamma_{a_j} \leq x_{a_i} \text{ for all } i = 1, \dots, |P|.$$

Notice that an empty product is 1. This will guarantee that we can reduce as much flow as possible along the path, without creating negative flow values, turning sources into sinks or vice versa. The reduction of flow along P creates a new flow x' by

$$x'_a := \begin{cases} x_{a_i} - u_P \prod_{j=1}^{i-1} \gamma_{a_j} & a = a_i \in A(P) \\ x_a & \text{else} \end{cases} \qquad \text{for all } a \in A(G)$$

and new supplies and demands b' by

$$b'_v := \begin{cases} b_v - u_P & v = s \\ b_v + u_P \gamma_P & v = t \\ b_v & \text{else} \end{cases} \qquad \text{for all } v \in V(G).$$

Since at least one of the constraints in the definition of u_P must be tight, we have $|A'(x')| < |A'(x)|$ or $|V'(x')| < |V'(x)|$ and we found the path P as a subgraph for $\mathcal{G}$.

Case 2: $A'(x)$ contains a cycle. Let $C = (a_1, \dots, a_{|C|})$ be this cycle and let s be the start node of a_1. For this case, we need to distinguish three sub-cases, depending to the gains along the cycle.

Case 2a: C is a unit-gain cycle: $\gamma_C = 1$. In this case, we define its bottleneck capacity u_C to be maximal such that

$$u_C \prod_{j=1}^{i-1} \gamma_{a_j} \leq x_{a_i} \text{ for all } i = 1, \dots, |C|.$$

Again, this allows us to split off as much flow as possible without violating non-negativity constraints. The remainder of this split-off is a new flow x' defined by

$$x'_a := \begin{cases} x_{a_i} - u_C \prod_{j=1}^{i-1} \gamma_{a_j} & a = a_i \in A(C) \\ x_a & \text{else} \end{cases} \qquad \text{for all } a \in A(G).$$

The supplies and demands are the same as before, since we split off flow along an unit-gain cycle. This gives us the cycle C as a subgraph for $\mathcal{G}$ and new flow x' that has less arcs with positive flow value than x by choice of u_C.

Case 2b: C is a flow-generating cycle: $\gamma_C > 1$. We define the bottleneck capacity u_C such that it is maximal under the constraints that

$$u_C \prod_{j=1}^{i-1} \gamma_{a_j} \leq x_{a_i} \text{ for all } i = 1, \ldots, |C|.$$

Splitting off flow along C alone would generate $u_C(\gamma_C - 1)$ additional supplies in s, since C is flow-generating. However, due to flow conservation, there is a (potentially trivial) path $P = (a'_1, \ldots, a'_{|P|})$ from s to either a sink t or a flow-absorbing cycle C'.

Case 2b$_1$: Path P from s leads to a sink t. If P leads to a sink t, we compute a new bottleneck capacity u' such that it is maximal under the constraints that $u' \leq u_C$ and

$$u'\gamma_C\gamma_P \leq |b_t|, \quad u'\gamma_C \prod_{j=1}^{i-1} \gamma_{a'_j} \leq x_{a'_i} \text{ for all } i = 1, \ldots, |P|.$$

C and P form a cycle-path together – the additional constraints ensure that we do not create negative flow values along the paths or use more demands then exists at t (see Figure 2.7). The new flow x' is defined by

$$x'_a := \begin{cases} x_{a_i} - u' \prod_{j=1}^{i-1} \gamma_{a_j} & a = a_i \in A(C) \\ x_{a'_i} - u'\gamma_C \prod_{j=1}^{i-1} \gamma_{a'_j} & a = a'_i \in A(P) \\ x_a & \text{else} \end{cases} \quad \text{for all } a \in A(G).$$

The corresponding new supplies and demands b' are:

$$b'_v := \begin{cases} b_v - u'\gamma_C\gamma_P & v = t \\ b_v & \text{else} \end{cases} \quad \text{for all } v \in V(G).$$

At least one of the constraints in the definition of u' must be tight, so we get $|A'(x')| < |A'(x)|$ or $|V'(x')| < |V'(x)|$ and the cycle-path consisting of C and P for $\mathcal{G}$.

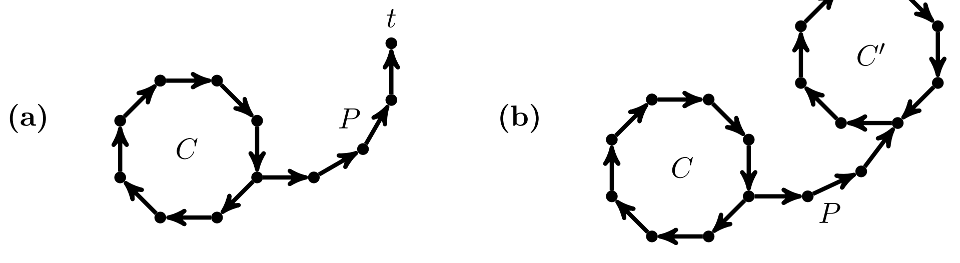


Figure 2.7: The subgraphs and their components for (a) Case 2b$_1$ and (b) Case 2b$_2$.

Case 2b$_2$: Path P from s leads to a flow-absorbing cycle C'. If P leads to a flow-absorbing cycle $C' = (a''_1, \dots, a''_{|C'|})$, we compute u' such that it is maximal under the constraints that $u' \leq u_C$ and

$$u'\gamma_C \prod_{j=1}^{i-1} \gamma_{a_j} \leq x_{a_i} \text{ for all } i = 1, \dots, |P|, \quad u'\gamma_C\gamma_P \prod_{j=1}^{i-1} \gamma_{a''_j} \leq x_{a''_i} \text{ for all } i = 1, \dots, |C'|.$$

C, P and C' form a bi-cycle together (see Figure 2.7). Once again, the additional constraints ensure that no negative flow values are created in the new flow x' which is defined by

$$x'_a := \begin{cases} x_{a_i} - u' \prod_{j=1}^{i-1} \gamma_{a_j} & a = a_i \in A(C) \\ x_{a'_i} - u'\gamma_C \prod_{j=1}^{i-1} \gamma_{a'_j} & a = a'_i \in A(P) \\ x_{a''_i} - u'\gamma_C\gamma_P \prod_{j=1}^{i-1} \gamma_{a''_j} & a = a''_i \in A(C') \\ x_a & \text{else} \end{cases} \qquad \text{for all } a \in A(G).$$

In this case, the supplies and demands are not effected. Since at least one of the constraints in the definition of u' must be tight, we have again $|A'(x')| < |A'(x)|$ and a bi-cycle consisting of C_1, P and C_2 for $\mathcal{G}$.

Case 2c: C is a flow-absorbing cycle: $\gamma_C < 1$. This case can be handled analogously to Case 2b but creates flow along path-cycles instead of cycle-paths.

Analysis. Notice that every case decreases either the number of arcs with positive flow values or the number of nodes with non-zero, finite supply or demand. Thus, we need to apply this case distinction at most $m + n$ times, which gives us the desired bound on the number of subgraphs $G_1, \dots, G_j$. □

If we have unit gains, we have neither flow-generating nor flow-absorbing cycles. Thus, we do not need path-cycles, cycle-paths and bi-cycles for the decomposition, which can be done solely with paths and cycles. This leads to following theorem, which was shown by Ford and Fulkerson [39] and Gallai [42]. In its original form, this was stated for the case of a single source and sink with infinite supply and demand, respectively.

Theorem 2.3 (Flow Decomposition Theorem). *Let x be a generalized flow in a directed graph G with capacities u, costs c and supplies and demands b. Then there exists a family of subgraphs $G_1, \dots, G_j$ of G along with flows $x_1, \dots, x_j$ in G that are only positive in their respective subgraphs, i. e., $(x_i)_a = 0$ for $i = 1, \dots, j$ and $a \notin A(G_i)$. Furthermore, the sum of these flows yields the original flow:*

$$x_a = \sum_{i=1}^{j} (x_i)_a \qquad \textit{for all } a \in A(G).$$

The subgraphs $G_1, \dots, G_j$ and flows $x_1, \dots, x_j$ can be chosen such that $j \leq n + m$ and G_i is either a path or a cycle.

Path-based flows. These flow-decomposition theorems have the interesting aspect that the flow in each subgraph – i. e., each path, cycle, etc.. – can be specified by the flow value of a single arc. The remaining flow values are then uniquely determined by the flow conservation constraints. Thus, we can define variables x_{G_i} that specify the flow value of some designated arc in G_i and gain a description of a flow that is equivalent to giving the flow values of every arc. We refer to this description of a flow as its *path-based* description (even when it contains cycles, cycle-paths, path-cycles or bi-cycles), in contrast to the previously defined *arc-based* description (or *edge-based* in the case of an undirected graph). For subgraphs that consist of a single sequence, we set x_{G_i} to be the flow value of its first arc.

The path-based description is particularly easy in the unit-gain case, since – due to flow conservation constraints – the flow values must be the same for every arc in the path or cycle. Thus, we can just write x_P for the flow value of the path, which is then the flow value of all its arcs. The same holds for unit-gain cycles.

The main advantage of a path-based description is that flow-conservation constraints are guaranteed automatically. However, the number of potential paths and cycles is usually exponential in the size of the graph, which is a serious disadvantage of this kind of flow description.

2.2.3 Residual Graphs

A very important tool for computing flows are *residual graphs* (also called *residual networks*). They consist of a graph and a flow in the graph and provide the capability to send more flow or to undo flow already sent. This feature of cancelling flow is helpful in designing algorithms for flow problems, since it can help to fix decisions that turned out to be suboptimal. We begin by giving a formal definition for a single commodity and arbitrary gains.

Definition 2.4. *Let G be a directed graph G with capacities u, costs c, gains γ and supplies and demands b. Let x be a flow in G. We define a* reverse arc $\overleftarrow{a} = (w,v)$ *for every arc $a = (v,w) \in A(G)$ and denote the set of reverse arcs by $\overleftarrow{A(G)}$. Moreover, we set $\overleftarrow{\overleftarrow{a}} := a$. The residual graph G_x of G and a flow x is defined by*

$$V(G_x) := V(G),$$
$$A(G_x) := \{a \in A(G) \mid x_a < u_a\} \cup \{\overleftarrow{a} \mid a \in A(G),\, x_a > 0\} \subseteq A(G) \cup \overleftarrow{A(G)}.$$

The capacities of the residual graph are referred to as residual capacities u_x *and they are defined by*

$$(u_x)_a := \begin{cases} u_a - x_a & a \in A(G) \\ \gamma_{\overleftarrow{a}} x_{\overleftarrow{a}} & a \in \overleftarrow{A(G)} \end{cases} \quad \text{for all } a \in A(G_x),$$

where costs and gain factors are extended to G_x by:

$$c_{\overleftarrow{a}} := -c_a, \quad \gamma_{\overleftarrow{a}} = \frac{1}{\gamma_a}, \quad \text{for all } a \in A(G).$$

The residual capacities are defined in such a way that for arcs $a \in A(G)$ the residual capacity is exactly the remainder of the capacity of a not already used by x. For arcs $\overleftarrow{a} \in \overleftarrow{A(G)}$, the residual capacity is exactly the amount of flow that is sent by x in a, i. e., the amount of flow that can be cancelled in a. Notice that a residual graph can have parallel arcs due to the reverse arcs. But if this is not desired, we can prevent this as described in Chapter 2.1. Figure 2.8 shows an example of a residual graph.

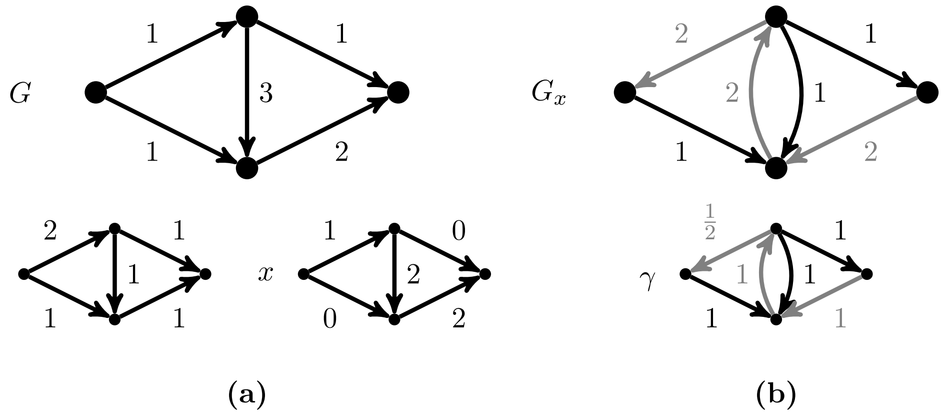


Figure 2.8: (a) A graph G with capacities annotated along side the arcs, and gain factors and a flow below it. (b) The corresponding residual graph G_x and the gain factors for it. Reverse arcs are gray.

2.3 Flows over Time

In this chapter, we will introduce flows that incorporate a temporal dimension. In these flows, flow does not arrive instantaneously at its destination but only after a delay. These flows are called *flows over time* or *dynamic flows*. We begin with the definitions, which we will make for the most general case with multiple commodities and gains. The single commodity case and the case of unit-gains will then be treated as special cases. We will define flows over time in directed graphs first and then extend this to undirected graphs, analogously to our approach for static flows.

2.3.1 Definitions

Recall that we make the assumption that sources do not have incoming arcs and sinks have no outgoing arcs (see Chapter 2.2.1).

Flows over time in directed graphs. Let G be a directed graph with capacities u, transit times τ, costs c, gains γ, supplies and demands b of commodities $i \in K = \{1, \dots, k\}$ and a time horizon T. A *generalized multi-commodity flow over time* f in G is a function $f : A(G) \times K \to ([0, T) \to \mathbb{R}_0^+)$ that assigns a Lebesque-integrable flow rate function $f_a^i : [0, T) \to \mathbb{R}_0^+$ to every arc $a \in A(G)$ for every commodity $i \in K = \{1, \dots, k\}$.

We call the value assigned to an a arc a at a point in time θ for a commodity i the *flow rate* of commodity i at time θ into arc a. The flow rate specifies, as its name indicates, the rate with which the flow is entering the arc. This could for example be the number of cars entering a road at a given point in time. Due to the transit times, flow entering an arc $a = (v, w)$ at time θ will arrive at w at time $\theta + \tau_a$. This flow rate function is typically only defined in $[0, T)$ – however, for the sake of convenience we assume that $f_a^i(\theta) := 0$ for $\theta \notin [0, T)$, $i \in K$, $a \in A(G)$. These flow rate functions have to obey the following constraints:

- *Flow conservation constraints* ensure that we do not use more supplies and demands than we have:

$$\min(0, b_v^i) \leq \sum_{a \in \delta_G^+(v)} \int_0^\theta f_a^i(\xi)\ d\xi - \sum_{a \in \delta_G^-(v)} \int_0^{\theta - \tau_a} \gamma_a f_a^i(\xi)\ d\xi \leq \max(0, b_v^i)$$

  for all $v \in V(G), i \in K, \theta \in [0, T)$. If all supplies and demands are used, we say *all supplies and demands have been fulfilled* and the following equalities hold:

$$\sum_{a \in \delta_G^+(v)} \int_0^T f_a^i(\xi)\ d\xi - \sum_{a \in \delta_G^-(v)} \int_0^{T - \tau_a} \gamma_a f_a^i(\xi)\ d\xi = b_v^i$$

  for all nodes $v \in V(G)$ and commodities $i \in K$.

- *Capacity constraints* make certain that the flow rates assigned to the commodities for an arc do not violate the capacities of an arc:

$$\sum_{i \in K} f_a^i(\theta) \leq u_a \qquad \text{for all } a \in A(G), \theta \in [0, T).$$

- *Non-negativity constraints* are used to guarantee non-negativity of the flow:

$$0 \leq f_a^i(\theta) \qquad \text{for all } a \in A(G), i \in K, \theta \in [0, T).$$

Analogously to static flows, there are two important special cases. The first is the case where we have only a single commodity. In this case, f is called a *single commodity flow over time* and we call f a *generalized flow over time*. The second case is if we have unit gains, i. e., $\gamma_a = 1$ for all arcs $a \in A(G)$. In this setting, we have "normal" flow conservation and refer to f just as a *multi-commodity flow over time*.

Similar to static flows, we refer to the flow rate functions of a commodity $i \in K$ by $f^i : A(G) \to \mathbb{R}, f^i(a) := f_a^i := f(a, i)$. Each $f^i, i \in K$ can be interpreted as a single commodity flow over time. For notational brevity, we omit all the notations for commodities in the single commodity case and write f_a instead of f_a^1, and so on. Furthermore, if it is clear from the context that our flows are multi-commodity and / or generalized, we omit this and call the generalized / multi-commodity flows just flows over time.

The definition of flows over time used above allows flow to arrive at an intermediate node, wait there for some time, and continue onwards. This behaviour of flow is called *storage* of flow or *holdover*. Depending on the problem modelled by the flow, this can be the desired behaviour. However, if storage of flow is a problem – in our example with cars and streets, waiting at nodes would be equivalent to parking on crossings – we cannot use the above definition. In this case, we need additional constraints to forbid waiting at intermediate nodes:

$$\min(0, b_v^i) \leq \sum_{a \in \delta_G^+(v)} f_a^i(\theta) - \sum_{a \in \delta_G^-(v)} \gamma_a f_a^i(\theta - \tau_a) \; d\xi \leq \max(0, b_v^i)$$

for all $v \in V(G), i \in K, \theta \in [0, T)$. Together with the flow conservation constraints from above, these constraints are called *strict flow conservation constraints* and we refer to flows fulfilling them as *flow over time without holdover* or *flow over time without (intermediate) storage*.

The *value* $|f|$ of the flow over time f is defined as the amount of demands of all commodities that f fulfills within the time horizon T:

$$|f| := \sum_{i \in K} |f^i|, \quad |f^i| := \sum_{v \in S_-^i} \sum_{a \in \delta_G^-(v)} \int_0^{T - \tau_a} \gamma_a f_a^i(\xi) \; d\xi.$$

Analogously, the *value* $|f|_\theta$ of a flow over time f until time θ is the amount of flow that has reached the sinks until time θ:

$$|f|_\theta := \sum_{i \in K} |f^i|_\theta, \quad |f^i|_\theta := \sum_{v \in S_-^i} \sum_{a \in \delta_G^-(v)} \int_0^{\theta - \tau_a} \gamma_a f_a^i(\xi) \; d\xi.$$

Thus, $|f| = |f|_T$. The *concurrent value* $|f|_{\text{conc}}$ of f is defined as the minimal fraction of demands of a commodity that is fulfilled by f:

$$|f|_{\text{conc}} := \min_{i \in K} \frac{|f^i|}{|B_-^i|} \in [0, 1].$$

Therefore, the concurrent value is determined by the commodity of which the least fraction of demands has been fulfilled. The *cost* of a flow over time f is defined by

$$c(f) := \sum_{i \in K} c(f^i), \quad c(f^i) := \sum_{a \in A(G)} \int_0^{T - \tau_a} c_a f_a^i(\xi) \; d\xi.$$

The *arrival pattern* of a flow over time f is the function $p_f : [0, T) \to \mathbb{R}_0^+, p(\theta) := |f|_\theta$ that assigns to every point in time θ the total amount of flow that has arrived at the sinks until time θ.

Let f_θ^{max} be a flow over time in a graph G with time horizon θ that maximizes $|f|_\theta$ over all flows over time for G. We define $p_G : [0, T) \to \mathbb{R}_0^+, p_G(\theta) := |f_\theta^{max}|_\theta$ as the function that specifies for every point in time θ the maximum value of flow that can be sent to the sinks in G until time θ. The function p_G is called the *earliest arrival pattern* of G. We write p instead of p_G if G can be inferred from the context.

Flows over time in undirected graphs. We define flows over time in undirected graphs by using the same transformation into directed graphs that we employed to define static flows in undirected graphs in Section 2.2.1. This time, we have an undirected graph G with capacities u, costs c, gains γ, transit times τ and supplies and demands b. The modification into a directed graph G' is once again done by replacing every edge $e = \{v, w\} \in E(G)$ with the construct shown in Figure 2.9.

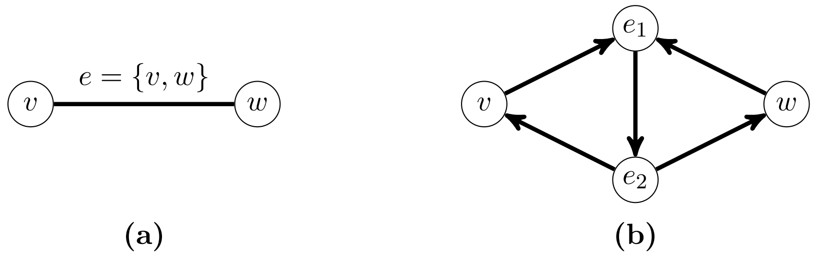


Figure 2.9: (a) An undirected edge and (b) a gadget of directed arcs as its replacement.

G' is defined by

$$V(G') := V(G) \cup \{ e_1, e_2 \mid e \in E(G)\},$$
$$A(G') := \{ (v, e_1), (w, e_1), (e_1, e_2), (e_2, v), (e_2, w) \mid e \in E(G)\}.$$

Capacities u', costs c', gains γ' and transit times τ' are defined by

$$u'_a := \begin{cases} u_e & a = (e_1, e_2) \\ \infty & \text{else} \end{cases}, \quad c'_a := \begin{cases} c_a & a = (e_1, e_2) \\ 0 & \text{else} \end{cases},$$
$$\gamma'_a := \begin{cases} \gamma_a & a = (e_1, e_2) \\ 1 & \text{else} \end{cases}, \quad \tau'_a := \begin{cases} \tau_a & a = (e_1, e_2) \\ 0 & \text{else} \end{cases}$$

for all $a \in A(G')$. Supplies and demands b' are defined for all $v \in V(G')$ and $i \in K$ by

$$b'^i_v := \begin{cases} b^i_v & v \in V(G) \\ 0 & \text{else} \end{cases}.$$

As in the static transformation, the attributes of the original edge are transferred to (e_1, e_2) with the other arcs and new nodes getting neutral attributes.

Forcing flow between v and w to travel through (e_1, e_2) enforces a common capacity constraint. However, the construction has the draw back that it creates v- and w-cycles. We will see later that we only use this construction for problems that do not benefit from the new cycles, which makes it fine for our purposes.

2.3.2 Continuous and Discrete Time Models

Our definitions of flows over time are based on assigning Lebesque-integrable flow rate functions $f^i_a : [0, T) \to \mathbb{R}^+_0$ to every arc a for commodity i. This assumes a *continuous* time model, as the flow rate can be different at any time $\theta \in [0, T)$. In practical applications, we will usually not see the phenomenon that flow rates change infinitely many times. Therefore, *discrete* time models are often used as well. In a discrete time model, we assign flow rate functions $f_a^i : \{0, 1, \ldots, T-1\} \to \mathbb{R}_0^+$ to every arc a for commodity i. It is then assumed that the flow rate is constant in a time interval $[\theta, \theta+1)$ and specified by $f_a^i(\theta)$. Thus we can choose only T flow rates per arc and commodity, instead of infinitely many ones. However, this is normally no big restriction, if we can choose a sufficiently fine discretization of time. In most cases, results for one time model carry over to the other (e. g., Fleischer and Tardos [33] or Fleischer and Skutella [32]). We will study these equivalence in greater detail in Chapter 4.1, where we will see that results in a discrete time expansion carry over to flows over time with a continuous time model (under some restrictions, of course).

2.4 Approximating Flows over Time

We conclude our preliminaries with a short introduction into approximation concepts for flows over time. Normally, a c-approximation algorithm for an optimization problem is defined as follows (Williamson and Shmoys [119]).

Definition 2.5. *A* c-approximation algorithm *for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of c of the value of an optimal solution. If OPT is the value of an optimal solution to our optimization problem, then a c-approximation algorithm computes a solution with a value of at most $OPT \cdot c$ if the optimization problem is a minimization problem. If the optimization problem is a maximization problem, a c-approximation algorithm computes a solution with a value of at least OPT/c.*

We already mentioned in the introduction that for flows over time there are generally multiple avenues for approximation. Some of them fit directly with the definition of approximation algorithm as above. We begin with flow-approximations, as in the introduction.

Definition 2.6. *A* c-flow-approximation *algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution which needs to fulfill the following flow-related criteria depending on the objective:*

- *If the objective is to maximize the flow value, the flow value of the solution needs to be at least $1/c$ times the flow value of an optimal solution.*
- *If the objective is to minimize the time horizon or the costs, the solution needs to be at least as good as the optimal solution, but is allowed to fulfill only $1/c$ of the demands and use only $1/c$ of the supplies.*

These two different criteria stem from the fact that in the first case, we approximate optimality, and in the second feasibility. Usually, approximating optimality might seem more intuitive, but the approach of approximating optimality is impractical for problems where finding any feasible solution is NP-hard. One prominent example is the minimum bounded degree spanning tree problem. For such problems, there is still the option of approximating feasibility, i. e., we allow our solutions to violate constraints slightly. E. g., for the minimum bounded degree spanning tree problem, it is possible to give approximation algorithms that violate the degree bound slightly [45, 103].

Flows over time have a slightly different problem, namely that a description of a solution can be exponential in the input size. This is due to the fact that we might need to specify many different flow rates for an arc at different time points. As a consequence of this, many approximation algorithms for flows over time approximate the temporal dimension. We therefore define time-approximations next.

Definition 2.7. *A* c-time-approximation *algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution to fulfill the following time-related criteria depending on the objective:*

- *If the objective is to minimize the time horizon, the time horizon of the solution needs to be at most c times the time horizon of an optimal solution.*
- *If the objective is to maximize the flow value or to minimize the costs, the solution needs to be at least as good as an optimal solution, but it is allowed to use a time horizon that is longer by a factor of c.*

In some cases, we will see algorithms that approximate both time and value. These *bicriteria* approximation algorithms are defined as follows.

Definition 2.8. *A* c-time-c'-value-approximation *algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution to fulfill the following time- and value-related criteria depending on the objective:*

- *If the objective is to minimize the time horizon, the time horizon of the solution needs to be at most c times the time horizon of an optimal solution, and is allowed to fulfill only* $1/c'$ *of the demands and use only* $1/c'$ *of the supplies.*
- *If the objective is to maximize the flow value, the solution needs to be at least* $1/c'$ *times the flow value of an optimal solution and is allowed to use a time horizon that is longer by a factor of c.*

For all of these settings, we would like to have approximation algorithms whose approximation guarantees are as good as possible. Ideally, this would mean that we find an algorithm A_ε with an approximation guarantee of $1 + \varepsilon$ for any $\varepsilon > 0$. If we can do so, we have a *polynomial-time approximation scheme* (PTAS).

Definition 2.9 ([119])**.** *A* polynomial-time approximation scheme *is a family of algorithms* $\{A_\varepsilon\}$ *such that there is an algorithm for each* $\varepsilon > 0$*, such that* A_ε *is a* $(1 + \varepsilon)$*-approximation algorithm.*

However, with a PTAS, we have no guarantee how the running time of the algorithms behaves if ε becomes very small. Even more desirable are therefore *fully polynomial-time approximation schemes* (FPTAS).

Definition 2.10. *A* fully polynomial-time approximation scheme *is a family of algorithms $\{A_\varepsilon\}$ such that there is an algorithm for each $\varepsilon > 0$, such that A_ε is a $(1+\varepsilon)$-approximation algorithm and the running time of A_ε is polynomial in the input size of the problem and $1/\varepsilon$.*

3 Flow over Time Problems

In this chapter, we introduce the flow over time problems which we study in this thesis. For every problem, we provide formal definitions and a brief history. We give an overview of the state of art for each problem and highlight the contributions of this thesis to the respective problems.

The organization of this chapter is as follows. We begin by dealing with the classical problems, which have only a single objective and just consist of a flow over time computation. We classify these problems by the number of commodities and the attributes they use, leading to the following structure for this chapter:

- In Section 3.1 we study problems that have capacities, transit times and supplies and demands of a single commodity. The goal is either to send as much flow as possible within a given deadline, or to send a specified amount of flow as fast as possible. Costs or gains are not considered here. Examples for this type of problem are the maximum flow over time or the quickest flow over time problem.

- Section 3.2 deals with problems which have supplies and demands of multiple commodities in addition to capacities and transit times (but still no costs or gains). The goals are similar to the previous section – in fact, the problems we consider here are the multi-commodity counterparts to the problems of the last section. Aside from introducing these problems, we discuss a peculiarity of multi-commodity flows over time in this section: letting flow wait can allow us to send more flow or to send flow faster, which might seem counter-intuitive at first.

- Section 3.3 considers problems that have costs in addition to capacities, transit times and supplies and demands. The objective is then to find a flow over time that fulfills all supplies and demands within a given time, such that the costs of the flow are minimal. The resulting family of problems are minimum cost flow over time problems. We consider both single and multiple commodity versions of these problems here.

- Section 3.4 deals with problems that add gains to capacities, transit times and supplies and demands. The objectives remain the same as in Section 3.1. These problems are called generalized maximum flow over time problems.

After these problems, we study two types of more complex problems, which are particularly relevant to evacuations. The first type are earliest arrival flows, who try to maximize the flow sent at multiple points in time instead of a single point. The second type are problems in undirected graphs, where we are allowed to orient the graph before solving a flow over time problem, making this a class of two stage problems.

- Section 3.5 studies the mentioned earliest arrival flows, where we are looking for flows that maximize the amount of flow sent at all points in time. Since such flows are not guaranteed to exist, we characterize the existence of such flows as well as algorithms for finding them.
- Section 3.6 deals with problems in undirected graphs, where we are allowed to orient edges in a preprocessing step. Here we study the influence of the orientations on the quality of the flow over time solutions.

3.1 Single Commodity Problems without Costs or Gains

In this chapter we will introduce flow over time problems that just use capacities, transit times and supplies and demands of a single commodity, and nothing more. We categorize these problems into maximum flow problems, where we have infinite supplies and demands and want to send as much as possible, and transshipment problems, where we have specific, finite amounts of supplies and demands that need to be shipped as fast as possible.

3.1.1 Maximum Flows over Time

We begin with the problems that have infinite supplies and demands. Infinite supplies and demands have the advantage that we can assume without loss of generality, that we have only a single source and a single sink. This is because we can add a *super-source* and *super-sink*, which have infinite supply and demand, respectively. These *super-terminals* are then connected to the sources and sinks in the original graph, using arcs with infinite capacities and zero transit time. Figure 3.1 sketches this construction. For reference purposes, we make the following observation.

Observation 3.1. *Without loss of generality, we can assume that for every commodity the number of sources with infinite supply and sinks with infinite demand is 1.*

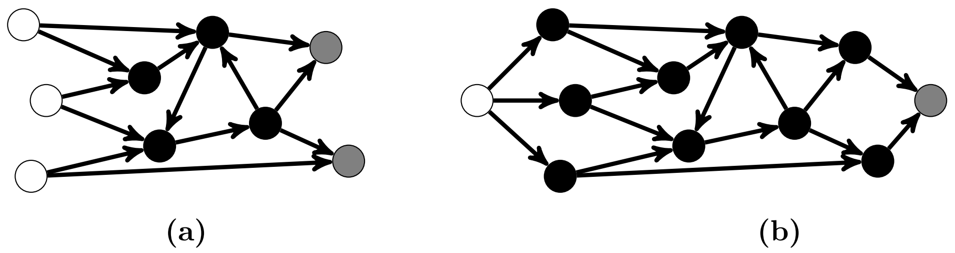


Figure 3.1: (a) A graph with multiple sources (white) and sinks (gray). (b) The same graph with an added super-source and -sink.

We will see in the next section that such a construction cannot be used with finite supplies and demands.

A natural question in this setting is to ask how much flow can be send from the sources to the sinks within a given time horizon T. This problem is called the *maximum flow over time problem* and was introduced by Ford and Fulkerson [38] in the 1950's.

MAXIMUM FLOW OVER TIME PROBLEM

Input:	A directed graph G with capacities u, transit times τ, infinite supplies and demands b of a single commodity and a time horizon T.
Output:	A flow over time f in G maximizing the flow sent $\|f\|$.

They proposed two techniques for dealing with them, both of which reduce the problem to known static flow problems. The first reduction uses a technique called *time expansion* to remove the temporal dimension, transforming the problem into a static maximum flow problem. The downside of this technique is that it causes a large blow-up in the size of the graph – the input for the static maximum flow problem is larger by a factor that is linear in the time horizon, resulting in a pseudo-polynomial running time for this approach. Details on time expansion can be found in Chapter 4.

The second reduction generates a static minimum cost flow problem by transforming the transit times into costs. The solution to the minimum cost flow problem can be decomposed into a family of paths (see Theorem 2.3), along which flow is send for as long as the time horizon and the path lengths permit. Such flows are called *temporally repeated flows*. This technique exploits that we have a single source and sink with infinite supplies and demands, respectively. This allows us to send as much flow as we want without risking that a source might run out of supply or a sink run out of demand. In contrast to time expansion, there is no blow-up involved. Together with a strongly polynomial algorithm for the static minimum cost problem (e. g., Orlin [85]), this yields a strongly polynomial algorithm for the maximum flow over time problem. This algorithm has also the properties that no storage of flow is necessary for an optimum solution and that it produces an optimal solution that sends flow only along simple paths.

3.1.2 Quickest Flows and Transshipments

In this section we will deal with problems where finite amounts of supplies and demands are given, and we have to fulfill them all as fast as possible. In the case of finite supplies and demands, we cannot make the assumption that we have only a single source and sink. Adding super-terminals is not helpful here, as shifting supplies and demands to them leaves us with no guarantee that we get a flow that respects the original supplies and demands. Notice that adding finite capacities to the construction of Observation 3.1 does not work, as we would need capacities that limit the total flow over an arc in the time interval $[0, T)$, which we do not have in our model.

Thus, it makes sense to distinguish problems with a single source and sink and problems with multiple sources and sinks. We begin with the single source and sink case. Since we are given specific supplies and demands, the question in this setting is usually to determine the smallest time horizon such that all supplies and demands can be fulfilled. One application for this are evacuation scenarios [13, 20, 22, 56]. In this setting, we have some evacuees who need to be brought to safe locations and usually, the sooner the better. This problem is referred to as the *quickest flow problem* or *quickest flow over time problem.*

QUICKEST FLOW PROBLEM

Input: A directed graph G with capacities u, transit times τ, finite supplies and demands b of a single commodity, such that there is only a single source and sink.

Output: The minimal time horizon T such that a flow over time f with time horizon T can fulfill all supplies and demands.

Burkard, Dlaska and Klinz [18] gave a strongly polynomial algorithm for the quickest flow problem by embedding the strongly polynomial algorithm of Ford and Fulkerson [38] for the maximum flow over time problem into Megiddo's parametric search framework [77]. Since this algorithm builds on Ford and Fulkerson's algorithm, it does neither require storage nor sending flow along non-simple paths. Using time expansion to obtain a pseudo-polynomial algorithm would work as well.

Notice that we can obtain an upper bound for the minimal time horizon, if necessary, through the given transit times and supplies and demands. The transit time of a simple path in a graph G is at most $n \cdot \max_{a \in A(G)} \tau_a$ and by the algorithm of Burkard, Dlaska and Klinz we know that we do not require non-simple paths for an optimal solution. Therefore, we do not have to use paths that have a transit time larger than $n \cdot \max_{a \in A(G)} \tau_a$. Furthermore, we can use any path P for at most $B_+ / \min_{a \in P} u_a$ time units before supplies and demands have been exhausted. This gives us a bound for T that is pseudo-polynomial in the input, which we formalize in Observation 3.2. This observation can be extended to case of multiple sources and sinks (Hoppe [65]).

Observation 3.2. *The minimal time horizon necessary to fulfill all supplies and demands in a graph G with capacities u, transit times τ and supplies and demands b is at most $n \cdot \max_{a \in A(G)} \tau_a + \frac{B_+}{\min_{a \in P} u_a}$.*

If we consider the application of evacuations, having only a single source and sink is quite restrictive – usually, not all the evacuees are grouped up but spread over an area. Also, there might be more than a single safe location for them to go. Thus, we would prefer to solve the version with multiple sources and sinks, which is called the *quickest transshipment problem* or *quickest transshipment over time problem.* The version with multiple sources and single sink is sometimes referred to as *evacuation problem* [13, 20]. In this case, the single sink is usually justified because the sink represents something like the whole outside of a building – for the evacuation, it is important to get the people out, but the specific destination outside is not so important.

QUICKEST TRANSSHIPMENT PROBLEM

Input: A directed graph G with capacities u, transit times τ, finite supplies and demands b of a single commodity.

Output: The minimal time horizon T such that a flow over time f with time horizon T can fulfill all supplies and demands.

Again, using time expansion combined with a search framework yields a pseudo-polynomial algorithm. Finding a strongly polynomial algorithm is not so easy, however, since we cannot use Ford and Fulkerson's algorithm – we cannot introduce super-terminals easily, and our supplies and demands can run out. Thus, we need another algorithm to solve the corresponding feasibility problem for a given time horizon, which is called the *transshipment over time problem.*

TRANSSHIPMENT OVER TIME PROBLEM

Input: A directed graph G with capacities u, transit times τ, finite supplies and demands b of a single commodity, a time horizon T.

Output: YES, if a flow over time f exists that can fulfill all supplies and demands within time horizon T, NO otherwise.

Hoppe and Tardos [65, 67] managed to find a strongly polynomial algorithm for this feasibility problem, which can then be combined with Meggido's search framework. This algorithm produces an optimal solution that does not require storage and uses only simple paths. However, their algorithm relies on submodular function minimization as a subroutine. While submodular function minimization can be done in strongly polynomial time, all known algorithms have a high running time – the recently presented algorithm by Orlin [86], which improved the best previously known bound by more than a factor of n, still has a running time of $O(n^5\text{EVAL} + n^6)$. EVAL is here the time necessary for an evaluation of the submodular function that is to be minimized, which is a static maximum flow computation in our case. Weakly polynomial-time algorithms can improve the exponent of the n a bit, but for practical purposes, the pseudo-polynomial algorithm based on time expansion can easily be faster for this problem. Here we will discuss the time expansion approach in Chapter 4, but not the submodular function based one – it seems not to be as easily extendable to other flow problems as time expansion, and the obtained algorithms have very high running times.

3.1.3 Summary

We conclude this chapter by a summary of the problems presented here. These problems are listed in Table 3.1. Notice that the dash in the objective column signifies feasibility problems – such problems have no specific objective function. The results known for these problems are listed in Table 3.2. We do not mention pseudo-polynomial algorithms based on time expansion there, as all of the problems presented here have such algorithms.

Problem	Objective	Comment
Maximum Flow over Time	max $\|f\|$	Infinite supplies and demands
Transshipment over Time	—	Finite supplies and demands
Quickest Flow	min T	Single source and sink
Quickest Transshipment	min T	Multiple sources and sinks

Table 3.1: An overview of the flow over time problems with a single commodity and neither costs nor gains.

Problem	Complexity	Technique
Maximum Flow over Time	**P**	*Static minimum cost flow* Ford, Fulkerson [38]
Transshipment over Time	**P**	*Submodular function minimization* Hoppe, Tardos [65, 67]
Quickest Flow	**P**	*Max. Flow over Time + Meggido* Burkard, Dlaska, Klinz [18]
Quickest Transshipment	**P**	*Trans. over Time + Meggido* Hoppe, Tardos [65, 67]

Table 3.2: An overview of results for the flow over time problems with a single commodity and neither costs nor gains.

3.2 Multiple Commodity Problems without Costs or Gains

The problems we considered so far had only a single commodity. However, we often require multiple commodities in order to model applications. Some recent examples where problems that can be modelled as multi-commodity flows over time occur include aircraft routing [99], cloud computing [1, 105], disaster management [57, 88], evacuations [76], logistics [3, 62], packet routing [90] and shared-ride systems [16].

In this chapter, we will first introduce multi-commodity flow over time problems and then study an important phenomenon that does not occur with single commodity flows over time and is related to storage in nodes.

3.2.1 Multi-commodity Maximum Flows over Time

We begin with the introduction of the multi-commodity version of the maximum flow over time problem, which is called the *multi-commodity maximum flow over time problem.*

Multi-commodity Maximum Flow over Time Problem

Input: A directed graph G with capacities u, transit times τ, infinite supplies and demands b of k commodities and a time horizon T.

Output: A flow over time f in G maximizing the flow sent $|f|$.

We will see later (Chapter 3.2.4) that storage makes a difference for problems with multiple commodities – it can allow us to send more flow in the same time horizon in some instances. Since storage is not desired in some applications, we define the multi-commodity problems in two versions – the normal one, where storage is allowed, and another one, where storage is forbidden.

Multi-commodity Maximum Flow over Time Problem without Storage

Input: A directed graph G with capacities u, transit times τ, infinite supplies and demands b of k commodities and a time horizon T.

Output: A flow over time without storage f in G maximizing the flow sent $|f|$.

Analogously to the single commodity case, we can assume that there is a single source and sink for every commodity. The multi-commodity maximum flow problem with or without storage can be solved by time expansion – this transforms the problem into a static multi-commodity maximum flow problem, which can be formulated as an linear program. Although there is no strongly polynomial algorithm known that solves arbitrary linear programs, Tardos [108] showed that for linear programs representing multi-commodity maximum flow problems there is a strongly polynomial algorithm. This works for both the version with and without storage. However, time expansion causes a pseudo-polynomial blow-up of the graph.

The question whether a better, efficient algorithm for a version of this problem exists was answered by Hall, Hippler and Skutella [55]. They showed by an reduction from PARTITION that both versions of this problem are weakly NP-hard. Their reduction also holds for only two commodities that have only a single source and sink, and series-parallel graphs. In this sense, a pseudo-polynomial algorithm is the best we can hope for, at least as far as exact algorithms are concerned – unless P=NP should be true.

However, the result of Hall, Hippler and Skutella still permits efficient approximation algorithms. Since this problem is easy in the static case, it makes sense to approximate the temporal dimension. One way to do this was proposed by Fleischer and Skutella [32]: they introduced the concept of *condensed time-expanded graphs*, which rely on rounded transit times that lead to a rougher discretization of time, which in turn leads to a smaller time expansion. More on this technique can be found in Chapter 5.1.

This technique leads to an FPTAS for this problem, as long as optimal solutions exist that use only simple paths. While this is the case if storage is allowed, it is not if storage is forbidden – running around in cycles can simulate waiting, which can be advantageous (see also Chapter 3.2.4).

While the non-simple paths cannot be handled by normal condensed time-expanded networks, it is possible to obtain an FPTAS for the problem with forbidden storage by using *sequence-condensed time-expanded graphs*, which do not round transit times of individual arcs, but only transit times of small arc sequences. This was proposed by Groß and Skutella [52]. This technique is described in detail in Chapter 5.3 and is one of the contributions of this thesis. The previously best known approximation algorithm by Fleischer and Skutella [32] had a guarantee of $2 + \varepsilon$ and was based on temporally repeated flows.

3.2.2 Multi-commodity Quickest Flows over Time

For the case of finite supplies and demands, we get the *multi-commodity flow over time problem* as counterpart to the transshipment over time problem and the *multi-commodity quickest flow problem* as analogon to the quickest transshipment problem. Notice that in the setting of multiple commodities, we call problems with multiple sources and sinks flow problems instead of transshipment problems – this is because they have been established this way [55]. We will see later that the number of sources and sinks has no big influence on the computational complexity, which justifies that we do not differ between single and multiple terminal problem versions.

MULTI-COMMODITY FLOW OVER TIME PROBLEM

Input: A directed graph G with capacities u, transit times τ, finite supplies and demands b of a k commodities, a time horizon T.

Output: YES, if a flow over time f exists that can fulfill all supplies and demands within time horizon T, NO otherwise.

MULTI-COMMODITY QUICKEST FLOW PROBLEM

Input: A directed graph G with capacities u, transit times τ, finite supplies and demands b of k commodities.

Output: The minimal time horizon T such that a flow over time f in G exists that fulfills all supplies and demands.

As it was the case for multi-commodity maximum flows over time, allowing or forbidding storage can make a big difference. We will see in Chapter 3.2.4 that a flow without storage might take twice as long to fulfill the same supplies and demands. Thus, we study these problems both with and without storage.

MULTI-COMMODITY FLOW OVER TIME PROBLEM WITHOUT STORAGE

Input: A directed graph G with capacities u, transit times τ, finite supplies and demands b of a k commodities, a time horizon T.

Output: YES, if a flow over time without storage f exists that can fulfill all supplies and demands within time horizon T, NO otherwise.

MULTI-COMMODITY QUICKEST FLOW PROBLEM WITHOUT STORAGE

Input: A directed graph G with capacities u, transit times τ, finite supplies and demands b of k commodities.

Output: The minimal time horizon T such that a flow over time without storage f in G exists that fulfills all supplies and demands.

The results for this problem are very similar to the results for the multi-commodity maximum flow over time problem. Once again, we can use time expansion to remove the temporal dimension. This transforms the multi-commodity flow over time problem into a static multi-commodity flow problem, which can be formulated as a linear program. This works both for the case with and without storage at the cost of a pseudo-polynomial blow-up.

The reduction from Hall, Hippler and Skutella [55] can be used to show that the multi-commodity flow over time problem (and consequently, the multi-commodity quickest flow problem) are weakly NP-hard. The reduction retains the property that it works in the case of two commodities with a single source and sink each, and series-parallel graphs.

For approximation algorithms, the condensed time-expanded graphs (Chapter 5.1) of Fleischer and Skutella [32] give an FPTAS for the multi-commodity flow over time problem with storage, and the sequence-condensed time-expanded graphs (Chapter 5.3) of Groß and Skutella [52] give an FPTAS for the version without storage. Notice that both FPTAS approximate the time horizon, i. e., they relax the feasibility of this problem. In fact, the sequence condensation approximates both time and value.

The $2 + \varepsilon$-approximation algorithm of Fleischer and Skutella [32] using temporally repeated flows can also be used here. The multi-commodity quickest flow problem can be solved by combining an algorithm for the multi-commodity flow over time problem with a search framework (e. g., Megiddo [77]). An upper bound for the minimal time horizon can, for example, be obtained through the $2+\varepsilon$-algorithm by Fleischer and Skutella [32]. Depending on the algorithm used for multi-commodity flow over time problem, this yields an pseudo-polynomial exact algorithm or FPTAS for the multi-commodity quickest flow problem.

3.2.3 Maximum Concurrent Flows over Time

A multi-commodity flow over time problem which has no single commodity counterpart is the *maximum concurrent flow over time problem.* In this problem, we have multiple commodities and might not be able to fulfill all supplies and demands in the given time horizon. Here, we are interested in the fraction of the demands that we can fulfill. More specifically, we want to find the maximum value x, such that there is a flow over time that fulfills at least x-times the demand of every commodity. x is then the concurrent flow value $|f|_{\text{conc}}$.

Maximum Concurrent Flow over Time Problem

Input: A directed graph G with capacities u, transit times τ, finite supplies and demands b of a k commodities and a time horizon T.

Output: A flow over time f in G maximizing the concurrent flow value $|f|_{\text{conc}}$.

This problem is also weakly NP-hard as a consequence of the multi-commodity flow over time problem being weakly NP-hard [55]. Since this problem can be formulated as an LP, we can apply time expansion and condensation to obtain exact pseudo-polynomial algorithms and FPTASs approximating the time horizon. However, we will place our main focus on the other problems that we introduced and only skim this one.

3.2.4 The Importance of Storage

We have already mentioned that for multi-commodity flows over time, storage can make a difference. In this section, we will study and give tight bounds on the scale of this effect. In particular, we are interested in the minimal time horizon necessary to send given supplies and demands in a graph if we allow or forbid storage. Obviously, allowing storage cannot be worse than forbidding storage, since we only get more options. Let T^*_{st} be the minimal time horizon to fulfill all supplies and demands with storage, and T^* be the minimal time horizon without storage. We are interested in the ratio T^*/T^*_{st}, which we will refer to as the *benefit of storage*.

This effect was already studied a decade ago by Fleischer and Skutella [30], who did not name this effect. They managed to show that the benefit of storage is at most 2 for any instance. Furthermore, they gave an instance where the benefit of storage is 4/3. Recently, Groß and Skutella showed that the factor of 2 for the benefit of storage is tight. In this section, we will discuss the lower bound results by Fleischer and Skutella [30] and Groß and Skutella. The upper bound result by Fleischer and Skutella [30] consists essentially of an algorithm which computes a flow over time without storage that takes at most twice as long as a flow with storage to fulfill all supplies and demands (in any instance).

A 4/3-lower bound on a path. We begin with the lower bound of 4/3 by Fleischer and Skutella [30], where an example without a formal proof is given. It is also still the best known bound for acyclic graphs – the improvement by Groß and Skutella requires a cycle.

Theorem 3.3. *There are instances based on a path for the quickest multi-commodity flow problem, where the benefit of storage is* 4/3 *– i. e., the time horizon necessary to fulfill all supplies and demands is a factor of* 4/3 *larger without storage than with it.*

Proof. Consider the instance depicted in Figure 3.2.

This example has three commodities, each with a single source and sink and a unique source-sink path. Therefore, we have only to determine when we send flow into a path and where we let flow wait. If we allow storage, we can send flow from the first and third

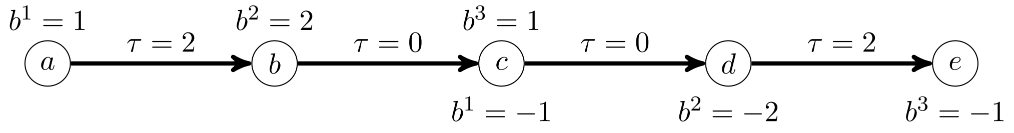


Figure 3.2: An example where storage allows us to fulfill all supplies and demands faster. Transit times and supplies and demands are zero unless annotated otherwise, capacities are all 1.

commodity into their respective paths during $[0, 1)$ and do not let any of this flow wait. For the second commodity, we send flow into its path during the interval $[0, 2)$ and we let the flow sent during $[0, 1)$ wait at node c for two time units. Then all supplies and demands have been fulfilled at time 3.

If we forbid storage, we cannot use the waiting at c. Due to capacity constraints and the fact that flow cannot wait, we get the following inequalities:

$$f^1_{(b,c)}(\theta) + f^2_{(b,c)}(\theta) \leq 1, \quad f^2_{(c,d)}(\theta) + f^3_{(c,d)}(\theta) = f^2_{(b,c)}(\theta) + f^3_{(c,d)}(\theta) \quad \leq 1 \quad \text{for all } \theta \geq 0.$$

If all supplies and demands are fulfilled, the following equalities must hold:

$$\int_2^T f^1_{(b,c)}(\theta)\ d\theta = \int_0^{T-2} f^3_{(c,d)}(\theta)\ d\theta = 1, \quad \int_0^T f^2_{(b,c)}(\theta)\ d\theta = 2.$$

The interval boundaries are a consequence of the transit times of the arcs. Due to $\tau_{(a,b)} = 2$, $f^1_{(b,c)}$ has to be zero in $[0, 2)$. Similarly, $f^3_{(c,d)}$ has to be zero in $[T - 2, T)$. Furthermore, we know that three flow units must traverse through $(b, c), (c, d)$:

$$\int_2^T \underbrace{f^1_{(b,c)}(\theta) + f^2_{(b,c)}(\theta)}_{\leq 1}\ d\theta + \int_0^2 f^2_{(b,c)}(\theta)\ d\theta = 3,$$

$$\int_0^{T-2} \underbrace{f^2_{(b,c)}(\theta) + f^3_{(c,d)}(\theta)}_{\leq 1}\ d\theta + \int_{T-2}^T f^2_{(b,c)}(\theta)\ d\theta = 3.$$

Adding these equalities and replacing the first integrals with upper bounds yields

$$2(T-2) + \underbrace{\int_0^2 f^2_{(b,c)}(\theta)\ d\theta + \int_{T-2}^T f^2_{(b,c)}(\theta)\ d\theta}_{=2} \geq 6.$$

if all supplies and demands are fulfilled. This yields $T = 4$ as minimal time horizon if storage is forbidden, completing the proof. □

Furthermore, it can be beneficial for a multi-commodity flow over time to run in cycles, if storage is forbidden, as we can see by a slight extension of the above example. The cycle can then act as a replacement for waiting in a node, see for example Figure 3.3.

Observation 3.4. *There are instances for the quickest multi-commodity flow problem without storage where solutions using non-simple paths can fulfill all supplies and demands faster than solutions which use simple paths only.*

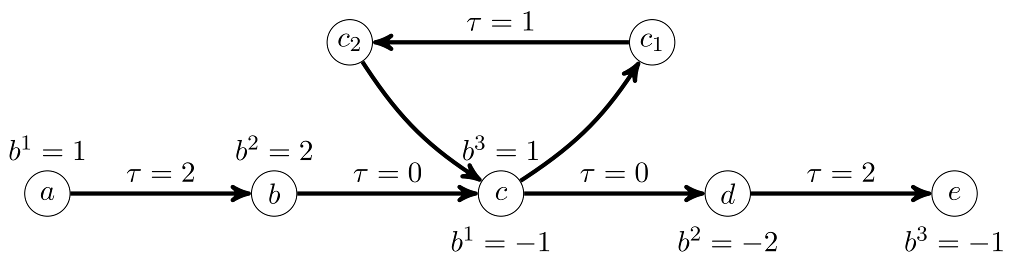


Figure 3.3: An example where storage allows us to fulfill all supplies and demands faster. Transit times and supplies and demands are zero unless annotated otherwise, capacities are all 1.

A 2-lower bound on a cycle. This 4/3-lower bound can be improved by using cycles. This result is due to Groß and Skutella, which is unpublished as of yet.

Theorem 3.5. *There is a family of instances based on a cycle for the quickest multicommodity flow problem, where the benefit of storage converges against* 2*.*

Proof. Consider the following family of directed graphs G_k, $k \in \mathbb{N}$ defined by

$$\begin{aligned} V(G_k) &:= \{v_1, v_2, \ldots, v_k\}, \\ A(G_k) &:= \{a_1 = (v_1, v_2), \ldots, a_{k-1} = (v_{k-1}, v_k), a_k = (v_k, v_1)\}. \end{aligned}$$

All arcs get unit transit times and unit capacities. There are k commodities with the following supplies and demands for a node $v \in V(G_k)$ for a commodity $i \in \{1, \ldots, k\}$:

$$b_v^i := \begin{cases} 2 & i = 1, v = v_2, \\ -2 & i = 1, v = v_1, \\ 1 & i > 1, v = v_{i \bmod k+1}, \\ -1 & i > 1, v = v_i, \\ 0 & \text{else.} \end{cases}$$

Figure 3.4 illustrates such an instance.

We will now analyze the time horizon necessary to fulfill all commodities in $G_k, k \to \infty$ and show that if storage is permitted, this can be done within a time horizon of $k+1$; but if storage is forbidden, we require a time horizon of more than $2k-2$. We show this in two steps in Lemma 3.6 and Lemma 3.7, which proves that the benefit of storage converges against 2 due to $\lim_{k\to\infty}(2k-2)/(k+1) = 2$. □

We begin by proving that we can fulfill all supplies and demands within a time horizon of $k+1$, if storage is permitted.

Lemma 3.6. *There is a flow over time with storage f with time horizon $k+1$ in G_k, $k \geq 3$ that fulfills all supplies and demands.*

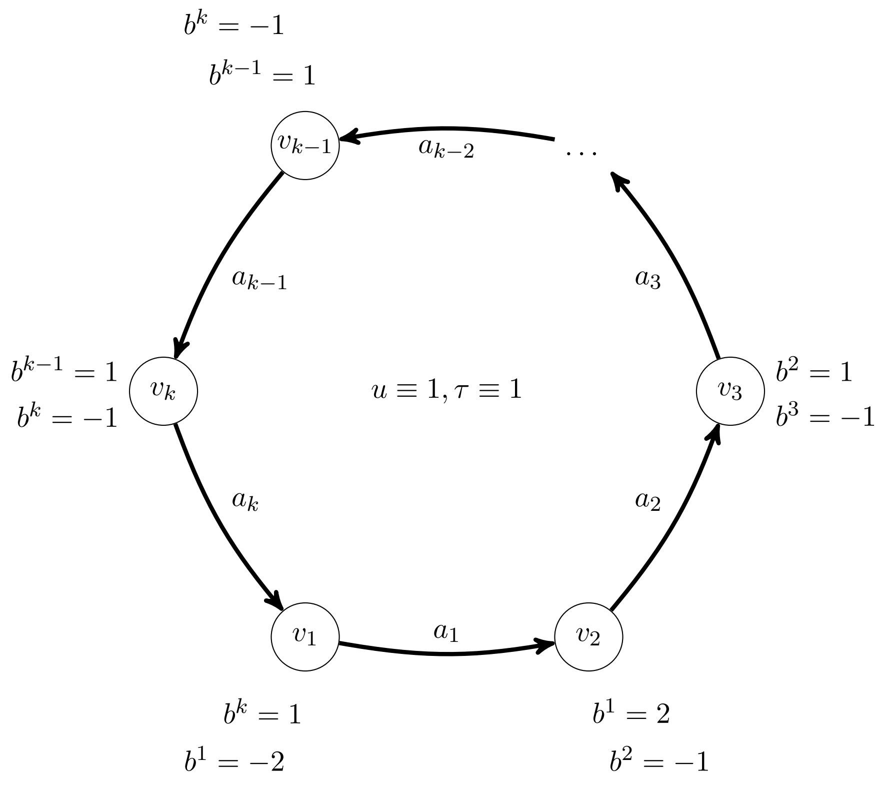


Figure 3.4: An instance $G_k, k \in \mathbb{N}$. All arcs have unit capacities and transit times, node supplies and demands are zero unless specified otherwise in the picture.

Proof. The following multi-commodity flow over time f satisfies all supplies and demands within time $k+1$.

- For commodity 1, we send flow with a rate of 1 into the path $v_2 \to \dots \to v_k \to v_1$ during the time interval $[0,2)$. This flow does not wait at any node and is completed at time $(k-1)+2 = k+1$.

- For commodities $i > 1$, we send flow with a rate of 1 into the path $v_{i \bmod k+1} \to \dots \to v_k \to v_1 \to \dots \to v_i$ during the time interval $[0,1)$. This flow waits at node v_2 for one time unit, if $i > 2$. These flows are completed by $(k-1)+1+1 = k+1$ time.

This flow satisfies supplies and demands as well as flow conservation and non-negativity constraints by construction. Thus, we only need to check the capacity constraints. By our choice of flow rates, capacity violations can only occur if two paths send flow into an arc at the same time.

We define $A(j,\theta)$ to be the set of arcs commodity j sends flow into at time $\theta \in [0,k)$.

By construction of the flow, we get

$$A(j,\theta) := \begin{cases} \{a_2\} & j=1, \theta<1, \\ \{a_{1+\lfloor\theta\rfloor}, a_{2+\lfloor\theta\rfloor}\} & j=1, 1\leq\theta<k-1, \\ \{a_k\} & j=1, k-1\leq\theta<k, \\ \{a_{((j+\lfloor\theta\rfloor) \bmod k)+1}\} & j>1, j+\lfloor\theta\rfloor\leq k, \\ \emptyset & j>1, j+\lfloor\theta\rfloor = k+1, \\ \{a_{(j+\lfloor\theta\rfloor) \bmod k}\} & j>1, j+\lfloor\theta\rfloor \geq k+2. \end{cases}$$

For $i,j \in \{2,\dots,k\}$, $i<j$, $a \in A(i,\theta)\cap A(j,\theta)$ would require

$$(i+\lfloor\theta\rfloor) \bmod k = (j+\lfloor\theta\rfloor) \bmod k \quad \Rightarrow \quad i \equiv j \bmod k$$

or

$$(i+\lfloor\theta\rfloor) \bmod k = (j+\lfloor\theta\rfloor) \bmod k + 1, j \geq i+2 \quad \Rightarrow \quad 1 \equiv j-i \bmod k$$

which is not possible. Thus, $A(i,\theta)\cap A(j,\theta)=\emptyset$ for $i<j$ and $i,j\in\{2,\dots,k\}$. Similarly, we can check that $A(1,\theta)\cap A(j,\theta)=\emptyset$ for $j>1$ and $\theta\in[0,k)$, which guarantees the satisfaction of the capacity constraints and completes the proof. □

In the next step, we show that without storage, a time horizon of at least $2k-2$ is required to send all supplies and demands.

Lemma 3.7. *A flow over time without storage that fulfills all supplies and demands in G_k, $k\geq 3$ requires a time horizon of at least $2k-2$.*

Proof. Assume that a flow over time without storage exists that fulfills all supplies and demands with a time horizon of $T<2k-2$. We define

$$U[0,k-1) := \sum_{a\in A(G_k)} \int_0^{k-1} u_a \, d\xi = k(k-1) = k^2-k$$

as the maximal amount of flow that can be sent into arcs of G_k in the time interval $[0,k-1)$. Now let us consider by how much U decreases, if we send flow from a commodity i into its source-sink path at time $\theta\in[0,k-1)$. If we send from into its source-sink path any later than that, it will not reach the sink before $2k-2$ and can therefore not help us to fulfill supplies and demands in time.

If we send flow of commodity i into its source-sink path at time $\theta\in[0,k-1)$ with rate x, we use x capacity on arc v_{i+1} at time θ, and we use x capacity of arc $v_{(i+1+j) \bmod k}$ at time $\theta+j$, $j\in\{1,\dots,k-2\}$. Also, we lose x capacity of arc $v_{(i+1+j) \bmod k}$ at time $\theta-j$, $j\in\{1,\dots,\lfloor\theta\rfloor\}$, as storage is not possible and $<k+1$ time is not enough to complete sending flow of any commodity. Thus, if we send flow of commodity i into its path starting at time $\theta\in[0,k-1)$ with rate x, we lose $(k-1)x$ capacity of $U[0,k-1)$.

We need to send $k+1$ flow units in total which in order to reach their sinks before time $2k-2$, must start their paths before time $k-1$. Each of these units must therefore lose $k-1$ capacity of U. However, $(k-1)(k+1)=k^2-1>k^2-k=U$. Thus, we do not have enough capacities to send all $k+1$ supplies to their sinks in time. □

3.2.5 Summary

Let us conclude this section with a summary of the problems we introduced and an overview of results known for them. The problems introduced are listed in Table 3.3. All of these problems have versions where storage is allowed and forbidden, but in order to prevent bloat, we do not list both versions of each problem there.

Problem	Objective	Comment
Multi-commodity Maximum Flow over Time	max $\|f\|$	Infinite supplies and demands
Multi-commodity Flow over Time	—	Multiple sources and sinks, finite supplies and demands
Multi-commodity Quickest Flow	min T	
Multi-commodity Maximum Concurrent Flow over Time	max $\|f\|_{\text{conc}}$	

Table 3.3: An overview of the flow over time problems with multiple commodities.

In terms of computational complexity, these problems turn out to be very similar. The only major difference is that for problems without storage, more sophisticated techniques are required to obtain an FPTAS. An overview of results can be found in Table 3.4. Pseudo-polynomial algorithms based on time expansion exist for all of these problems and have been omitted.

Problem	Complexity	Approximation
Multi-commodity Flow over Time with Storage	**weakly NP-hard** *Reduction from* PARTITION Hall, Hippler, Skutella [55]	Time-**FPTAS** *Time Condensation* Fleischer, Skutella [32] Chapter 5.1
Multi-commodity Flow over Time without Storage		Time-/Value-**FPTAS** *Sequence Condensation* Groß, Skutella [52] Chapter 5.3

Table 3.4: Results for multi-commodity flow over time problems. All results hold for the maximum and quickest versions of the problems as well.

3.3 Minimum Cost Flows over Time

In this chapter, we discuss minimum cost flow over time problems in their single and multi-commodity versions. These problem uses capacities, transit times and supplies and demands.

In the single commodity version, this problem is a generalization of the maximum flow over time problem studied by Ford and Fulkerson [38] that adds costs to the arcs. Analogously, the multi-commodity version is a generalization of the multi-commodity maximum flow over time problem. This is an important modelling feature, as sending flow is usually not free. Flow might for example correspond to goods that need to be shipped, which incurs costs for fuel or other kinds of transportation fees. Many of the examples listed in Chapter 3.2 have costs that would preferably be minimized. Some examples include aircraft routing [99], cloud computing [21] and power system management [36]. More applications where costs play an important factor can be found in Ahuja, Magnanti and Orlin [2], Aronson [4], Orlin [84] and Powell, Jaillet and Odoni [92]. This leads us to the *minimum cost flow over time* and *multi-commodity minimum cost flow over time* problems.

Minimum Cost Flow over Time Problem

Input: A directed graph G with capacities u, costs c, transit times τ, finite supplies and demands b of a single commodity and a time horizon T.

Output: A flow over time f in G fulfilling all supplies and demands of minimum cost $c(f)$.

Multi-commodity Minimum Cost Flow over Time Problem

Input: A directed graph G with capacities u, costs c, transit times τ, finite supplies and demands b of k commodities and a time horizon T.

Output: A flow over time f in G fulfilling all supplies and demands of minimum cost $c(f)$.

The most straightforward way to solve this problem is to use time expansion (Ford and Fulkerson [39], see Chapter 4). This transforms the problem into its static counterpart without transit times, the (multi-commodity) minimum cost flow problem, at the cost of a pseudo-polynomial blow-up in graph size. The static problems can be formulated as a linear programs, yielding pseudo-polynomial algorithms for the original problems. These linear programs can be solved in strongly polynomial time (Tardos [108]), and for the single commodity version, there are also strongly polynomial combinatorial algorithms (e. g., Orlin [85]). Other exact approaches, like the algorithm proposed by Aronson and Chen [5], are also only pseudo-polynomial.

Efficient algorithms were found for special cases of this problem, however. Orlin [84] studied a version of this problem with an infinite time horizon and gave a polynomial algorithm for this setting. For the general case, weak NP-hardness was shown by Klinz and Woeginger [71] by a reduction from EVEN-ODD-PARTITION (Garey and Johnson [43]). This reduction works even in the case of a single commodity and series-parallel graphs.

For this problem, we again make a case distinction whether storage is allowed or not. If storage is allowed, there are always optimal solutions that send flow along simple paths only and that do not require storage (Fleischer and Skutella [32]). In this setting, we can employ time condensation [32] to obtain an FPTAS, which is described in detail in Chapter 5.

However, if storage is forbidden, this approach fails to give a good approximation guarantee, because flow along cycles to simulate storage can become necessary for an optimal solution. In this case, we can employ the sequence condensation developed by Groß and Skutella [52], described in Chapter 5.3, which yields an FPTAS for this case as well.

Notice that both FPTASs work for single and multiple commodities. Together, these results give the picture of the complexity of minimum cost flow over time problems depicted in Table 3.5. Once again, pseudo-polynomial algorithms based on time expansion exist for all these problems and have been omitted from the table.

Storage	Complexity	Approximation
With storage	**weakly NP-hard** *Reduction from* EVEN-ODD-PARTITION Klinz, Woeginger [71]	Time-**FPTAS** *Time Condensation* Fleischer, Skutella [32] Chapter 5.1
Without storage		Time-**FPTAS** *Sequence Condensation* Groß, Skutella [52] Chapter 5.3

Table 3.5: An overview of results for minimum multi-commodity cost flow over time problems.

3.4 Generalized Flows over Time

In this chapter, we consider flow models that have gains in addition to capacities, transit times and supplies and demands. Gains can, for example, be used to model interest – e. g., we might put some amount of money into an investment and get three percent interest after a year, which could be modelled by an arc with a gain factor of 1.03 and a transit time of one year. Other applications include modelling leakage, evaporation, breeding or theft, for example. We will focus on the generalization of the maximum flow over time problem, which is the *generalized maximum flow over time problem*, but we can apply the same generalization to problems with finite supplies and demands.

Generalized Maximum Flow over Time Problem

Input: A directed graph G with capacities u, gains γ, transit times τ, supplies and demands b of a single commodity and a time horizon T.

Output: A flow over time f in G maximizing $|f|$.

Generalized flows (without transit times) have been suggested as a tool in production planning as early as 1939 by Kantorovich [68]. Interestingly, generalized flows over time have not been studied so far, even though there has been considerable research on the static generalized maximum flow problem. Since static generalized flow problems can be formulated as linear programs [23], they can be solved in polynomial time. The first combinatorial polynomial-time algorithm for computing static generalized maximum flows was proposed by Goldberg, Plotkin and Tardos [46] and has subsequently been improved by Radzik [94, 95]. Fleischer and Wayne [35], Goldfarb and Jin [47], Goldfarb, Jin and Orlin [48], Restrepo and Williamson [97] and Wayne [116, 117] described further polynomial-time algorithms.

Truemper [111] noted that generalized maximum flow algorithms show several analogies to minimum cost flow algorithms, if the negative logarithm of a gain factor is used as the cost. For example, there are successive highest-gain path algorithms for generalized flows and successive shortest paths algorithms for minimum cost flow problems which behave analogously, and such similarities also extend to cycle cancelling algorithms. Nonetheless, these analogies are limited – it is an open problem whether a strongly polynomial time algorithm for the generalized maximum flow problem exists, contrary to the minimum cost flow problem.

For this problem, we distinguish three cases depending on the given gains. Aside from arbitrary gains, we will study *lossy* gains, where $\gamma_a \leq 1$ for all arcs $a \in A(G)$ holds. For such gains, there are no flow-generating cycles, which allows us to find optimal solutions that send flow only along paths. Furthermore, we will study gains where every path with same transit time has also the same gain – in this case, we call transit times and gains *proportional* to each other. Since transit times are additive and gain factors multiplicative along paths, proportional means that there exists a $c \in \mathbb{R}$ such that $\gamma_a = 2^{c \cdot \tau_a}$ for every arc $a \in A(G)$. Such gain factors are motivated by the fact that in many applications effects such as leakage, evaporation or interest rates are strictly time-bound. Also many processes of growth or decay in nature can be captured by such proportional gain factors.

For arbitrary gain factors, Groß and Skutella [51] show that the generalized maximum flow over time problem is NP-hard – there is even no polynomial approximation algorithm that leaves the time horizon intact, unless P = NP.

For the case of lossy networks, we show that the concept of condensed time-expanded networks by Fleischer and Skutella [32] can be extended to generalized flows over time (Chapter 5.1). This gives an FPTAS that approximates the time horizon. The NP-hardness proof of [51] applies to this setting as well.

For the case of proportional losses, i. e., gains which are both lossy and proportional, Groß and Skutella [51] gave an FPTAS which only approximates the flow value and not the time horizon – this is very uncommon for flow over time problems, because usually, the temporal dimension is responsible for the difficulty and must be dealt with. Another peculiarity of this FPTAS is that the running time depends only logarithmically on the error, whereas normally a polynomial dependency would be expected. This FPTAS does also not require storage. Furthermore, Groß and Skutella show that generalized maximum flows over time are also always earliest arrival flows in this case. More on earliest arrival flows can be found in the next section.

Gains	Complexity	Approximation
Arbitrary	**weakly NP-hard** *Reduction from* PARTITION Groß, Skutella [51] Chapter 7	**?**
Lossy		Time-**FPTAS** *Time Condensation* Groß, Skutella [51] Chapter 5
Lossy & Proportional	**Earliest Arrival Flow** Groß, Skutella [51] Chapter 7	Value-**FPTAS** *Successive Shortest Path* Groß, Skutella [51] Chapter 6

Table 3.6: An overview of results for generalized flows over time.

3.5 Earliest Arrival Flows

In this section, we will study flows which have multiple objectives. More precisely, we will study flows which attempt to maximize the amount of flow sent at multiple points in time. Compared to that, maximum flows over time maximize the amount of flow sent only at a single time – the time horizon. Flows which attempt to maximize the amount of flow at multiple times are called *multiple deadline flows* (Stiller and Wiese [106]). A special case of multiple deadline flows are flows which are maximal at all points in time – such flows are called *universal maximal flows* or *earliest arrival flows*. Even though they are a special case of multiple deadline flows, earliest arrival flows have a much longer history – they were introduced by Gale [41] in 1959. It is not clear that they always exist, but Gale managed to prove that for graphs with a single sink, they do always exist.

For this type of flow, it matters if we are using the discrete or continuous time model, as in the first case, we have to be optimal at times $0, 1, \dots, T-1$ whereas in the second case, we have to be optimal at all $\theta \in [0, T)$. In general, we will use the discrete time model, which was also the setting which Gale studied. However, we will give some results for the continuous time model as well.

Maximizing the amount of flow sent at all points in time is very desirable in evacuation planning. When preparing for an evacuation, it is usually unclear how much time there will actually be to enact the plans. Thus, it would be preferential, if we had a plan that was optimal independent of the time available to execute it – and this is exactly what earliest arrival flows offer. Since the existence of earliest arrival flows was at first only guaranteed for a single sink, the computation of earliest arrival flows was at first restricted to graphs with a single sink.

The resulting problem is called the *earliest arrival flow problem* or the *earliest arrival transshipment problem.* The former is usually used for instances with a single source and sink, whereas the latter is used for instances with multiple sources and single sink. We will not use that distinction, however, as we will also deal with earliest arrival flow problems with multiple sources and sinks, for which no term has been established yet. We will refer to all such problems as earliest arrival flow problems and specify the number of sources and sinks that they have explicitly instead.

Earliest Arrival Flow Problem

Input:	A directed graph G with capacities u, transit times τ, finite or infinite supplies and demands b of a single commodity and a time horizon T.
Output:	An earliest arrival flow over time f in G.

Notice that the time horizon is sometimes omitted from the input if supplies and demands are finite. In this case the time horizon is implicitly given by the time horizon of a quickest flow for the input.

Gale's proof for the existence of earliest arrival flows was non-constructive. The first algorithms for computing earliest arrival flows were found by Minieka [78] and Wilkinson [118]. Their algorithms are based on time expansion, and are therefore not efficient but only pseudo-polynomial. Finding an earliest arrival flow is complicated by the fact that the earliest arrival pattern, which specifies the amount of flow an earliest arrival flow sends at a given time, is a piecewise linear function with potentially exponentially many breakpoints, as follows from the work of Zadeh [122].

For the continuous time model, Philpott [91] showed the existence of earliest arrival flows in graphs with one sink and Fleischer and Tardos [33] developed an algorithm for them. An algorithm for earliest arrival flows that works in both time models and graphs with multiple sources and a single sink was given by Baumann and Skutella [11]. Their algorithm is based on an algorithm for the quickest transshipment problem given by Hoppe [67], and its running time is polynomial in input size and the size of the earliest arrival pattern.

The complexity of the earliest arrival flow problem was open for a long time, but very recently, Disser and Skutella [25] managed to prove that computing an earliest arrival flow is NP-hard. However, for special classes of graphs, efficient algorithms can be found. For example, for series-parallel graphs, an efficient algorithm was described by Ruzika, Sperber and Steiner [100].

For arbitrary graphs, we have to settle for approximation algorithms, if we want our algorithms to be efficient. Since earliest arrival flows maximize multiple objectives at once, approximation algorithms for them usually approximate all objectives at once as well. As before, we have two ways for approximation – either by time or by value.

Definition 3.8. *An* α-time-approximate earliest arrival flow *is a flow over time f that sends at every point in time $\theta \in \{0, \dots, T-1\}$ at least as much flow value as possible* *at time θ/α, i. e. $|f|_\theta \geq p\left(\lfloor \frac{\theta}{\alpha} \rfloor\right)$.*

Definition 3.9. *A β-*value-approximate earliest arrival flow *is a flow over time f that sends at every point in time $\theta \in \{0, \ldots, T-1\}$ at least a β-fraction of the maximum flow value at time θ, i.e. $|f|_\theta \geq \frac{p(\theta)}{\beta}$.*

α-time-approximate earliest arrival flows were defined by Baumann and Köhler [10] in the context of flows over time with flow-dependent transit times. In this transit time model, the transit time of an arc depends on the flow, as the name suggests. In this setting, earliest arrival flows do usually not exist as well. Baumann and Köhler gave an approximation algorithm which guarantees for each point in time θ that the amount of flow that has reached the sink until θ is at least equal to the maximum amount of flow that can be sent to the sink until time $\theta/4$. β-value-approximate were defined by Groß, Kappmeier, Schmidt and Schmidt [50] for earliest arrival flows in networks with multiple sinks. Both concepts were used before, though, even if they were not explicitly named. In fact, Hoppe and Tardos [66] gave a value-approximative FPTAS for the earliest arrival flow problem and Fleischer and Skutella [32] described a time-approximative FPTAS for earliest arrival flows.

In settings with multiple sinks, earliest arrival flows do not necessarily exist [11]. Here, the case of zero transit times is relatively well understood – in networks with a single source and multiple sinks, they do exist [101], but this does not hold for networks with both multiple sources and sinks (Fleischer [34]). For this setting, Schmidt and Skutella [101] gave a characterization of the class of all graphs with zero transit times that always allow for earliest arrival flows regardless of their capacities and supplies and demands.

For graphs with multiple sinks and arbitrary transit times, it is still possible to find approximative earliest arrival flows. Groß, Kappmeier, Schmidt and Schmidt [50] gave an algorithm that computes a 2-value-approximate earliest arrival flow in an arbitrary network, and complement it with a family of instances where no value-approximation better than 2 is possible. This lower bound holds even for zero transit times. Thus, $\beta = 2$ is the best possible value-approximation factor. The algorithm is pseudo-polynomial in the general case, as it uses time expansion, but it is efficient for the special case of networks with zero transit times. We will study this algorithm in Chapter 4.2.

Time-approximation on the other hand turns out to be almost impossible for networks with multiple sources and sinks, although Groß, Kappmeier, Schmidt and Schmidt [50] extended an approximation technique by Baumann and Köhler [10] to give a 4-time-approximation for the case of a single source and multiple sinks. The lower bounds are not tight for this case, however, as the best known lower bound here is 2 [50]. Compared to this, the case with multiple sources and sinks has a lower bound that is linear in the time horizon. Table 3.7 shows an overview of existence results for approximate earliest arrival flows with arbitrary transit times – for zero transit times, the only difference is that earliest arrival flows exist if there is a single source.

For evacuation scenarios, like for evacuating a specific ship or building, a constructive algorithm that computes a 2-approximate earliest arrival flow is certainly more useful

Approximation	1 Source 1 Sink	2+ Sources 1 Sink	1 Source 2+ Sinks	2+ Sources 2+ Sinks
α-Time	$\mathbf{1}$		$\mathbf{[2,4]}$	$\mathbf{\Omega(T)}$
	Gale [41] Baumann, Skutella [11]		Groß et al. [50]	
β-Value	$\mathbf{1}$		$\mathbf{2}$	
	Gale [41] Baumann, Skutella [11]		Groß et al. [50] Chapter 4.2	

Table 3.7: An overview of existence results for approximate earliest arrival flows with arbitrary transit times.

than a non-existence result. However, even more desireable would be to have an earliest arrival flow approximation that is the best possible approximation for the given instance.

Groß, Kappmeier, Schmidt and Schmidt adopted the concept of geometrically condensed time-expanded networks from Fleischer and Skutella [32] in order to give an FPTAS for computing the best possible approximation factor for a given instance. For an arbitrary $\varepsilon > 0$, it incurs additional $(1 + \varepsilon)$-factors for both time and value. This FPTAS is presented in Chapter 5.2.

3.6 Orientations and Flows over Time

We conclude this chapter with a discussion of the problem of orienting an undirected graph such that the orientation is as beneficial as possible for a subsequent flow over time computation. We are going to assume that the orientation is done once, before the flow over time is sent, and cannot be changed afterwards. This is motivated by evacuation settings where we might not have the resources to change direction of lanes during an ongoing evacuation. There are surely cases where changing direction can be done more often, both in evacuation scenarios and in everyday life – consider for example the reversal of lanes to speed up traffic into a city in the morning and out of it in the afternoon or evening. However, since little research has been done on the connection of orientations and flows over time, we focus on one model first, which is the model of a single orientation in the beginning. This model is probably one of the easiest to realize in practice, which is why we study it first. Other models we will have to leave to further research.

We will assume that we have to orient each edge completely into one direction and splitting the capacity is not possible. This is no significant restriction, as we could still model lanes of a road as individual edges, if we want to orient the lanes individually.

We have chosen a relatively simply and straightforward model, so it would be interesting to know how much potential is lost by this simplicity. Thus, we need something to compare our models to. For this purpose, we compare flows over time in an oriented network with flows over time in the original undirected network. As we mentioned in the introduction, the flow over time in the undirected graph can be interpreted as a flow with an optimal utilization of the edge, like what an artificial intelligence controlling all vehicles at once might achieve.

More formally, we introduce two *prices of orientation* that are based on the comparison of what is possible with the best orientation in our model compared to what is possible in the undirected network. The first price is based on the flow value and is essentially the quotient of the maximal flow values in both scenarios. Formally, it is defined as follows.

Definition 3.10. *The* flow price of orientation *for an undirected network over time* $N = (G, u, b, \tau, T)$ *is the ratio between the value of a maximum flow over time* f_N *in* N *and maximum of the values of maximum flows over time* $f_{\overrightarrow{N}}$ *in orientations* $\overrightarrow{N}$ *of* N*:*

$$\frac{|f_N|}{\max_{\overrightarrow{N} \text{ orientation of} N} |f_{\overrightarrow{N}}|} \ .$$

Analogously, the second price is based on the time horizon and is the quotient between the time horizons necessary to send everything in both scenarios. It is defined as follows.

Definition 3.11. *The* time price of orientation *for an undirected network over time* $N = (G, u, b, \tau)$ *is the ratio between the minimal time horizon* $T(f_{\overrightarrow{N}})$ *of a quickest flow* $f_{\overrightarrow{N}}$ *in an orientation* $\overrightarrow{N}$ *of* N *and the time horizon* $T(f_N)$ *of a quickest flow over time* f_N *in* N*:*

$$\frac{\min_{\overrightarrow{N} \text{ orientation of} N} T(f_{\overrightarrow{N}})}{T(f_N)} \ .$$

To our knowledge, these prices of orientation have not been studied before for flows over time. However, there has been a lot of research on orienting graphs, and orienting graphs in conjunction with static flows. Robbins [98] studied the problem of orienting streets already in 1939. His research was motivated by the problem of controlling the congestion of city streets during a weekend, in particular by making streets one-way only during the weekend. He showed that a undirected graph has to be 2-edge connected, if it can be oriented into a strongly connected directed graph, and vice versa. This result was later generalized by Nash-Williams [80]. An example for orientations in conjunction with static flows is the minimum cost strong network orientation problem (Burkard, Feldbacher, Klinz and Woeginger [19]), which finds application in automated guided vehicle systems (see also [44, 115]).

The problem of changing or prescribing the direction of lanes or streets is also well studied, as it is an important strategy to mitigate congestion during an emergency situation or at rush hours. This problem is then called a *contraflow*, *reversible flow* or *lane reversal* problem. Contraflows are an important tool for hurricane evacuation [120], and in that context the importance of modeling time has become prevalent in the past decade [121]. It is also employed to handle traffic during rush hours [61].

Here, we want to combine the orientation of a network with flows over time – more specifically, we want to find an orientation of the network that is as beneficial as possible for the flow over time problem. We begin by combining orientations with the maximum flow over time problem. This leads us to the *maximum contraflow over time problem.*

Maximum Contraflow over Time Problem

Input:	A undirected network $N = (G, u, \tau, b, T)$, consisting of a graph G with capacities u, transit times τ, supplies and demands b of a single commodity and a time horizon T.
Output:	An orientation $\overrightarrow{N}$ of N such that the value of a maximum flow over time in $\overrightarrow{N}$ is maximal over all possible orientations of N.

Similarly, we define a contraflow version of the quickest flow problem, which becomes the *quickest contraflow problem.*

Quickest Contraflow Problem

Input:	A undirected network $N = (G, u, \tau, b)$, consisting of a graph G with capacities u, transit times τ and finite supplies and demands b of a single commodity.
Output:	An orientation $\overrightarrow{N}$ of N such that the time horizon of a quickest flow in $\overrightarrow{N}$ is minimal over all possible orientations of N.

The same kind of generalization can be done for the multi-commodity versions of these problems, resulting in the *maximum multi-commodity flow over time* and the *quickest multi-commodity contraflow* problems. Since we have two types of flow values in settings with multiple commodities, the total flow value and the concurrent flow value, we also get the *maximum concurrent multi-commodity contraflow over time problem.*

Also, we can define a contraflow version of the earliest arrival flow problem. For a given network $N = (G, u, \tau, b)$, this *earliest arrival contraflow problem* asks for an orientation $\overrightarrow{N}$ of N and a flow over time f in $\overrightarrow{N}$, such that $|f|_\theta = p(\theta)$ holds for all points in time θ, if p is the earliest arrival pattern of the undirected network. This means that we have to find an orientation and a flow over time that is maximal for any point in time over all possible orientations. Since such a flow over time and orientation will usually not exist, we will have to settle for α-time- or β-value-approximate earliest arrival flows. For these approximations, the $|f|_\theta = p(\theta)$ constraint is replaced by $|f|_\theta \geq p\left(\left\lfloor \frac{\theta}{\alpha} \right\rfloor\right)$ and $|f|_\theta \geq \frac{p(\theta)}{\beta}$, respectively, for all points in time θ.

For the case of a single source and sink, it follows from the work of Ford and Fulkerson [39] that both the time and the value price of orientation is 1. This is because their algorithm for the maximum flow over time problem produces a temporally repeated solution out of a static minimum cost flow. This temporally repeated solution uses each edge in only one direction, and can therefore be used to obtain the desired orientation. For problems with multiple sources or sinks, this does not hold, however. Consider the example depicted in Figure 3.5.

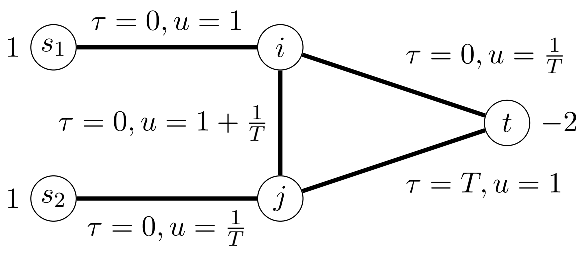

1
s_1
$\tau = 0, u = 1$
i
$\tau = 0, u = \frac{1}{T}$
$\tau = 0, u = 1 + \frac{1}{T}$
t
-2
1
s_2
j
$\tau = T, u = 1$
$\tau = 0, u = \frac{1}{T}$

Figure 3.5: An instance where an optimal quickest flow solution has to use edge (i, j) in both directions.

In the undirected graph, we can fulfill the supplies and demands in time $T + 1$ by sending flow along $s_1 \to i \to j \to t$ and $s_2 \to j \to i \to t$. If we have to orient $\{i, j\}$, we have two possibilities. Either, we orient it as (i, j) and send the supply of s_2 via j to t, which takes $2T$ time. Or, we orient it as (j, i), then we have to send the supply of s_1 via i to t, which blocks $\{i, t\}$ for T time units. However, since we cannot send more than $\frac{1}{T}$ units of flow from s_2 via j to t in time $T + 1$, we have to send $1 - \frac{1}{T}$ flow units of s_2 through $\{i, j\}$, which is not possible in time $T + 1$.

It was also shown that finding an optimal orientation for a quickest contraflow problem with multiple sources and sinks is NP-hard (see Kim and Shehkar [69], Rebennack, Arulselvan, Elefteriadou and Pardalos [96]). Because of the complexity of the problem, heuristic and simulation tools are predominantly used in praxis, e. g., [69, 112, 113, 120].

We extend these results by showing tight bounds for the flow price of orientation, which turns out to be 2 if we have either multiple sources or multiple sinks, and 3 if we have both. We give an algorithm to compute an optimal orientation, which reduces the problem to the single source and sink case, by simulating supplies and demands through capacities of auxiliary arcs. This simulation is not precise, however, and we show how to obtain good capacities to obtain the 2 and 3 bounds by using an iterative approach whose convergence is guaranteed by Brouwer fixed-points. Furthermore, we describe instances for which the bounds of 2 and 3 are tight, respectively. For the time price of orientation, we give instances where the price is linear in the number of nodes.

Since we have two ways to pay the price of orientation – flow value and time horizon – the question arises whether it might be desirable to pay the price partly as flow value and partly as time horizon. We prove that by doing so, we can achieve a bicriteria-price of 2/2 for the case of multiple sources and sinks, compared to 3 and $\Omega(n)$ if we would pay as flow value or time horizon alone. This price means that we can send at least half the flow value in twice the amount of time. Table 3.8 shows an overview of our results.

Notice that even for simple graphs like trees, we cannot hope for a price of orientation of 1, as we will see in Chapter 8. In practice, orienting road networks is an important aspect of evacuation management. In terms of evacuations, earliest arrival flows (or approximations thereof) are very desirable, as they provide optimal routings independent of the time that is available. However, we are able to show that earliest arrival flows

Price of Orientation	1 Source 1 Sink	1 Source 2+ Sinks	2+ Sources 1 Sink	2+ Sources 2+ Sinks
Value	**1** Ford, Fulkerson [39]	**2**	**2** Arulselvan, Groß, Skutella [7] Chapter 8	**3**
Time	**1** Ford, Fulkerson [39]		**$\Omega(n)$** Arulselvan, Groß, Skutella [7] Chapter 8	
Time & Value	**1** Ford, Fulkerson [39]		**2/2** Arulselvan, Groß, Skutella [7] Chapter 8	

Table 3.8: An overview of price of orientation results.

and the approximations developed by Baumann and Köhler [10] and Groß, Kappmeier, Schmidt and Schmidt [50] do not exist in this setting.

In terms of computational complexity, it turns out that the maximum contraflow over time problem cannot be approximated by value within a factor of 2, unless $\mathsf{P} = \mathsf{NP}$. Similarly, the quickest contraflow problem cannot be approximated by time within a factor of 2, unless $\mathsf{P} = \mathsf{NP}$. For the multi-commodity flow problems, there are even stronger results – the quickest multi-commodity contraflow problem cannot be approximated in time better than within a factor of $\Theta(\sqrt{U})$, where U is the largest capacity in the input, and the maximum concurrent multi-commodity contraflow problem cannot be approximated at all, unless $\mathsf{P} = \mathsf{NP}$. These complexity results are all due to Arulselvan, Groß and Skutella [7] and can be found in Chapter 8.2.

4 Time Expansion

In this chapter we focus on techniques which are based on time expansion. All these techniques have in common that they transform a dynamic problem with transit times and a time horizon into a static problem without transit times and time horizon. This problem can then be solved by network flow algorithms that normally cannot handle transit times. However, these reductions can cause an exponential blow-up in network size, but we will see how we can handle the blow-up in the next chapter.

This chapter is split into two parts. In the first part, we give an introduction into time expansion by defining time-expanded networks and proving that they allow us to compute flows over time as static flows. This result is also the reason that we can usually switch between the continuous and discrete time models.

In the second part, we show an iterative time expansion that adds time layers sequentially instead of all time layers at once to value-approximate earliest arrival flows. This result was published in Groß, Kappmeier, Schmidt and Schmidt [50].

4.1 Introduction

The concept of time expansion goes back to Ford and Fulkerson [38, 39]. We assume now that we have integral transit times and an integral time horizon, but we will discuss how to handle the non-integral case later. The idea of time expansion is to create a copy of the nodes of the network for every integral point in time from 0 to the time horizon. The node copies represent a node at a specific point in time (or in a time interval of length 1, depending on the preferred view). These copies are then linked with arcs according to the transit times of the arcs. The resulting network is called a *time-expanded network*. Depending on the problems they are used for there are several variants which depend on the attributes used (e.g., capacities, costs, gains, etc.) and whether storage of flow in nodes is permitted or not. We give general definitions covering most variants, beginning with the case that storage of flow is forbidden. In this case, we still need to model that flow can be stored in sources and sinks. This is done by *holdover arcs*. These arcs link two copies of a node in adjacent time layers. Thus, flow travelling through a holdover arc can be interpreted as flow staying in a node for one time step.

4.1.1 Definitions

For this chapter, we assume that sources have no incoming edges and sinks have no outgoing edges. This can be done without loss of generality by introducing dummy nodes (see Chapter 2.2.1), and this makes handling the holdover arcs much simpler. For the sake of convenience, we assume $-\infty, +\infty \in \mathbb{R}$.

Definition 4.1 (Time-expanded network without storage). *Let G be a directed graph with integral transit times $\tau : A(G) \to \mathbb{N}_0$ and an integral time horizon $T \in \mathbb{N}_0$. The* time-expanded network without storage G^T *of G for the time horizon T is constructed by defining*

$$\begin{aligned} V(G^T) &:= \{ v_\theta \mid v \in V(G), \theta \in \{0, 1, \ldots, T-1\} \}, \\ H(G^T) &:= \{ (v_\theta, v_{\theta+1}) \mid v \in V(G) : \exists i \in K : b^i_v \neq 0, \theta \in \{0, \ldots, T-2\} \}, \\ A(G^T) &:= H(G^T) \cup \{ a_\theta = (v_\theta, w_{\theta+\tau_a}) \mid a = (v,w) \in A(G), \theta \in \{0, \ldots, T - \tau_a - 1\} \}. \end{aligned}$$

The arcs in $H(G^T)$ are called holdover arcs. *Capacities $u : A(G) \to \mathbb{R}^+_0$ for G are extended to capacities $u^T : A(G^T) \to \mathbb{R}^+_0$ for G^T by giving each copy of an arc the capacity of the original arc and holdover arcs infinite capacity, i. e.,*

$$u^T_{a'} := \begin{cases} u_a & a' = a_\theta \in A(G^T) \setminus H(G^T), \\ \infty & a' \in H(G^T). \end{cases}$$

Similarly, we can extend costs $c : A(G) \to \mathbb{R}$ and gains $\gamma : A(G) \to \mathbb{R}^+_0$ from G to G^T by giving holdover arcs zero costs and unit gains, and giving copied arcs the costs and gains of the original arc:

$$c^T : A(G^T) \to \mathbb{R}, c^T_{a'} := \begin{cases} c_a & a' = a_\theta \in A(G^T), \\ 0 & a' \in H(G^T), \end{cases}$$

$$\gamma^T : A(G^T) \to \mathbb{R}^+_0, \gamma^T_{a'} := \begin{cases} \gamma_a & a' = a_\theta \in A(G^T), \\ 1 & a' \in H(G^T). \end{cases}$$

Let $K = \{1, \ldots, k\}$ be the set of commodities. Balances $b : V(G) \times K \to \mathbb{R}$ are extended from G to balances $b^T : V(G^T) \times K \to \mathbb{R}$ for G^T by transferring supplies to the node copies in the first time layer and demands to the node copies in the last time layer. Or more formally,

$$\left(b^T\right)^i_{v_\theta} := \begin{cases} b^i_v & \theta = 0, b^i_v > 0, \\ b^i_v & \theta = T-1, b^i_v < 0, \\ 0 & \textit{else}. \end{cases}$$

Notice that by our assumption, holdover arcs cannot be used for storage in intermediate nodes. An illustration of this construction for a graph G with transit times, capacities, single-commodity balances and a time horizon of 4 is shown in Figure 4.1. If storage in intermediate nodes is allowed, the time-expanded network gains additional holdover arcs for all intermediate nodes to model this.

Definition 4.2 (Time-expanded network with storage). *Let G be a directed graph with integral transit times $\tau : A(G) \to \mathbb{N}_0$ and an integral time horizon $T \in \mathbb{N}_0$. The* time-expanded network with storage G^T_{st} *of G for the time horizon T differs from G^T only in the additional holdover arcs, i.e., we replace $H(G^T)$ in Definition 4.1 by $H(G^T_{st})$, which is defined as*

$$H(G^T_{st}) := \{ (v_\theta, v_{\theta+1}) \mid v \in V(G), \theta \in \{0, \ldots, T-2\} \}.$$

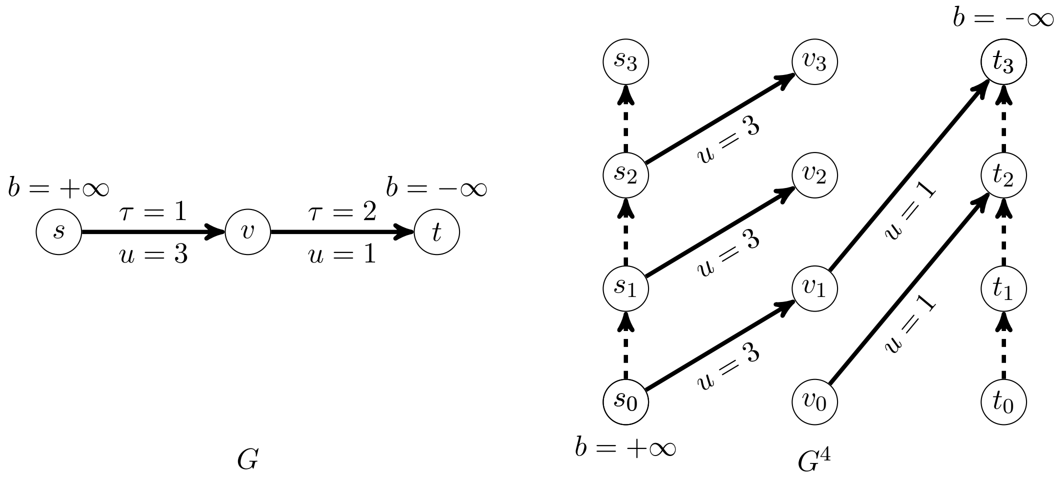


Figure 4.1: A network G and its time-expanded network without storage G^4. The holdover arcs are dashed. Balances that are not specified are 0.

Figure 4.2 shows the construction with storage at intermediate nodes for a graph G with transit times, capacities, single-commodity balances and a time horizon of 4. For notational brevity, we will omit the $_{st}$-subscript in the following if it is clear whether we allow storage at intermediate nodes or not.

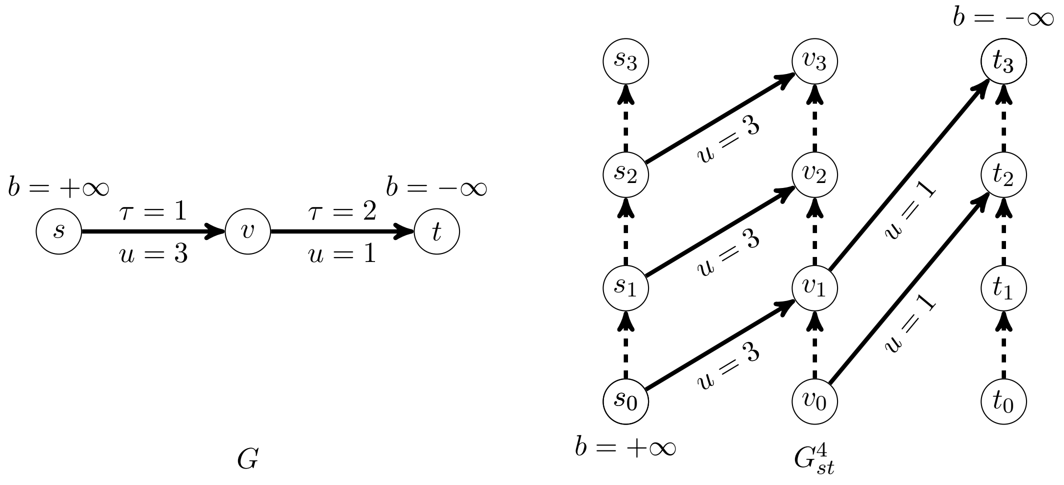


Figure 4.2: A network G and its time-expanded network with storage G^4_{st}. The holdover arcs are dashed. Balances that are not specified are 0.

4.1.2 Flows over Time and Static Flows in the Time Expansion

Now we need to link flows over time in the original graph G with static flows in the time-expanded graph G^T or G^T_{st}, respectively, which is done by following theorem, which is also based on the results of Ford and Fulkerson [38, 39]. Essentially, we can find an equivalent flow over time without storage in G for a static flow in G^T, and vice versa. Analogously, we can find an equivalent flow over time with storage in G for a static flow in G^T_{st}, and reversely. For the proof, we begin with the latter case and follow Skutella [104], where a proof sketch for a version of the theorem with storage, but without multiple commodities and gains is shown.

Theorem 4.3. *Let G be a directed graph with integral transit times $\tau : A(G) \to \mathbb{N}_0$, capacities $u : A(G) \to \mathbb{R}^+_0$, costs $c : A(G) \to \mathbb{R}$, gains $\gamma : A(G) \to \mathbb{R}^+_0$, commodities $K = \{1, \dots, k\}$, balances $b : V(G) \times K \to \mathbb{R}$ and an integral time horizon $T \in \mathbb{N}_0$, with the corresponding time-expanded network with storage G^T_{st} and extended capacities $u^T : A(G^T_{st}) \to \mathbb{R}^+_0$, costs $c^T : A(G^T_{st}) \to \mathbb{R}$, gains $\gamma^T : A(G^T_{st}) \to \mathbb{R}^+_0$, balances $b^T : V(G^T_{st}) \times K \to \mathbb{R}$. For any multi-commodity flow over time f in G there exists a static multi-commodity flow x in G^T_{st} that sends the same amount of flow for every commodity with the same cost, i.e., $|f^i| = |x^i|$ for all commodities $i \in K$ and $c(f) = c(x)$. Analogously, there exists a multi-commodity flow over time f in G for any static multi-commodity flow x in G^T_{st} with $|f^i| = |x^i|$ for all commodities $i \in K$ and $c(f) = c(x)$.*

Proof. We will show that given a flow x in G^T_{st} or flow over time f in G, we can construct an equivalent flow over time or flow, respectively. We begin by constructing a flow x in G^T_{st} given a flow over time f in G. The idea for that is to have each arc in the time-expanded network represent an arc in the original network during a time interval of length 1.

Constructing x from f. Given f, we define x by assigning the amount of flow f sends from a commodity i into an arc a in an time interval $[\theta, \theta + 1)$ to $x^i_{a_\theta}$:

$$x^i_{a_\theta} := \int_\theta^{\theta+1} f^i_a(\xi)\ d\xi \quad \text{for all } a_\theta \in A(G^T_{st}) \setminus H(G^T_{st}).$$

Then we set the flow values of the holdover arcs so that flow conservation is fulfilled for all nodes. For every commodity $i \in K$, we begin with the nodes $v \in V(G) : b^i_v \leq 0$ that do not have supply and set for all time points $\theta \in \{0, \dots, T-2\}$:

$$x^i_{(v_\theta, v_{\theta+1})} := \sum_{a \in \delta^-_G(v)} \int_0^{\theta+1-\tau_a} \gamma_a f^i_a(\xi)\ d\xi - \sum_{a \in \delta^+_G(v)} \int_0^{\theta+1} f^i_a(\xi)\ d\xi.$$

For sources, we determine the flow value of the holdover arcs based on how much is sent from later copies of the source. Thus, we set for all commodities $i \in K$ and nodes $v \in V(G) : b^i_v > 0, \theta \in \{0, \dots, T-2\}$:

$$x^i_{(v_\theta, v_{\theta+1})} := \sum_{a \in \delta^+_G(v)} \int_{\theta+1}^T f^i_a(\xi)\ d\xi.$$

Non-negativity and capacity constraints of x. Firstly, we need to check that x is indeed a flow in G_{st}^T, i.e., that is satisfies non-negativity, capacity and flow conservation constraints. Non-negativity of x is guaranteed, since f is non-negative and satisfies flow conservation. Holdover arcs have infinite capacities, so capacity constraints are fulfilled for them by default. They also hold for other arcs $a_\theta \in A(G_{st}^T) \setminus H(G_{st}^T)$ due to:

$$\sum_{i\in K} x_{a_\theta}^i = \sum_{i\in K} \int_\theta^{\theta+1} f_a^i(\xi)\; d\xi = \int_\theta^{\theta+1} \sum_{i\in K} f_a^i(\xi)\; d\xi \le \int_\theta^{\theta+1} u_a \; d\xi = u_a = u_{a_\theta}^T.$$

Flow conservation of x. That leaves us to verify flow conservation. For notational simplicity, we assume that $f_a(\theta) = 0$ for all arcs $a \in A(G)$ and $\theta \notin [0, T)$, and that $x_a = 0$, if a refers to a non-existing arc like (v_{-1}, w_0). We make a case distinction between nodes with supply and those without – due to the differences in the definitions of flow for their holdover arcs – and begin with the latter.

For a commodity $i \in K$, a node $v \in V(G)$ with $b_v^i \le 0$ and a time $\theta \in \{1, \dots, T-2\}$, flow conservation holds because of

$$\begin{aligned}
&\sum_{a\in\delta^-_{G_{st}^T}(v_\theta)} \gamma_a x_a^i = x^i_{(v_{\theta-1},v_\theta)} + \sum_{a\in\delta^-_G(v)} \gamma_a x^i_{a_{\theta-\tau_a}}\\
&= \sum_{a\in\delta^-_G(v)} \int_0^{\theta-\tau_a} \gamma_a f_a^i(\xi)\; d\xi - \sum_{a\in\delta^+_G(v)} \int_0^{\theta} f_a^i(\xi)\; d\xi + \sum_{a\in\delta^-_G(v)} \gamma_a \int_{\theta-\tau_a}^{\theta-\tau_a+1} f_a^i(\xi)\; d\xi\\
&= \sum_{a\in\delta^-_G(v)} \int_0^{\theta+1-\tau_a} \gamma_a f_a^i(\xi)\; d\xi - \sum_{a\in\delta^+_G(v)} \int_0^{\theta+1} f_a^i(\xi)\; d\xi + \sum_{a\in\delta^+_G(v)} \int_\theta^{\theta+1} f_a^i(\xi)\; d\xi\\
&\overset{(1)}{=} x^i_{(v_\theta,v_{\theta+1})} + \sum_{a\in\delta^+_G(v)} x^i_{a_\theta} = \sum_{a\in\delta^+_{G_{st}^T}(v_\theta)} x_a^i.
\end{aligned}$$

Analogously, we can show flow conservation for $\theta = 0$ by omitting the holdover arc (v_{-1}, v_0) for a node $v_0 \in G_{st}^T$:

$$\begin{aligned}
\sum_{a\in\delta^-_{G_{st}^T}(v_0)} \gamma_a x_a^i &= \sum_{\substack{a\in\delta^-_G(v),\\ \tau_a=0}} \gamma_a x^i_{a_0}\\
&= \sum_{\substack{a\in\delta^-_G(v),\\ \tau_a=0}} \gamma_a \int_0^1 f_a^i(\xi)\; d\xi - \sum_{a\in\delta^+_G(v)} \int_0^1 f_a^i(\xi)\; d\xi + \sum_{a\in\delta^+_G(v)} \int_0^1 f_a^i(\xi)\; d\xi\\
&= x^i_{(v_0,v_1)} + \sum_{a\in\delta^+_G(v)} x^i_{a_0} = \sum_{a\in\delta^+_{G_{st}^T}(v_0)} x_a^i.
\end{aligned}$$

Notice that for $\theta \in \{0, \dots, T-2\}$, we had $b_{v_\theta} = 0$ for all $v \in V(G)$, which does not hold for $\theta = T-1$ – this changes the equality (1) into the following inequality, which follows from the flow conservation of f:

$$\sum_{a\in\delta_G^-(v)} \int_0^{T-\tau_a} \gamma_a f_a^i(\xi)\, d\xi - \sum_{a\in\delta_G^+(v)} \int_0^{T} f_a^i(\xi)\, d\xi + \sum_{a\in\delta_G^+(v)} \int_{T-1}^{T} f_a^i(\xi)\, d\xi$$
$$\leq \left|b_{v_{T-1}}^i\right| + \sum_{a\in\delta^+_{G^T_{st}}(v_{T-1})} x_a^i.$$

This leaves us to check flow conservation for nodes v with a supply of commodity i, i. e. $b_v^i > 0$. For a $\theta \in \{1,\dots,T-1\}$, we have $b^i_{v_\theta} = 0$ and $\delta_G^-(v) = \emptyset$ by construction and get

$$\sum_{a\in\delta^-_{G^T_{st}}(v_\theta)} \gamma_a x_a^i = x^i_{(v_{\theta-1},v_\theta)} = \sum_{a\in\delta_G^+(v)} \int_{\theta+1}^{T} f_a^i(\xi)\, d\xi + \sum_{a\in\delta_G^+(v)} \int_{\theta}^{\theta+1} f_a^i(\xi)\, d\xi$$
$$= x^i_{(v_\theta,v_{\theta+1})} + \sum_{a\in\delta_G^+(v)} x^i_{a_\theta} = \sum_{a\in\delta^+_{G^T_{st}}(v_\theta)} x_a^i.$$

For $\theta = 0$, we get the following from the fact that f obeys flow conservation:

$$b^i_{v_0} + \sum_{a\in\delta^-_{G^T_{st}}(v_\theta)} \gamma_a x_a^i = b^i_{v_0} \geq \sum_{a\in\delta_G^+(v)} \int_{1}^{T} f_a^i(\xi)\, d\xi + \sum_{a\in\delta_G^+(v)} \int_{0}^{1} f_a^i(\xi)\, d\xi$$
$$= x^i_{(v_0,v_1)} + \sum_{a\in\delta_G^+(v)} x^i_{a_0} = \sum_{a\in\delta^+_{G^T_{st}}(v_0)} x_a^i.$$

With this, we have established flow conservation for x. Thus, x is a feasible flow in G^T_{st}.

Value and cost of x. Now we need to check that x and f are equivalent in terms of flow sent and cost. We begin with the former:

$$|x^i| = \sum_{\substack{v_\theta\in V(G^T_{st}),\\ b^i_{v_\theta}<0}} \sum_{a\in\delta^-_{G^T_{st}}(v_\theta)} \gamma_a x_a^i = \sum_{\substack{v\in V(G),\\ b^i_v<0}} \left(x^i_{(v_{T-2},v_{T-1})} + \sum_{a\in\delta_G^-(v)} \gamma_a x^i_{a_{T-1-\tau_a}} \right)$$
$$= \sum_{\substack{v\in V(G),\\ b^i_v<0}} \sum_{a\in\delta_G^-(v)} \sum_{\theta=0}^{T-1-\tau_a} \gamma_a x^i_{a_\theta} = \sum_{\substack{v\in V(G),\\ b^i_v<0}} \sum_{a\in\delta_G^-(v)} \int_0^{T-\tau_a} \gamma_a f_a^i(\theta)\, d\theta = |f^i|.$$

For the cost of x, we get

$$c(x) = \sum_{i\in K} \sum_{a_\theta\in A(G^T_{st})} c_{a_\theta} x^i_{a_\theta} = \sum_{i\in K} \sum_{a_\theta\in A(G^T_{st})} c_a \int_\theta^{\theta+1} f_a^i(\xi)\, d\xi$$
$$= \sum_{i\in K} \sum_{a\in A(G)} c_a \int_0^{T-\tau_a} f_a^i(\xi)\, d\xi = c(f).$$

This proves the first part of the theorem, which leaves us to show the reverse direction. For this, we have to define a flow over time f from a static flow x in the time-expanded network and show that f fulfills all requirements – non-negativity, capacity and flow conservation constraints, and that f and x have an equal flow value and cost.

Constructing f from x. For the other direction, we construct a flow over time based on the static flow in the time-expanded network by taking the flow value of an arc there, and interpret it as the flow rate for the original arc in the time interval that the copy represents. More formally, given x, we define f by

$$f_a^i(\theta) := x^i_{a_{\lfloor\theta\rfloor}} \qquad \text{for all } a \in A(G),\, i \in K,\, \theta \in [0,T).$$

Non-negativity and capacity constraints of f. Since $x_a^i \geq 0$ for all $a \in A(G_{st}^T)$, $i \in K$, we get non-negativity by default. f fulfills the capacity constraints, because x does as well:

$$\sum_{i\in K} f_a^i(\theta) = \sum_{i\in K} x^i_{a_{\lfloor\theta\rfloor}} \leq u_a \text{ for all } a \in A(G),\, \theta \in [0,T).$$

Flow conservation of f. Again, we assume that $f_a(\theta) = 0$ for all arcs $a \in A(G)$ and $\theta \notin [0,T)$, and that $x_a = 0$, if a refers to a non-existing arc like (v_{-1}, w_0). For every node $v \in V(G)$, commodity $i \in K$ and time $\theta \in [0,T)$, we have

$$\begin{aligned}
\max(0,b_v^i) + \sum_{a\in\delta_G^-(v)} \int_0^{\theta-\tau_a} \gamma_a f_a^i(\xi)\ d\xi &= \max(0,b_v^i) + \sum_{a\in\delta_G^-(v)} \sum_{\xi=0}^{\theta-\tau_a-1} \gamma_a x^i_{a_\xi}\ d\xi \\
&= \max(0,b_v^i) + x^i_{(v_{\theta-1},v_\theta)} + \sum_{a\in\delta_G^+(v)} \sum_{\xi=0}^{\theta-1} x^i_{a_\theta} \\
&\overset{(2)}{\geq} \min(0,b_v^i) + \sum_{a\in\delta_G^+(v)} \sum_{\xi=0}^{\theta-1} x^i_{a_\theta} \\
&= \min(0,b_v^i) + \sum_{a\in\delta_G^+(v)} \int_0^{\theta} f_a^i(\xi)\ d\xi.
\end{aligned}$$

For $\theta = T$, the inequality (2) becomes an equality, since (v_{T-1}, v_T) does not exist and x satisfies all supplies and demands. Thus, f fulfills flow conservation.

Value and cost of f. Now we need to check that x and f are equivalent in terms of flow sent and cost. We begin with the former and make use of the fact that sinks do not have outgoing arcs:

$$\begin{aligned}|f^i| &= \sum_{\substack{v\in V(G),\\ b_v^i<0}} \sum_{a\in\delta_G^-(v)} \int_0^{T-\tau_a} \gamma_a f_a^i(\theta)\ d\theta = \sum_{\substack{v\in V(G),\\ b_v^i<0}} \sum_{a\in\delta_G^-(v)} \sum_{\theta=0}^{T-1-\tau_a} \gamma_a x^i_{a_\theta} \\ &= \sum_{\substack{v\in V(G),\\ b_v^i<0}} \left(x^i_{(v_{T-2},v_{T-1})} + \sum_{a\in\delta_G^-(v)} \gamma_a x^i_{a_{T-1-\tau_a}} \right) = \sum_{\substack{v_\theta\in V(G^T_{st}),\\ b^i_{v_\theta}<0}} \sum_{a\in\delta^-_{G^T_{st}}(v_\theta)} \gamma_a x_a^i = |x^i|.\end{aligned}$$

For the cost of f, we get

$$\begin{aligned}c(f) =& \sum_{i\in K}\sum_{a\in A(G)} c_a \int_0^{T-\tau_a} f_a^i(\xi)\ d\xi \\ =& \sum_{i\in K}\sum_{a\in A(G)} c_a \sum_{\xi=0}^{T-\tau_a-1} x^i_{a_\xi} = \sum_{i\in K}\sum_{a\in A(G^T_{st})} c_a x_a^i = c(x).\end{aligned}$$

This completes the proof of the theorem. □

Thus, we can solve static flow problems in the time-expanded graph to solve flow over time problems in the original graph. For flows over time without storage, we get the following theorem, which can be proved analogously to Theorem 4.3.

Theorem 4.4. *Let G be a directed graph with integral transit times $\tau : A(G) \to \mathbb{N}_0$, capacities $u : A(G) \to \mathbb{R}_0^+$, costs $c : A(G) \to \mathbb{R}$, gains $\gamma : A(G) \to \mathbb{R}_0^+$, commodities $K = \{1,\dots,k\}$, balances $b : V(G)\times K \to \mathbb{R}$ and an integral time horizon $T \in \mathbb{N}_0$, with the corresponding time-expanded network without storage G^T and extended capacities $u^T : A(G^T) \to \mathbb{R}_0^+$, costs $c^T : A(G^T) \to \mathbb{R}$, gains $\gamma^T : A(G^T) \to \mathbb{R}_0^+$, balances $b^T : V(G^T)\times K \to \mathbb{R}$. For any multi-commodity flow over time f without storage in G there exists a static multi-commodity flow x in G^T that sends the same amount of flow for every commodity with the same cost, i.e., $|f^i| = |x^i|$ for all commodities $i \in K$ and $c(f) = c(x)$. Analogously, there exists a multi-commodity flow over time f without storage in G for any static multi-commodity flow x in G^T with $|f^i| = |x^i|$ for all commodities $i \in K$ and $c(f) = c(x)$.*

4.1.3 Extensions

We continue by briefly discussing possible extensions to time-expanded networks as discussed above. Notice that this list is by no means exhaustive.

Non-integral transit times. So far, we have considered integral transit times. Technically, we can use time expansion also for transit times or time horizons that are rational numbers by scaling the transit times so that they all become integral. This causes an additional blow-up in network size, however. If we assume that a transit time τ_a is given by a nominator n_a and a denominator d_a, we can scale all transit times and the time horizon by the least common denominator of all $d_a, a \in A$, which is at most $\prod_{a \in A(G)} d_a$. This guarantees the integrality of all transit times after the scaling. Notice that in order to keep the amount of flow send the same, we have to scale the capacities by the same factor to accommodate the finer time scale (see also the discussion in Chapter 5.1). We can handle a rational time horizon in a similar way.

For real numbers, this is not always possible. However, irrational numbers are hard to use computationally as there is usually a limit to the precision with which the computations are done. And since any real number has rational numbers arbitrarily close to it, it makes sense if we limit ourselves to rational numbers.

Attributes of holdover arcs. Depending on the application the time expansion is used for, it can be useful to assign finite capacities to holdover arcs – e.g., if holdover arcs model buffers of a limited size in telecommunication networks. Similarly, assigning non-zero costs to them can be used to model storage costs in logistic networks and non-unit gains can model interest of investments in financial flows. We will not make use of these modelling capabilities here; however, they are an advantage of time expansion that should be noted.

Time-dependent attributes. Another modeling advantage of time expansion is that it is easy to realize time-dependent attributes like capacities, costs and transit times. This is due to the fact that we have copies of a node or an arc for every point in time. Thus, we can just give each copy of a node or an arc different attributes. Tjandra [109, 110] discusses this approach in depth.

Flow-dependent attributes. Time expansion can also be used to approximate flow-dependent attributes – for example, transit times that change with the rate of flow entering an arc or costs that scale non-linearly with the amount of flow. We give a brief discussion of this idea (which is due to Köhler, Langkau und Skutella [72]) for the case of transit times that vary based on the rate of flow entering an arc. This type of flow-dependent transit times is referred to as *inflow-dependent* transit times. The basic idea for the approximation is to replace a single arc with inflow-dependent transit times with a structure that permits multiple ways between start and end of the arc, where each way has a limited capacity and increasing transit times. The resulting network is called a *bow graph* and its time expansion a *fan graph*. The first flow units can then use the fastest way but after the capacity of the fastest way is reached, the second-fastest way needs to be used, and so on. For a more detailed explanation and analysis of this technique we refer to [72]. A small sketch of a bow graph and its fan graph is depicted in Figure 4.3.

Cyclically expanded networks. For applications that deal with problems that show a periodic structure, e.g., the optimization of traffic signals, it can be useful to a employ a time-expanded network that is periodic as well. This means that copies of arcs that would normally not be realized in the time expansion due to leaving the time horizon, do now exist to generate the periodicity. Thus, an arc $a = (v, w)$ with a transit time of 2 would now get a copy $a_{T-1} = (v_{T-1}, w_1)$ if T is our time horizon. The resulting time-expanded network is called a *cyclically expanded network* or a *pattern expanded network*. Figure 4.3 shows a sketch of such a network. For more details on this technique and setting, we refer to Köhler and Strehler [73, 74] or Harks, König, Matuschke, Richter and Schulz [59]. Another approach to make use of periodicity has been proposed by Bast and Storandt [9], who use *frequency data compression*, a technique that compresses the periodic structures in a time-expanded graph.

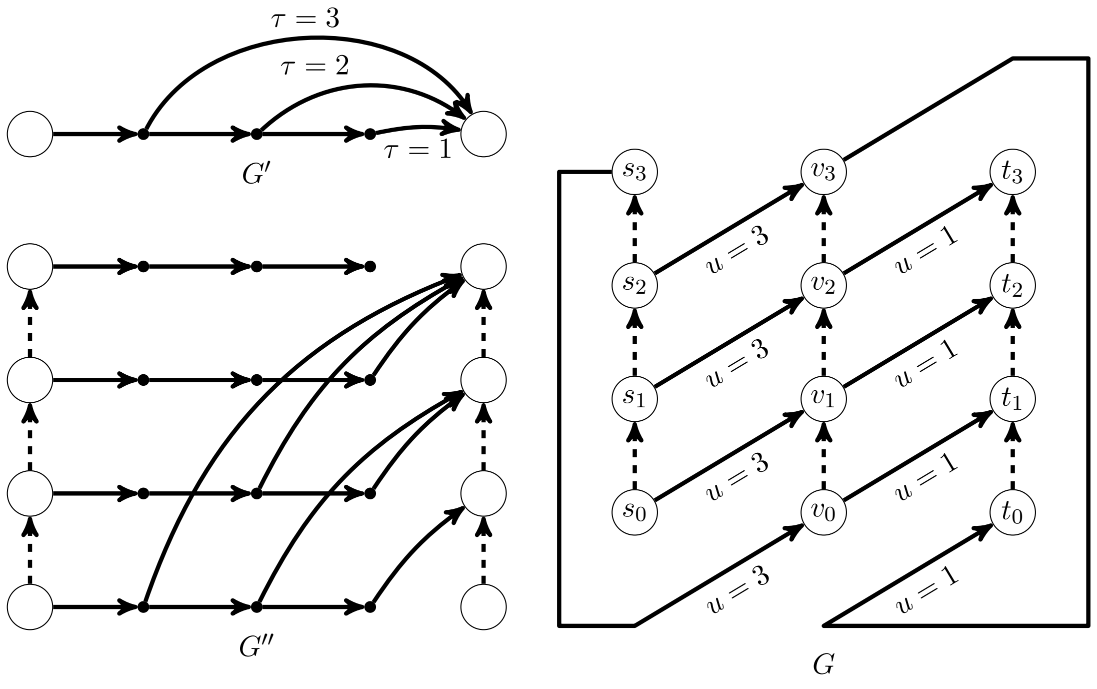


Figure 4.3: A bow graph G' and its fan graph G'', and a sketch of a cyclically expanded network G.

4.1.4 Discussion

By Theorem 4.3 and 4.4 we know that we can solve static network flow problems in a time-expanded network instead of solving flow over time problems in the original network. Combined with the possibility for extensions that we have sketched in the last section, this makes time expansion a powerful technique for these problems – they allow us to use all algorithms and techniques for static problems (e. g., residual networks) and offer easy extensibility to accomodate side-constraints.

However, the major disadvantage of time expansion is that the network size increases linearly in the time horizon. If the time horizon is not polynomially bounded in the size of the network, which we cannot assume in general, time expansion will cause a superpolynomial blow-up in network size.

Still, due to their powerful modelling capabilities and easy applicability, time-expanded networks find widely spread use in applications. Recent examples include data transfer optimization [107], evacuation management [26, 27], public transportation routing [8, 9], tour planning [60], traffic and traffic signal optimization [15, 73, 74], train timetabling [29], and water supply management [93].

Thus, these applications have to deal with potentially very large networks. This can be done in several ways. The most straight-forward one is by usage of vast amounts of computation power. More sophisticated are techniques that either generate only the actually used parts of a time-expanded network, or which do not generate the time-expanded network explicitly at all, i. e., by data structures that store the flow rate of an arc at every point in time. Depending on the flow rate functions for the arcs, this can be much more compact than an actual time-expanded network. However, we have usually no guarantee that the flow rate functions behave nicely or that only small parts of the time-expanded network need to be generated. Neither can we assume that we have an infinite amount of computation power or that the instances are small enough. Thus, the main algorithmic problem is to reduce the network size somehow – a problem we will deal with in Chapter 5.

4.2 Iterative Time Expansion

We will now discuss a modification of time expansion that is designed to compute 2-value-approximative earliest arrival flows. This approach constructs the time-expanded network gradually, time-step by time-step, and freezes the arriving flow at the sinks for every layer that is not the current one. We will see in the proof of the next theorem, that this construction is powerful enough to give us the desired 2-value-approximation for earliest arrival flows. This result was first published in Groß, Kappmeier, Schmidt and Schmidt [50].

Theorem 4.5. *Let (G, u, τ, b, T) be an instance of the earliest arrival flow problem. Then there exists a 2-value-approximative earliest arrival flow for this instance in the discrete time model.*

Proof. We will prove this result constructively by describing an algorithm for this problem. The algorithm will iteratively compute static maximum flows in residual time-expanded networks in which the time horizon increases by 1 in each iteration. To ensure the 2-value-approximation guarantee, we remove some arcs from the network before performing a maximum flow computation.

For this algorithm, we will use a slightly modified type of time-expanded network, that transforms our problem into a problem with a single super-source σ and single super-sink ω. For a given network (G, u, τ, b, T) we define our time-expanded network G^T by

$$V(G^T) := \{ v_\theta \mid v \in V(G), \theta \in \{0, \dots, T-1\}\} \cup \{ s^* \mid s \in S_+ \cup S_- \} \cup \{\sigma, \omega\},$$

and

$$\begin{aligned}A(G^T) := &\{\, a_\theta = (v_\theta, w_{\theta+\tau_a}) \mid a = (v,w) \in A(G), \theta \in \{0,\dots,T-1\}\}\\ &\cup \{\,(\sigma, s^*) \mid s \in S_+\} \cup \{\,(s^*, s_\theta) \mid s \in S_+, \theta \in \{0,\dots,T-1\}\}\\ &\cup \{\,(t^*, \omega) \mid t \in S_-\} \cup \{\,(t_\theta, t^*) \mid t \in S_-, \theta \in \{0,\dots,T-1\}\}\,.\end{aligned}$$

The capacities u^T are defined for all $a \in A(G^T)$ by

$$u_a^T := \begin{cases} u_a & a = a_\theta, \\ b_s & a = (\sigma, s^*), \\ -b_t & a = (t^*, \omega), \\ \infty & \text{else.} \end{cases}$$

The supplies and demands b^T are given by $b_\sigma := B_+$ and $b_\omega := -b_\sigma$. Figure 4.4 shows such a time-expanded network.

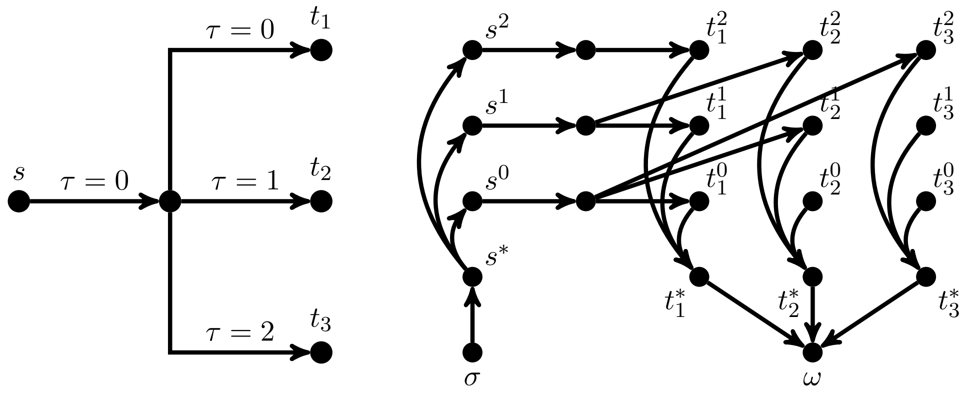


Figure 4.4: A graph G and its time expansion G^3 as defined above.

Since we are only considering single commodity earliest arrival flows, we omit the storage at intermediate nodes here. Adding it would be no problem for the construction, but it would get slightly more convoluted. The storage at sources and sinks is realized by the arcs of the form (s^*, s^*_θ) and (t^*_θ, t^*). These arcs are also the arcs that we will delete, later on. The algorithm is as follows.

1. Let $f^0 \equiv 0$ be the zero-flow in G^1 and set $\theta := 0$.
2. We define a subgraph $\overline{G}_{f^\theta}^{\theta+1}$ of the residual time-expanded network $G_{f^\theta}^{\theta+1}$ by removing the arcs (t^*, t^ξ) for all points in time $\xi = 0, \dots, \theta$ and all sinks $t \in S_-$.
3. Compute a static maximum σ-ω-flow x^θ in $\overline{G}_{f^\theta}^{\theta+1}$.
4. Add x^θ to our hitherto existing flow f^θ, yielding a new flow $f^{\theta+1} := f^\theta + x^\theta$.

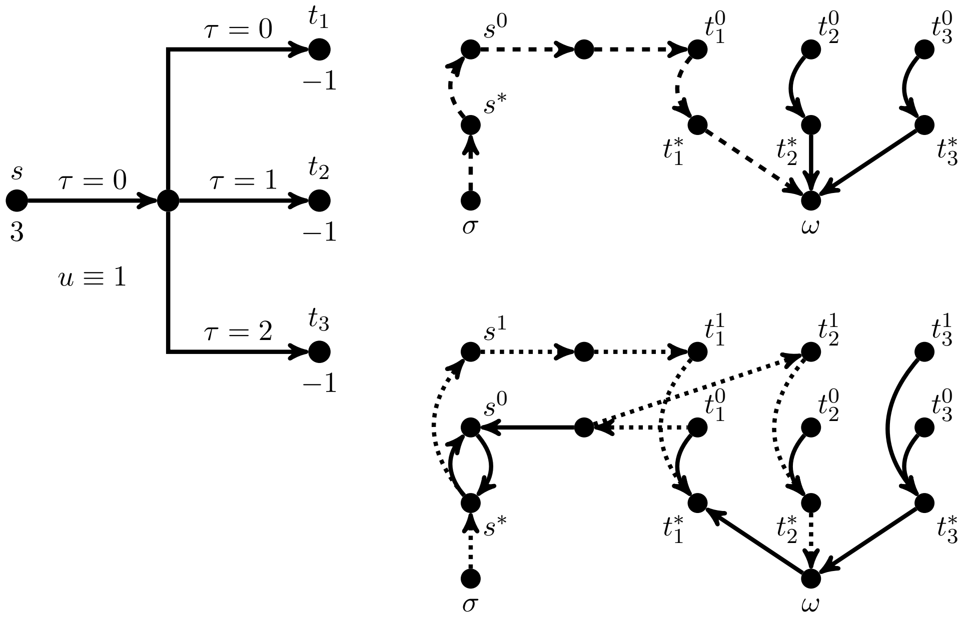


Figure 4.5: A graph G and its time expansion G^1 with a maximum flow x^0 marked by the dashed arcs. Below that the modified residual time-expanded network $\overline{G}^2_{x^0}$. Notice the missing (t_1^*, t_1^0) arc, which would complete the dotted σ-ω-path.

5. If not all supplies and demands have been fulfilled by $f^{\theta+1}$, set $\theta := \theta + 1$ and go back to Step 2, otherwise stop and output $f^{\theta+1}$.

Notice that the arcs that we remove to create $\overline{G}^{\theta+1}_{f^\theta}$ are reverse arcs and can only be present if we sent flow through their normal counterparts before. If we would not delete these arcs, we would iteratively compute maximum flows for the time horizons $1, 2$, and so on.

In order to show that this algorithm computes a 2-value-approximate earliest arrival flow, we prove that f^θ is a 2-value-approximate earliest arrival flow with time horizon θ. Let $f^\theta_{\max}$ be a maximum flow in the time-expanded network with time horizon θ. Define a new flow $g := |f^\theta_{\max} - f^\theta|$ in $G^\theta_{f^\theta}$ by

$$g_a := \begin{cases} (f^\theta_{\max})_a - (f^\theta)_a & a \in A(G^\theta), (f^\theta_{\max})_a - (f^\theta)_a \geq 0 \\ (f^\theta)_a - (f^\theta_{\max})_a & \overleftarrow{a} \in A(G^\theta), (f^\theta_{\max})_a - (f^\theta)_a < 0 \end{cases} \quad \text{for all } a \in A(G^\theta_{f^\theta}).$$

By definition of g, we have that $|f^\theta_{\max}| = |f^\theta| + |g|$. This flow is valid in the residual time-expanded network $G^\theta_{f^\theta}$, but not in $\overline{G}^\theta_{f^\theta}$ where some arcs were removed. In fact, since f^θ contains a maximum flow in $\overline{G}^\theta_{f^{\theta-1}}$, every flow unit in g must make use of the deleted arcs. The difference in the amount of flow between $G^\theta_{f^\theta}$ and $\overline{G}^\theta_{f^\theta}$ is bounded by the sum of the capacities of the deleted arcs, which were exclusively reverse arcs. Thus, the sum of the capacities of the deleted arcs is at most $|f^\theta|$.

It follows that $|g| \leq |f^\theta|$ and therefore $|f^\theta_{\max}| \leq |f^\theta| + |f^\theta|$, which proves that f^θ is in fact a 2-value-approximate earliest arrival flow. □

The algorithm described in Theorem 4.5 has only a pseudo-polynomial running time, since it is based on time expansion. However, as we are only interested in an existence result here, this is fine for our purposes. For a more efficient algorithm, see our FPTAS in Chapter 5.2.

Continuous time-model. Notice that the above algorithm only works in the discrete time model. For the continuous model, we conclude that there always exists a $(1+\varepsilon)$-time and 2-value-approximate earliest arrival flow due to the following lemma. This lemma assumes that no flow arrives before time 1. If all transit times are positive, this can be guaranteed by scaling transit times to make them integer. Otherwise, zero transit times have to be handled separately along the lines discussed in [32]. However, applications usually have positive transit times.

Lemma 4.6. *Let f be a α-time-β-value-approximate earliest arrival flow in the discrete time model, with ε as the length of a time step, in a network which has no flow arriving before time 1. Then there exists a $(1+\varepsilon)\alpha$-time-β-value-approximation in the continuous time model.*

Proof. Notice that we can interpret f as a continuous flow over time that sends flow at a constant rate during a time step. The value-guarantee is fulfilled up to time 1, since no flow can arrive earlier. For all later points in time θ that are multiples of ε, we know that $|f(\theta)| \geq p(\theta/\alpha)/\beta$. Furthermore, we know that p is monotonically increasing. This means that for a time point $\theta \in (k\varepsilon, (k+1)\varepsilon)$, $k \in \mathbb{N}$ we know that $|f((k+1)\varepsilon)| \geq p((k+1)\varepsilon/\alpha)/\beta \geq p(\theta/\alpha)/\beta$. Thus, our time-approximation guarantee gets worse by a constant value of ε. Since no flow arrives before time 1, this gives a factor of at most $(1+\varepsilon)$, proving our claim. □

Zero-transit times. We conclude this chapter by studying the algorithm from above in the special case of zero-transit times. As we discussed earlier, in this case earliest arrival flows exist always if we have either a single source or a single sink (Fleischer [34]), and for multiple sources and sinks we have a characterization of all networks that always allow for earliest arrival flows regardless of the capacities and supplies and demands (Schmidt and Skutella [101]).

As it turns out, zero-transit times are beneficial to our 2-value-approximative algorithm of Theorem 4.5 as well. In this setting, we can implement the algorithm by computing static maximum flows and repeating them until a terminal runs out of supply or demand. As this can happen at most once per terminal, we have at most $O(n)$ maximum flow computations. The approach works in the discrete and in the continuous time model. This leads to the following algorithm.

1. Define new balances $b'_v := b_v$ and set $\theta := 1$,

2. Compute a static maximum σ-ω-flow x^θ respecting b' and define
$$\mathrm{ex}_{x^\theta}(v) := \sum_{a\in\delta^-(v)} x^\theta_a - \sum_{a\in\delta^+(v)} x^\theta_a,$$
3. Determine the maximal amount of times δ^θ we can send x^θ until a source or sink runs empty,
$$\begin{array}{lrcll} \max & \delta^\theta, & & & \\ \text{s.t.} & b'_v + \delta^\theta \cdot \mathrm{ex}_{x^\theta}(v) & \geq & 0 & \text{for all } v \in S_+, \\ & b'_v + \delta^\theta \cdot \mathrm{ex}_{x^\theta}(v) & \leq & 0 & \text{for all } v \in S_-. \end{array}$$
4. Update the balances according to the flow x^θ sent:
$$b'_v := b'_v + \delta^\theta \cdot \mathrm{ex}_{x^\theta}(v) \text{ for all } v \in V(G),$$
5. If $b' \neq 0$, set $\theta := \theta + 1$ and go to Step 2.
6. Output the flow over time f that sends x^θ starting at $\sum_{j=1}^{\theta-1} \delta^j$ for δ^θ time units.

The correctness of this algorithm follows analogously to Theorem 4.5. The difference is here that due to the zero transit times we always send a static maximum flow at any point in time, which changes when terminals run out.

Lemma 4.7. *For networks with zero transit times, a 2-value-approximate earliest arrival flow can be computed with at most t static maximum flow computations in the continuous time model and at most $t \log b_{max}$ static maximum flow computations in the discrete time model, with $t := |S_+ \cup S_-|$ being the number of terminals and $b_{max} := \max_{v\in S_+\cup S_-} |b_v|$ being the highest supply or demand.*

Proof. The algorithm performs one static maximum flow calculation per step. The choice of x guarantees that at least one source or sink runs empty in every iteration, ensuring that $b' = 0$ after t iterations. Note that in the discrete time model, we have to choose an integer x_i, so we get $\lfloor x_i \rfloor$ compared to the continuous case. This means that we have sent more than half of supply or demand of a terminal since otherwise, we could have send f_i at least one more time. This leads to at most $\log b_{max}$ iterations per terminal, since our capacities and balances are integral. □

5 Time Condensation

In the last chapter, we studied time expansion. While being a very powerful technique, time expansion has the drawback of potentially causing exponential blow-ups in the network size. This is due to the fact that we copy the original network for every discrete time step until the time horizon. Thus, if we want an efficient algorithm that uses some form of time expansion, it is necessary to apply some form of rounding to the time horizon and the transit times in order to contain the blow-up. This rounding is referred to as time condensation in the setting of time-expanded networks. Since rounding introduces an error, we have to find the correct balance between the accuracy of the solution and the size of the blow-up.

We begin by giving an introduction into time condensation in a form that is applicable to generalized (multi-commodity) flows with lossy gains. This particular case was first published in Groß and Skutella [51], and builds very closely on the previous work by Fleischer and Skutella [32] who described flow condensation for minimum cost multi-commodity flows over time with conservative costs. After that, we discuss a special form of time condensation that is useful for earliest arrival flow problems to construct an FPTAS that computes the best possible approximations for earliest arrival flows in arbitrary networks. This result was published in Groß, Kappmeier, Schmidt and Schmidt [50]. We conclude the chapter by introducing sequence condensation, a more powerful generalization of time condensation which was developed by Groß and Skutella [52].

5.1 Time Condensation

The main idea behind time condensation is the following. If the time horizon is polynomially bounded in the size of the network, we can use time expansion and experience only a polynomial blow-up in network size, which would be fine for our purposes. This rises the question whether we might be able to simply scale down time horizons that are too large.

In some settings, this is surely possible. Consider a network where all transit times and the time horizon are whole minutes, but everything is specified in seconds. Thus, all transit times and the time horizon are multiples of 60. In this case, we could change the time unit from seconds to minutes, which is equivalent to dividing everything by 60. This seems like an insignificant change, but the time expansion of the network with times in minutes is a factor of 60 smaller than the time expansion of the network with times in seconds. The general concept behind this idea are *condensed time-expanded networks* or just *time-condensed networks* for short. We will now define time-condensed networks. When doing so, we have to take care to adjust capacities along with transit times, when we scale them. This is because a capacity is a bound on the maximal rate into an arc at a given time. Thus, a rate of 2 per second would be scaled to 120 per minute in our previous example.

Definition 5.1 (Condensed Time-Expanded Networks)**.** *Let $N = (G, u, c, \tau, \gamma, b, T)$ be a network. In order to define the* condensed time-expanded network *$N^T/\Delta = (G^T/\Delta, u', c', \gamma', b')$ of N for a $\Delta > 0$, we begin by defining scaled transit times and a scaled time horizon:*

$$\tau'_a := \lceil \tau_a/\Delta \rceil \text{ for all } a \in A(G), \quad T' := \lceil T/\Delta \rceil .$$

With these scaled transit times and time horizon, we define the actual condensed time-expanded network:

$$\begin{aligned} V(G^T/\Delta) &:= \left\{ v_\theta \mid v \in V(G), \theta = 0, 1, \dots, T'-1 \right\}, \\ A(G^T/\Delta) &:= \left\{ a_\theta = \left(v_\theta, w_{\theta+\tau'_a}\right) \mid a = (v,w) \in A(G), \theta = 0, \dots, T' - \tau'_a - 1 \right\} \\ &\quad \cup \left\{ a_{v,\theta} = (v_\theta, v_{\theta+1}) \mid v \in V(G), \theta = 0, \dots, T'-2 \right\}. \end{aligned}$$

Capacities for the condensed time-expanded network are adjusted to take the scaling of time into account:

$$u'_a := \begin{cases} \Delta u_a & a = a_\theta \\ \infty & a = (v_\theta, v_{\theta+1}) \end{cases} \quad \text{for all } a \in A(G^T/\Delta).$$

Gains, costs, as well as supplies and demands are handled like in normal time-expanded networks:

$$\gamma'_a := \begin{cases} \gamma_a & a = a_\theta \\ 1 & a = (v_\theta, v_{\theta+1}) \end{cases}, \quad c'_a := \begin{cases} c_a & a = a_\theta \\ 0 & a = (v_\theta, v_{\theta+1}) \end{cases} \quad \text{for all } a \in A(G^T/\Delta),$$

$$(b')^i_{v_\theta} := \begin{cases} b^i_v & \theta = 0, b^i_v > 0 \\ b^i_v & \theta = T'-1, b^i_v < 0 \\ 0 & \text{else} \end{cases} \quad \text{for all } v_\theta \in V(G^T/\Delta).$$

Notice that we did not assume that the transit times and the time horizon are multiples of Δ in the above definition. Also, we included storage arcs in the above definition, but these arcs can easily be removed if this is not desired. We will now begin our analysis with the case that they are multiples, and then generalize to arbitrary transit times and time horizons.

Analogously to normal time-expanded networks, there is an equivalence between static flows in the Δ-condensed time-expanded network and generalized flows over time in the original network.

Lemma 5.2. *If T and τ_a are multiples of Δ for all $a \in A(G)$, then every generalized multi-commodity flow over time f in a network N that is completes within the time horizon T corresponds to a static generalized multi-commodity flow x in the Δ-condensed time-expanded network N^T/Δ and vice versa. The corresponding flows send the same amount of flow and incur the same costs.*

Proof. Let f be an arbitrary generalized flow over time in G. We define a static generalized flow x by accumulating the flow values $f_a(\theta)$ during a time interval between two multiples of Δ:

$$x^i_{a_\theta} := \int_{\theta\Delta}^{(\theta+1)\Delta} f^i_a(\xi)\ d\xi \quad \text{for all } a \in A(G), i \in K, \theta = 0, \dots, T' - \tau'_a - 1.$$

The flow values of the holdover arcs are chosen such that flow conservation is guaranteed. We give a definition for flow values of holdover arcs at intermediate nodes, the flow values for holdover arcs at sources and sinks are defined analogously:

$$x^i_{(v_\theta, v_\theta+1)} := \sum_{a\in\delta^-_G(v)} \int_0^{(\theta+1)\Delta-\tau_a} \gamma_a f^i_a(\xi)\ d\xi - \sum_{a\in\delta^+_G(v)} \int_0^{(\theta+1)\Delta} f^i_a(\xi)\ d\xi$$

for all $i \in K, \theta = 0, \dots, T' - 2$ and all nodes $v \in V(G)$ with $b^i_v = 0$. Capacity constraints are obeyed by x, since

$$\sum_{i\in K} x^i_{a_\theta} = \int_{\theta\Delta}^{(\theta+1)\Delta} \sum_{i\in K} f^i_a(\xi)\ d\xi \le \int_{\theta\Delta}^{(\theta+1)\Delta} u_a\ d\xi = \Delta u_a$$

for all $i \in K, a \in A(G), \theta = 0, \dots, T' - 1$. For holdover arcs, the capacities are infinite and cannot be violated. Since the total amount of flow through an arc does not change, neither does the flow value or the cost.

On the other hand we can construct a generalized flow over time f in G from a static generalized flow x in G^T/Δ:

$$f^i_a(\theta) := \frac{x_{a_{\lfloor\theta/\Delta\rfloor}}}{\Delta} \quad \text{for all } i \in K, a \in A(G), \theta \in [0, T).$$

Since $x_{a_\theta} \le \Delta u_a$ by definition, we have that $f^i_a(\theta) \le u_a$ for all $i \in K, a \in A(G), \theta \in [0, T)$. Because x satisfies flow conservation, so does f as it is derived by sending a scaled x over a time interval. The total amount of flow send through an arc is again not changed, so that the total amount of flow and its cost remains the same. □

If only the transit times are multiples of Δ but the time horizon is not, we can get a similar although slightly weaker result.

Corollary 5.3. *If τ_a are multiples of Δ for all $a \in A(G)$, then every generalized multi-commodity flow over time f in a network G that completes within a time horizon T corresponds to a static generalized multi-commodity flow x in the Δ-condensed time-expanded network G^T/Δ and every static generalized multi-commodity flow in G^T/Δ corresponds to a generalized multi-commodity flow over time that is completed within the time horizon $T + \Delta$. The corresponding flows send the same amount of flow at the same cost.*

5.1.1 Time Condensation with Arbitrary Transit Times

So far, we have a time condensation that works only with transit times that are a multiple of some $\Delta > 0$. Of course, all transit times are multiples of 1, but choosing $\Delta = 1$ does not decrease the size of the time expansion. Thus, we need to be able to find a Δ that is sufficiently large in order to reduce the size of the blow-up. In other words, we must make arbitrary transit times multiples of a Δ that is large enough for our purposes. In principle, we can pick a Δ and round all transit times up to the next multiple of Δ. This will introduce a rounding error, which causes two major problems: an increase in path lengths, which can cause viable paths to become longer than the time horizon, and a distortion of path lengths, where path lengths change disproportionately to each other.

Problem: Increased path lengths. We have already seen in the case of transit times that are multiples of Δ and a time horizon which is not that a flow in the condensed time-expanded network corresponds to a flow over time that completes in time $T + \Delta$, and not in T. This effect is exacerbated if the transit times are not multiples Δ as well, and are rounded up to the next multiple instead. Consider the example in Figure 5.1. Here we have a path of arcs with transit time 1 and a time horizon of T. We round up the transit times, we get an increase in path length that is $\Delta - 1$ per arc. Thus, the length of the path increases by a factor of almost Δ.

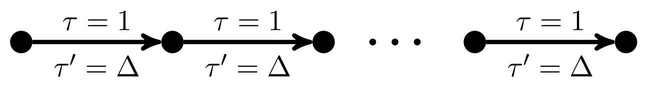


Figure 5.1: The transit time of a path can greatly increase when switching from the original transit times τ to transit times τ' rounded up to the next multiple of Δ.

Thus, if we do not want to lose previously feasible paths after the rounding, we have to consider an increased time horizon. The more we have to increase the time horizon, the worse our time-approximation gets. Thus, our first problem is to bound the increase of path lengths.

Solution: Bounding the increase in path length. For arcs $a \in A(G)$ we define rounded transit times by

$$\tau'_a := \left\lceil \frac{\tau_a}{\Delta} \right\rceil \Delta \quad \text{for all } a \in A(G).$$

These rounded transit times are by construction all multiples of Δ, therefore we have

$$\tau_a \leq \tau'_a \leq \tau_e + \Delta \quad \text{for all } a \in A(G).$$

Let $S = (a_1, \dots, a_{|S|})$ be an arc sequence in G. Since the rounding error per individual arc is bounded by Δ, we get the following bound for the rounding error of the length of S:

$$\tau_S \leq \tau'_S \leq \tau_S + \Delta |S| \ .$$

Thus, if S is a simple arc sequence – a path – then the rounding error is bounded by Δn. We restrict our application of time condensation to problems that have always optimal solutions that send flow only along simple sequences. Examples for such problems are (maximum) multi-commodity flow problems with storage, minimum cost flows without negative cycles and generalized flows over time with lossy gains. For all of these problems, cycles do not benefit the objective, and can be replaced by waiting. For other problems, we have to use sequence condensation (Chapter 5.3).

How should be choose Δ for such problems? We would like two properties:

- the path lengths should increase at most by a factor of $1 + O(\varepsilon)$ for $\varepsilon \to 0$:
$$T + \Delta n \leq (1 + O(\varepsilon))T,$$
- and the number of time layers after scaling by Δ should be polynomial in the size of the original network and ε^1:
$$\frac{T}{\Delta} \in O(\text{poly}(n, \varepsilon^{-1})).$$

We will now choose $\Delta := \varepsilon^2 T/n$. This fulfills both properties, as the maximal rounding error for a path is $\varepsilon^2 T$ and the number of time layers is $n\varepsilon^{-2}$.

Problem: Contorted path lengths. Since we round transit times of individual arcs, it can happen that paths with different lengths with regard to the non-rounded transit times τ can have the same length with regard to the rounded transit times τ'. Consider the example in Figure 5.2. With the unscaled transit times, we can send one unit of flow into the top and the bottom path during the time interval $[0, 1)$ and they arrive at arc a one after another – at time 3 and 4, respectively. However, with the scaled transit times for $\Delta = 2$ they arrive both at the same time, which can cause a violation of capacity constraints.

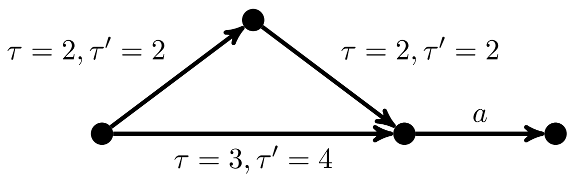


Figure 5.2: Paths with different transit times have the same transit times after scaling and rounding the transit times.

This can happen in the other direction as well – paths with the same lengths with regard to the non-rounded transit times τ can have the different lengths with regard to the rounded transit times τ'. Consider Figure 5.3. For $\Delta = 3$, we can send flow during the time interval $[0, 1)$ into the top and bottom path without causing conflicts, since the rounded transit times are 6 for the top and 3 for the bottom path. In the original network, both paths have a transit time of 3, however.

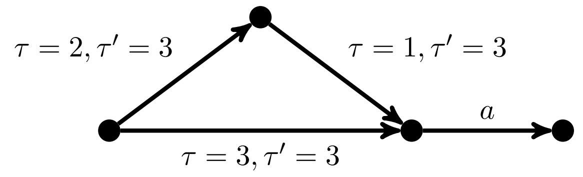


Figure 5.3: Paths with same transit times have different transit times after scaling and rounding the transit times.

The resulting capacity violations are problematic for two reasons. First, an optimal solution in the original network should somehow translate to a solution of the same value in the time-condensed network. Second, we want to be able to translate our solutions from the time-expanded network back to the original network. The second part is relatively easy, if we allow waiting in nodes. Then we can simulate the longer rounded transit times by waiting in a node, if necessary. However, this still leaves us to cope with the first problem.

Solution, Part I: Smoothed flows. Our main problem here is that flow units that arrived one after another may now be delayed so that they collide with each other. In order to lessen the impact of a collision, we now define a *smoothed* flow over time f_{sm} out of a given flow over time f. We write f as a sum of flows along a family of paths $\mathcal{P}$. Let $f_P(\theta)$ be the rate at which flow is sent into a path $P \in \mathcal{P}$ at time θ. The we define a path-based, smoothed version f^{sm} of f by:

$$f_P^{sm,i}(\theta) = \frac{1}{\varepsilon T} \int_{\theta - \varepsilon T}^{\theta} f_P^i(\xi)\, d\xi \qquad \text{for all } i \in K, , P \in \mathcal{P}, \theta \in [0, (1+\varepsilon)T)$$

Figure 5.4 illustrates what this smoothing does with a unit of flow – it averages the flow rate over a time window of length εT, thereby smearing the flow over a longer period of time. This prevents any sharp drops in flow rate at the beginning and the end of the flow unit. This has the side effect of increasing the time horizon that f_{sm} needs to complete by the length of the time window, which is εT in our case. Notice that the averaging cannot violate capacity constraints, and the total amount of flow through an arc also remains constant.

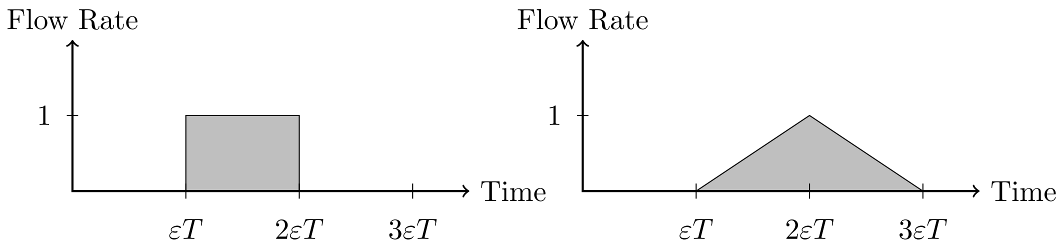


Figure 5.4: A flow rate function for an arc, and its smoothed counterpart.

Now, let us analyze what happens if we take a flow over time f that uses the normal transit times τ, smooth it, resulting into f^{sm} and then try to send f^{sm} using the rounded transit times τ'. Preferably, we would be able to show that f^{sm} violates capacities only slightly when using the rounded transit times τ'. In order to do so, we must first determine the flow rate into an arc $a \in A$ at a point in time θ. We do so by summing over all paths with contain our arc. For each such path, flow entering a path at time $\theta - \tau'_{P[\to v]}$ will enter a at θ, which is what we need (recall that $\tau'_{P[\to v]}$ refers to the time a flow unit needs to traverse P from its start to node v with regard to the transit times τ'). Finally, we need to take gain factors on the way to the start node $a = (v, w)$, which are denoted by $\gamma_{P[\to v]}$ for a path P. Then we can compute the flow rate of a flow over time f into an arc a for a commodity i at time θ:

$$f_a^i(\theta) = \sum_{P \in \mathcal{P}: a \in P} \gamma_{P[\to v]} f_P^i(\theta - \tau_{P[\to v]}) \ .$$

Notice that a flow over time with storage cannot be modelled by such path flows directly, because a path contains no possibility to specify time spend waiting in nodes. This can be fixed by adding a delay vector $\delta = (\delta_1, \dots, \delta_{|P|})$ to a path P, that specifies the waiting time at every node in P, resulting in a *path with delays* P^δ. For a path with delays P^δ, $P = (a_1, \dots, a_j, a, \dots, a_|P|)$, the time needed to traverse from its beginning to a node v is

$$\tau_{P^\delta[\to v]} := \sum_{i=1}^{j} (\tau_{a_i} + \delta_i) \, .$$

In order to check how much f^{sm} violates the capacities if we send it using the rounded transit times τ', we compute the total flow rate of f^{sm} into a at a point in time $\theta$for this scenario. Since we are interested in the total flow rate, we sum over the flow rates of all commodities:

$$f_a^{sm}(\theta) = \sum_{i \in K} \sum_{P^\delta \in \mathcal{P}: a \in P} \gamma_{P^\delta[\to v]} f_{P^\delta}^{sm,i}(\theta - \tau'_{P^\delta[\to v]}) \ .$$

If we send flow according to f_{sm} into the network with rounded transit times τ', we might violate capacity constraints. Using the definition of f^{sm} yields:

$$\begin{aligned} f_a^{sm}(\theta) &= \sum_{i \in K} \sum_{P^\delta \in \mathcal{P}: a \in P} \gamma_{P^\delta[\to v]} f_{P^\delta}^{sm,i}(\theta - \tau'_{P^\delta[\to v]}) \\ &= \sum_{i \in K} \sum_{P^\delta \in \mathcal{P}: a \in P} \gamma_{P^\delta[\to v]} \frac{1}{\varepsilon T} \int_{\theta - \varepsilon T}^{\theta} f_{P^\delta}^i(\xi - \tau'_{P^\delta[\to v]}) \, d\xi \\ &= \frac{1}{\varepsilon T} \sum_{i \in K} \sum_{P^\delta \in \mathcal{P}: a \in P} \gamma_{P^\delta[\to v]} \int_{\theta - \tau'_{P^\delta[\to v]} - \varepsilon T}^{\theta - \tau'_{P^\delta[\to v]}} f_{P^\delta}^i(\xi) \, d\xi \ . \end{aligned}$$

Now we use the that the length of a path with normal and rounded transit times differs by at most Δn:

$$\begin{aligned}
&\frac{1}{\varepsilon T}\sum_{i\in K}\sum_{P^\delta\in\mathcal{P}:a\in P}\gamma_{P^\delta[\to v]}\int_{\theta-\tau'_{P^\delta[\to v]}-\varepsilon T}^{\theta-\tau'_{P^\delta[\to v]}} f^i_{P^\delta}(\xi)\ d\xi\\
\le\ &\frac{1}{\varepsilon T}\sum_{i\in K}\sum_{P^\delta\in\mathcal{P}:a\in P}\gamma_{P^\delta[\to v]}\int_{\theta-\tau_{P^\delta[\to v]}-\Delta n-\varepsilon T}^{\theta-\tau_{P^\delta[\to v]}} f^i_{P^\delta}(\xi)\ d\xi\\
=\ &\frac{1}{\varepsilon T}\int_{\theta-\Delta n-\varepsilon T}^{\theta}\sum_{i\in K}\sum_{P^\delta\in\mathcal{P}:a\in P}\gamma_{P^\delta[\to v]} f^i_{P^\delta}(\xi-\tau_{P^\delta[\to v]})\ d\xi\\
=\ &\frac{1}{\varepsilon T}\int_{\theta-\Delta n-\varepsilon T}^{\theta} f_a(\xi)\ d\xi\ .
\end{aligned}$$

We know that the flow rate of f into a is bounded by the capacity of a:

$$\frac{1}{\varepsilon T}\int_{\theta-\Delta n-\varepsilon T}^{\theta} f_a(\xi)\ d\xi \le \frac{1}{\varepsilon T}(\Delta n+\varepsilon T)u_a = \left(1+\frac{\Delta n}{\varepsilon T}\right)u_a\ .$$

For our choice of $\Delta = \frac{\varepsilon^2 T}{n}$ this yields:

$$\left(1+\frac{\Delta n}{\varepsilon T}\right)u_a = \left(1+\frac{\frac{\varepsilon^2 T}{n}n}{\varepsilon T}\right)u_a = (1+\varepsilon)u_a\ .$$

Thus, f^{sm} violates the capacities by at most a factor of $(1+\varepsilon)$ if f^{sm} is send with the rounded transit times.

Solution, Part II: Scaled flows. Having the smoothed flow which violates capacities by at most a factor of $(1+\varepsilon)$ when sent with rounded transit times is good, but not perfect. In order to guarantee that capacity constraints are upheld, we *scale* the smoothed flow by a factor of $1/(1+\varepsilon)$. This guarantees that the capacity constraints are satisfied, but reduces the flow value by the same amount:

$$f^{sc} := \frac{1}{1+\varepsilon} f^{sm}\ .$$

Solution, Part III: Amplified flows. Now we have a scaled smoothed flow f^{sc} that does not violate the capacity constraints when sent with the rounded transit times, but it does not yet send enough flow. However, we can remedy this by amplifying – we do everything $(1+\varepsilon)$-times as long to send $(1+\varepsilon)$ more flow. We formalize this technique in the next lemma.

Lemma 5.4. *Let f be a generalized multi-commodity flow over time that fulfills some supplies and demands b in a time horizon of T with a cost at most C. Then for any $\lambda \ge 1$, there exists a generalized multi-commodity flow over time f' that fulfills supplies and demands λb in a time horizon of λT with a cost of at most λC.*

Proof. Let $N = (G, u, \tau, \gamma, c, b, T)$ be the instance in which f is sent. Then f corresponds to a static flow x in the time-expanded network $N^T = (G^T, u^T, \gamma^T, c^T, b^T)$. Notice that we can rescale time to make all transit times and the time horizon integer, if necessary.

Now consider an instance $\lambda N = (G, u, \lambda\tau, \gamma, c, b, \lambda T)$ where all transit times and the time horizon are multiplied by λ. Then, the λ-time-condensed network $\lambda N/\lambda$ of λN is identical to the time-expanded network N^T but all capacities are multiplied by λ.

Therefore λx is a feasible flow in $\lambda N/\lambda$ which fulfills supplies and demands λb in a time horizon of λT with a cost of at most λC. As we have seen in Lemma 5.2, λx corresponds to a flow over time with a time horizon of λT in λN, which can be seen as a relaxation of N since it has increased times. These increased transit times can be simulated by flow in N by waiting in nodes, which concludes our proof. □

Solution, Part IV: Putting the parts together. Let us summarize what we have done so far. We rounded transit times to be able to use time condensation, but were met with the problems of increased and contorted path lengths.

- We bounded the increases in path length due to rounding by the choice of Δ and restricting us to problems with optimal simple paths solutions. This yielded $\Delta n = \varepsilon^2 T$ as an upper bound for the increase in path length.
- We introduced smoothed flows as a way to make a flow more resistant to being sent with the rounded transit times instead of normal transit times. The smoothed flow violates the capacities by a factor of at most $(1+\varepsilon)$, at the cost of increasing the time horizon by an additional εT.
- In order to fix the capacity violation of the smoothed flow, we scaled it by a factor of $1/(1+\varepsilon)$. The resulting flow does not violate the capacities when sent with the rounded instead of normal transit times, but the scaling costs us a factor of $1/(1+\varepsilon)$ in flow sent.
- Finally, we showed the we can increase the time horizon by a factor of $(1+\varepsilon)$ to send a factor of $(1+\varepsilon)$ more flow.

Together, we can use these techniques to transform a solution in the original network into a similar solution with rounded transit times, and the other way round, although we have not talked about this yet. In the next section, we will formalize these claims and use them to design an FPTAS for the generalized multi-commodity flow problem with lossy gains.

5.1.2 An FPTAS for Generalized Maximum Flows over Time

Here, we are considering the generalized maximum (multi-commodity) flow over time problem with lossy gains, i. e., $\gamma_a \leq 1$ for all arcs a. This means that flow is never gained but only lost when travelling through the network. This restriction guarantees that cycles are not necessary for optimal solutions, allowing us to apply time condensation. Notice that networks without flow-generating cycles can be turned into networks with lossy gains by scaling techniques, see e.g., Wayne [117]. Thus, the results discussed in this section hold for all networks without flow-generating cycles. A consequence of the absence of flow-generating cycles is that in such networks with lossy gains, only paths increase the flow value.

We will study the multi-commodity variant of this problem here, as it can be solved by the same technique as the single commodity problem. In the following, we will speak of supplies and demands b for the problem. Usually, we these supplies and demands are infinite for the generalized maximum (multi-commodity) flow over time problem, but the argumentation holds for finite supplies and demands as well. We could also add costs and a cost bound to this problem, as we have seen that smoothing, scaling and amplifying do not affect costs too much.

We will now describe an FPTAS that takes an instance of the generalized maximum multi-commodity flow over time problem $N = (G, u, \tau, \gamma, b, T)$ and a precision $\varepsilon > 0$ as its input and computes a generalized multi-commodity flow over time f with a time horizon of $(1 + O(\varepsilon))T$ that fulfills as much supplies and demands from b as possible. It works as follows:

1. Set the time condensation factor $\Delta := \varepsilon^2 T/n$, and set an extended time horizon of $T' := \lceil (1 + \varepsilon^3)T/\Delta \rceil \Delta$ to use with the time condensation.
2. Create the Δ-time-condensed network $N^{T'}/\Delta$ and solve the corresponding static generalized multi-commodity maximum flow problem in it using a linear program formulation. Let x be an optimal solution of this linear program formulation.
3. Transform x, which is a static flow in $N^{T'}/\Delta$, into a flow over time f in N with time horizon $(1 + \varepsilon)T'$ that has at least as much flow value than $|x|$.

If storage at intermediate nodes is not forbidden, it is sufficient to compute a flow x fulfilling supplies and demands b in Step 2, as we can simulate the longer rounded transit times by waiting in the original network. This simulation replaces the work for the transformation of x in Step 3, where we lose a factor of $1/(1 + \varepsilon)$ of flow sent during the transformation. In order to show the correctness of the FPTAS, we need to prove two claims.

1. Let f be a generalized multi-commodity flow over time in N that fulfills supplies and demands b within a time horizon of T. Then there exists a static generalized multi-commodity flow x in $N^{T'}/\Delta$ that fulfills supplies and demands $(1 + \varepsilon)b$ for $T' := \lceil (1 + \varepsilon)^3 T/\Delta \rceil \Delta$ and $\Delta := \varepsilon^2 T/n$.
2. Let x be static generalized multi-commodity flow in the Δ-condensed time-expanded network $N^{T'}/\Delta$ that fulfills supplies and demands $(1 + \varepsilon)b$. Then we can construct a generalized multi-commodity flow f in N that fulfills supplies and demands b within a time horizon of $(1 + \epsilon)T'$.

The first claim guarantees us for every flow over time in N the existence of a static flow in the time-condensed network $N^{T'}/\Delta$ that has a time horizon of T and sends $(1 + \varepsilon)$

more flow. The second claim guarantees us that we can transform a flow from the time-condensed network $N^{T'}/\Delta$ back into a flow over time in the original network N with a loss of a factor of at most $1/(1+\varepsilon)$. Together, they guarantee us that we will be able to find a solution at least as good as the optimal solution within a time horizon of $(1+O(\varepsilon))T$.

We begin with proving the first claim, for which we have laid the groundwork by introducing the techniques of smoothing, scaling and amplifying flows over time.

Lemma 5.5. *Let f be a generalized multi-commodity flow over time in N that fulfills supplies and demands b within a time horizon of T. Then there exists a static generalized multi-commodity flow x in $N^{T'}/\Delta$ that fulfills supplies and demands $(1+\varepsilon)b$ for $T' := \lceil (1+\varepsilon)^3 T/\Delta \rceil \Delta$ and $\Delta := \varepsilon^2 T/n$.*

Proof. According to Lemma 5.2 it is sufficient to show that there exists a flow over time f' in the network N with the rounded transit times τ' satisfying supplies and demands $(1+\varepsilon)b$ within a time horizon of T'. This is because the rounded transit times are multiples of Δ, and T' is a multiple of Δ as well.

We know due to Lemma 5.4 that we can amplify the flow over time f to fulfill the supplies and demands $(1+\varepsilon)^2 b$ within a time horizon of $(1+\varepsilon)^2 T$. We call the amplified flow over time f^{amp}. Now we apply smoothing as discussed above to f^{amp}, resulting in a flow over time f^{sm+amp} which completes within a time horizon of $(1+\varepsilon)^2 T + \varepsilon T$ and violates capacities by a factor of at most $(1+\varepsilon)$ if send with the transit times τ' in N. f^{sm+amp} also fulfills the supplies and demands $(1+\varepsilon)^2 b$.

Thus, scaling f^{sm+amp} by a factor of $1/(1+\varepsilon)$ yields a flow over time $f^{sc+sm+amp}$ that does not violate capacities when sent in N with the rounded transit times τ'. $f^{sc+sm+amp}$ requires a time horizon of $(1+\varepsilon)^2 T + \varepsilon T + \varepsilon^2 T$, with the additional $\varepsilon^2 T$ stemming from the fact that path lengths can increase by at most $\varepsilon^2 T$ when using the rounded transit times instead of the normal ones. Due to the scaling, $f^{sc+sm+amp}$ fulfills the supplies and demands $(1+\varepsilon)b$ as we required. Our proof is completed by the fact that $(1+\epsilon)^2 T + \epsilon^2 T + \epsilon T \leq (1+\epsilon^3)T \leq T'$. □

For the first claim, we had to send a flow computed using normal transit times with rounded transit times. For the second claim, we have to do the reverse: take a flow computed using rounded transit times and send it with normal transit times instead. If we are allowed to let flow wait, this is easy – we simulate the longer rounded transit times by waiting in nodes. But, even if we are not allowed to use waiting for this, we can apply smoothing and scaling again.

Lemma 5.6. *Let x be a static generalized multi-commodity flow in the Δ-condensed time-expanded network $N^{T'}/\Delta$ that fulfills supplies and demands $(1+\varepsilon)b$. Then we can construct a generalized multi-commodity flow f in N that fulfills supplies and demands b within a time horizon of $(1+\epsilon)T'$.*

Proof. Given such a static generalized multi-commodity flow x in $N^{T'}/\Delta$, we can obtain a flow over time f from it in the network N with rounded transit times τ' and a time horizon of T'. We can try to send f with the original transit times, but we would run into the previously discussed problems. Luckily, the techniques of smoothing and scaling flow can be applied here as well to solve our problem.

Consider a decomposition of f into flows over time on paths with delays $P^\delta \in \mathcal{P}$ with flow values f_{P^δ}. We define a smoothed and scaled flow by

$$f_{P^\delta}^{sc+sm,i}(\theta) = \frac{1}{1+\varepsilon}\frac{1}{\varepsilon T'}\int_{\theta-\varepsilon T'}^{\theta} f_{P^\delta}^i(\xi)\;d\xi \quad \text{for all } i \in K, P^\delta \in \mathcal{P}, \theta \in [0,(1+\varepsilon)T')$$

that completes in a time horizon of $(1+\varepsilon)T'$. Above we checked what happens if we send a flow computed using normal transit times with rounded transit times instead. Here, we need to do the reverse thing – f was computed using rounded transit times and will be sent with the original transit times.

The actual proof to show that f^{sc+sm} does not violate capacities using transit times τ is relatively similar to what we saw above. Recall that path lengths can be up to $\varepsilon^2 T'$ shorter with regard to the normal transit times τ instead of the rounded transit times τ'. The total flow rate into an arc a at time θ is then:

$$\begin{aligned}
f_a^{sc+sm}(\theta) &= \sum_{i\in K}\sum_{P^\delta\in\mathcal{P}:a\in P} \gamma_{P^\delta[\to v]} f_{P^\delta}^{sc+sm,i}(\theta - \tau_{P^\delta[\to v]}) \\
&= \frac{1}{1+\varepsilon}\frac{1}{\varepsilon T'}\sum_{i\in K}\sum_{P^\delta\in\mathcal{P}:a\in P} \gamma_{P^\delta[\to v]}\int_{\theta-\tau_{P^\delta[\to v]}-\varepsilon T'}^{\theta-\tau_{P^\delta[\to v]}} f_{P^\delta}^i(\xi)\;d\xi \\
&\le \frac{1}{1+\varepsilon}\frac{1}{\varepsilon T'}\sum_{i\in K}\sum_{P^\delta\in\mathcal{P}:a\in P} \gamma_{P^\delta[\to v]}\int_{\theta-\tau'_{P^\delta[\to v]}-\varepsilon T'}^{\theta-\tau'_{P^\delta[\to v]}+\varepsilon^2 T'} f_{P^\delta}^i(\xi)\;d\xi \\
&= \frac{1}{1+\varepsilon}\frac{1}{\varepsilon T'}\int_{\theta-\varepsilon T'}^{\theta+\varepsilon^2 T'} f_a(\xi)\;d\xi \le u_a \;.
\end{aligned}$$

This proves our claim. □

Runtime of the FPTAS. The last part missing is to analyze the runtime of the FPTAS and check that it is polynomial in the input size and ε^{-1}. The Δ-condensed time-expanded network has $O(n\varepsilon^{-2})$ time layers and its size is therefore polynomial in the input size and ε^{-1}. Thus, we can solve the linear program formulation in Step 1 in polynomial time. In order to execute Step 3 in polynomial time as well, we need to ensure that f has a polynomial-size encoding.

Lemma 5.7. *A path-decomposition of the flow x computed in Step 2 into flows on j paths in $N^{T'}/\Delta$ can be transformed into a path-decomposition $(f_P)_{P\in\mathcal{P}}$ on j paths with delays in N such that the time interval $[0,T')$ can be partitioned into $T'/\Delta = O(n\varepsilon^{-2})$ subintervals where f_P is constant.*

Proof. Consider a path-decomposition of x into flow on paths $P \in \mathcal{P}$ in $N^{T'}/\Delta$ with flow values x_P^i, $P \in \mathcal{P}, i \in K$. Such a path P will use storage at a source, then leave the source and continue onwards to a sink. We can assume here that the path does not contain a later copy of the source, since then we would have a cycle in N which could be replaced by storage in a source. Furthermore, we can assume that the path ends in a sink, because ending in flow-absorbing cycle does not help with the objective.

Any path $P \in \mathcal{P}$ induces therefore a path with delays P^δ in N with a flow rate function $f^i_{P^\delta} : [0, T') \to \mathbb{R}^+, i \in K$ that is 0 except for an interval $[\Delta\xi, \Delta(\xi+1))$ for some $\xi \in \{0, \ldots, T'/\Delta - 1\}$, where it is x^i_P/Δ. Multiple paths in $N^{T'}/\Delta$ can correspond to the same path in N, possibly resulting in a flow function for a path P^δ that is piecewise constant in an interval $[\Delta\xi, \Delta(\xi+1))$ for some $\xi \in \{0, \ldots, T'/\Delta - 1\}$. □

Therefore the $f^{sc+sm,i}_{P^\delta}, i \in K$ have at most $O(n\varepsilon^{-2})$ breakpoints as well and are piecewise linear due to

$$f^{sc+sm,i}_{P^\delta}(\theta) = \frac{1}{1+\varepsilon}\frac{1}{\varepsilon T'}\int_{\theta-\varepsilon T'}^{\theta} f^i_{P^\delta}(\xi)\ d\xi$$

and can be computed efficiently. Lemma 5.5 guarantees us that for every solution in the original graph, we can find a solution of equivalent value in time-condensed graph, which has an implicit time horizon that is longer by a factor of $(1+\varepsilon)^3$ than the original time horizon. Solutions in the time-condensed graph on the other hand have a solution of equivalent value in the original graph due to Lemma 5.6 that again has a time horizon which is larger by a factor of $(1+\varepsilon)$. Together, this gives an increase of the time horizon by a factor of $(1+\varepsilon)^4$. However, for $\varepsilon \to 0$ we can find for any $\varepsilon' > 0$ an $\varepsilon > 0$ such that $(1+\varepsilon)^4 \leq 1 + \varepsilon'$. This yields the following theorem.

Theorem 5.8. *Given an instance of the generalized maximum multi-commodity flow over time problem $N = (G, u, \tau, \gamma, b, T)$ and a precision $\varepsilon > 0$, we can compute a generalized multi-commodity flow over time f with a time horizon of $(1+\varepsilon)T$ that fulfills at least as much supplies and demands from b as possible within a time horizon of T.*

Problems with time condensation and arbitrary gains. Unfortunately, it is not clear whether Theorem 5.8 can be generalized to the case of arbitrary gain factors on the arcs. The main problem is that the assumption of simple flow paths no longer holds in the general setting. Taking a closer look at the analysis in the proof of Theorem 5.8, it suffices to bound the number of times a flow particle visits a node by a polynomial in the input size. But even this relaxed condition does in general not hold for arbitrary gain factors, as can be seen from the example depicted in Figure 5.5. We assume unit transit times and gains, unless specified otherwise. Then in a generalized maximum flow over time each flow particle has to use the cycle as many times as possible to generate as much flow as possible behind the bottleneck edge leaving s. Note, however, that this cycle cannot be used to generate flow from scratch as its transit time is not zero – i. e., we can amplify flow by spending time in the cycle, but we cannot generate flow from scratch.

5.2 Geometric Time Condensation

In Chapter 4.2 we described an algorithm that computes a 2-value-approximative earliest arrival flow in arbitrary networks. We also know that there are instances where no $\beta < 2$-

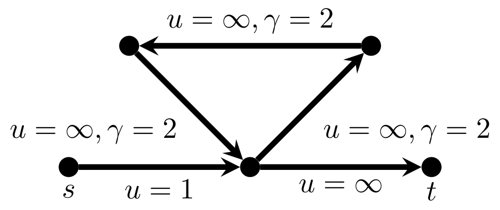


Figure 5.5: An instance where flow visits nodes multiple times. The exact number depends on the time horizon and the transit time of the cycle and can be pseudo-polynomial.

value-approximations exist (Groß, Kappmeier, Schmidt and Schmidt [50]). However, for a specific instance, better value-approximations might exist. The same is true for time approximations.

In this section, we will study the use of time expansion and condensation to approximate the best possible approximation factor. The results presented here have first been published in [50]. Our goal here is the following: if an instance permits a β-value-approximative earliest arrival flow, we want to be able to compute it, or an approximation of it. An approximation in this case would for example be a $(1+\varepsilon)$-time- and $(1+\varepsilon)\beta$-value-approximative earliest arrival flow, for some $\varepsilon > 0$. Similarly, we want to compute an α-time- or at least $(1+\varepsilon)\alpha$-time-approximative earliest arrival flow, if an instance permits a α-time-approximative earliest arrival flow.

Theorem 5.9. *An α-time-approximate earliest arrival flow or a β-value-approximate earliest arrival flow can be computed using an LP of pseudo-polynomial size, where α and β are the best time- and value-approximations possible for the given instance. This assumes the discrete time model.*

Proof. We can use the following LP to compute the best possible value-approximation. G is the graph we are computing the value-approximate earliest arrival flow in, T is the time horizon and $\mathbb{T} := \{0, \dots, T-1\}$ is the set of all discrete time steps up to the time horizon.

$$\begin{array}{llll}
\max & \beta', & & \\
\text{s.t.} & \sum_{a \in \delta_v^+} \sum_{\xi=0}^{\theta} x_{a,\xi} - \sum_{a \in \delta_v^-} \sum_{\xi=0}^{\theta-\tau_a} x_{a,\xi} & \leq \max\{0, b_v\} & \text{for all } v \in V(G), \theta \in \mathbb{T}, \\
& \sum_{a \in \delta_v^+} \sum_{\xi=0}^{\theta} x_{a,\xi} - \sum_{a \in \delta_v^-} \sum_{\xi=0}^{\theta-\tau_a} x_{a,\xi} & \geq \min\{0, b_v\} & \text{for all } v \in V(G), \theta \in \mathbb{T}, \\
& x_{a,\theta} & \leq u_a & \text{for all } a \in A(G), \theta \in \mathbb{T}, \\
& \sum_{v \in V(G): b_v < 0} \sum_{a \in \delta_v^-} \sum_{\xi=0}^{\theta-\tau_a} x_{a,\xi} & \geq \beta' p(\theta) & \text{for all } \theta \in \mathbb{T}, \\
& x_{a,\theta} & \geq 0 & \text{for all } a \in A(G), \theta \in \mathbb{T}.
\end{array}$$

In this LP, $\beta' = 1/\beta$ is the inverse of the value-approximation guarantee β, which we are looking for. $p(\theta)$, $\theta \in \mathbb{T}$ is the value of the earliest arrival pattern at time θ. The solutions of this LP are by construction feasible flows that are $1/\beta'$-value-approximations. Since every feasible flow in the discrete time model can be transferred into a solution of this LP, we obtain the best possible approximation factor by solving this LP.

Notice that in order to obtain a time-approximation, we can adjust the above LP by replacing $\beta' p(\theta)$ by $p(\lfloor \theta/\alpha \rfloor)$ in the fourth type of inequalities and by replacing $\max \beta'$ by $\min \alpha$ in the objective function. □

We cannot reduce the size of the above LP easily via time condensation, because earliest arrival flows give guarantees about the flow values at all points in time. Other flow problems usually ask for a flow that fulfills certain properties at a specific point in time, namely the time horizon. The previously introduced time condensation was built around the fact that we wanted to give a good time-approximation guarantee at the time horizon T. There, an error of εT is just a factor of $(1+\varepsilon)$ more. However, at time 1, this error is a factor of εT more.

Therefore we need a time-condensed network that is more precise near time 0 and less precise near time T. Such a time-condensed network was introduced by Fleischer and Skutella [32], and is referred to as *geometrically condensed time-expanded network*, because the size of the time-intervals increases geometrically. For a directed graph G, they define an *L-time-expanded network* N^L for a sorted list of times $L = \langle 0 = \theta_0 < \theta_1 < \cdots < \theta_r = T \rangle$ by creating r copies of the node set $V(G)$. Let v_q be the q-th copy of a node v. Then arcs $a = (v, w)$ are inserted into the node copies by adding a copy a_q of a between every v_q and $w_{q'_a}$ with $\theta_q + \tau_a \leq \theta_{r-1}$ and $q'_a := \min \left\{ q'' \mid \theta_q + \tau_a \leq \theta_{q''} \right\}$. The capacity of a_q is $u_{a_q} := u_a(\theta_{q+1} - \theta_q)$. Essentially, the classic time-expanded network is a special case of an L-time-condensed network with $L = \langle 0, 1, \ldots, T \rangle$. The more general L-time-condensed networks just omit some time layers and adjust the arcs and their capacities correspondingly. Geometrically time-condensed networks are L-time-condensed networks with

$$\begin{aligned} L := &\langle 0, 1, 2, \ldots, 2p, \\ &2(p+1), 2(p+2), \ldots, 2 \cdot 2p, \\ &4(p+1), 4(p+2), \ldots, 4 \cdot 2p, \\ &\ldots \\ &2^{\ell-1}(p+1), 2^{\ell-1}(p+2), \ldots, 2^{\ell} p \rangle \ . \end{aligned}$$

Here, $p := \lceil 2n\varepsilon^{-2} \rceil$ is chosen depending on the desired accuracy $\varepsilon > 0$, and $\ell := \min \left\{ \ell' \mid 2T \leq 2^{\ell'} p \right\}$. T is here the time horizon of an optimal flow over time. Notice that the cardinality of $L \in O(n\varepsilon^{-2} \log T)$ is polynomial in the input size and ε^{-1}. Fleischer and Skutella [32] managed to show several results for such L-time-expanded networks, which essentially generalize time-condensation to L-time-expanded networks. The next two lemmas are similar to Lemma 5.5 and 5.6 – they guarantee us for every solution a similar solution in the L-time expansion, and that solutions in the L-time expansion can be transferred back to the original network.

Lemma 5.10 (Lemma 5.2 in [32])**.** *Let $L = \langle 0 = \theta_0, \dots, \theta_r\rangle$ with $\theta_q - \theta_{q-1} \leq \theta_{q+1} - \theta_q$ for all $q = 1, \dots, r-1$. Then any static flow in N^L corresponds to a flow over time in N such that the amount of flow reaching t in N by time θ_{q+1}, $q = 0, \dots, r-1$ is the same as the amount of the flow reaching node t_q in N^L.*

Lemma 5.11 (Lemma 5.3 in [32])**.** *Let $L = \langle 0 = \theta_0, \dots, \theta_r\rangle$ with $\theta_q - \theta_{q-1} \leq \theta_{q+1} - \theta_q$ for all $q = 1, \dots, r-1$. Let f be any flow over time that completes ny time θ_r in the network Nmodified such that all transit times are increased by $\Delta := \theta_r - \theta_{r-1}$. Then f induces a feasible static flow in N^L of the same value. Then any static flow in N^L corresponds to a flow over time in N such that the amount of flow reaching t in N by time θ_{q+1}, $q = 0, \dots, r-1$ is the same as the amount of the flow reaching node t_q in N^L.*

Besides the problem that the relative error becomes larger the closer we get to zero, another problem is that in time-expanded networks, the flow rate is averaged between two time layers. This can introduce an error as large as the difference between two time layers. If we assume that our transit times are integer, which we can guarantee by scaling, we have path lengths of 0, 1, and so on. If we discretize time into steps of length ε the error by averaging is no problem for flow which arrives using paths of length at least 1 because it takes then at most a factor of $(1 + \varepsilon)$ longer. We can then either assume that we have no zero-transit times, as they usually do not occur in evacuation settings, or use techniques for zero-transit time earliest arrival flows to handle them in a preprocessing step (see for example Hajek and Ogier [54]). A detailed discussion of choosing a good time discretization can be found in Fleischer and Skutella [32, Lemma 5.1].

The combination of Theorem 5.9 and geometrically condensed time-expanded networks yields the following theorem.

Theorem 5.12. *For arbitrary $\varepsilon > 0$ with $1/\varepsilon$ integral, a $(1 + O(\varepsilon))\alpha$-time- and $(1 + \varepsilon)$-value-approximate earliest arrival flow or a $(1 + O(\varepsilon))$-time- and $(1 + \varepsilon)\beta$-value-approximate earliest arrival flow can be computed in running time polynomial in the input size and ε^{-1}, where α/β are the best time/value-approximations possible for the given instance. This holds in the discrete and continuous time model.*

Proof. We use the LP described in the proof of Theorem 5.9 and base it on a geometrically condensed time-expanded network instead of a time-expanded network. We use the same geometrically condensed time-expanded networks as described above (see also [32, Section 5.3]) and assume a sufficiently fine discretization of time as we discussed before.

This gives us a condensed time-expanded network N^L with time layers that are geometrically spread over $[0, T)$, with more layers near 0 and less layers near T. The geometrical spread ensures that the resulting network has the necessary precision for an FPTAS, but is still polynomial in the input size and ε^{-1}. The resulting network has the following form, with $\theta_0, \dots, \theta_r$ denoting the time points at which the time layers are placed.

$$\begin{array}{lllll}
\max & \beta', & & & \\
\text{s.t.} & \sum\limits_{a\in\delta_v^+} x_a - \sum\limits_{a\in\delta_v^-} x_a & \leq & \max\{0, b_v\} & \text{for all } v \in V(N^L), \\
& \sum\limits_{a\in\delta_v^+} x_a - \sum\limits_{a\in\delta_v^-} x_a & \geq & \min\{0, b_v\} & \text{for all } v \in V(N^L), \\
& x_{a_i} & \leq & u_a(\theta_{i+1} - \theta_i) & \text{for all } a_i \in A(N^L), \\
& |x(\theta)| & \geq & \beta' p(\theta) & \text{for all } \theta \in L, \\
& x_a & \geq & 0 & \text{for all } a \in A(N^L).
\end{array}$$

The $|x(\theta)|$ in the LP refers to the amount of flow of x that has arrived at the sinks until time θ. By construction, an optimal solution (β', x) to this LP contains a static flow x in the L-time-expanded network that obeys supplies and demands, flow conservation and capacities. Furthermore, it offers the best value-approximation $\beta = 1/\beta'$ possible at all time points $\theta \in L$.

The approximation guarantee follows then by the same argumentation as in [32] – we can use Lemma 5.10 and 5.11 similar to Lemma 5.5 and 5.6 in order to guarantee the solutions carry over between the original and L-time-expanded network. This yields the same $1 + O(\varepsilon)$-time-approximation guarantee as in [32]. Notice that we can exchange the $|x(\theta)| \geq \beta' p(\theta)$, $\theta \in L$ constraint by $|x(\alpha\theta)| \geq p(\theta)$, $\theta \in L$ in order to obtain a time-approximation. □

5.3 Sequence Condensation

The time condensation techniques we studied so far have the drawback that they can only be applied to problems having optimal solutions which use only simple paths. Prominent classes of problems that do not fall into this category include multi-commodity flows over time without intermediate storage, minimum cost flows over time, and generalized maximum flows over time.

In this chapter, we present a more elaborate condensation technique for time-expanded networks whose analysis no longer requires that flow is being sent along simple paths only. Normal time condensation uses the number of arcs on a path to bound the rounding error, which does not yield good bounds for non-simple paths due to the potentially high number of arcs in them. To this end, we will introduce a new type of LP formulation for flow over time problems that is somewhere in between an arc-based and a path-based formulation. We call this a *sequence-based formulation*, and the technique of applying time condensation to it *sequence condensation*. The sequence LP has a dual which can be approximately separated in polynomial time for most flow problems, or more precisely, for flow problems that use standard flow conservation.

We start by studying the separation problem of the dual of the sequence LP from a general point of view in Chapter 5.3.1 – after all, if we cannot handle the separation problem well enough, our LP formulation does not help us. It turns out that while the separation problem is NP-hard, we describe an FPTAS for it, which is sufficient for our purposes. This FPTAS is based on dynamic programming in conjunction with Meggido's parametric search framework [77].

After that, we will describe sequence condensation using the example of the maximum multi-commodity flow over time problem without intermediate storage in Chapter 5.3.2 and show that it yields an FPTAS for this NP-hard problem. Finally, we discuss how sequence condensation can be extended to minimum cost multi-commodity flow over time problems in Chapter 5.3.3. The results in this chapter have first been published in Groß and Skutella [52].

5.3.1 The Dual Separation Problem

In this section, we define the dual separation problem, which we call the *restricted minimum cost s-t-sequence problem.* This problem consists of finding an arc sequence of minimum cost between two nodes whose length is restricted to lie in a given interval. This length constraint implies that it can be necessary to incorporate cycles in an optimal solution. We present an FPTAS for this NP-hard problem. In Chapter 5.3.2 we see that this problem can be used to separate the dual of sequence LPs, resulting in solutions to the primal problems with approximated feasibility.

Definition 5.13. *An instance of the* restricted minimum cost s-t-sequence problem *is given by a directed graph G with costs $c_a \geq 0$ and lengths $\ell_a \geq 0$ on the arcs $a \in A(G)$, two designated nodes $s, t \in V(G)$ and a threshold value $\Delta > \ell^*$, with $\ell^* := \max_{a \in A(G)} \ell_a$. The task is to find an s-t-sequence S of minimum cost $c_S := \sum_{a \in S} \#(a, S) \cdot c_a$ under the constraint that $\Delta - \ell^* \leq \ell_S \leq \Delta$.*

This problem is NP-hard, similar to related resource-constrained shortest path problems (e. g., [58]). The FPTAS that we will develop for this problem approximates the length constraint, i. e., it computes a solution at least as good as an optimal solution but with a relaxed length constraint $(1 - \varepsilon)\Delta - \ell^* \leq \ell_S \leq (1 + \varepsilon)\Delta$ for a given $\varepsilon > 0$. We begin by classifying the arcs into two groups.

Definition 5.14. *Given $\varepsilon > 0$, an arc $a \in A(G)$ is* short *if $\ell_a \leq \varepsilon\Delta/n$, and* long *otherwise. A sequence is short, if all of its arcs are short, and long otherwise.*

It follows that a solution to the restricted minimum cost s-t-sequence problem can – due to the length constraint – contain at most n/ε long arcs. Since a long cycle needs to contain at least one long arc, it follows that at most n^2/ε arcs of the solution can be part of long cycles; the remaining arcs are part of an s-t-path or short cycles. Due to the fact that a path does not contain duplicate nodes, we make the following observation.

Observation 5.15. *Let S be a solution to the restricted minimum cost s-t-sequence problem. Then S contains at most $n + n^2/\varepsilon$ arcs that are not part of short cycles.*

For our FPTAS, we would like to round the arc lengths without introducing too much error. Sequences without short cycles are fine, because they have a nicely bounded number of arcs. However, we cannot simply restrict ourselves to sequences without short cycles. Therefore, we now define a *nice v-w-sequence S* to be a v-w-sequence that contains at most one short cycle, but is allowed to make several passes of this cycle. We will see that we can handle such sequences efficiently while losing not too much accuracy. Notice that we can ignore zero-length cycles, as these cycles do neither help us reach t nor can they improve the objective value.

Lemma 5.16. *For an s-t-sequence S with a path-cycle-decomposition, let C be a short cycle in the decomposition of S with maximal length-to-cost ratio $\frac{\sum_{a\in C}\ell_a}{\sum_{a\in C}c_a}$ and let ℓ_{short} be the total length of all short cycles in the decomposition of S. (Zero-cost cycles are assumed to have infinite ratio.) We define a corresponding nice sequence S' with cycle C by removing all short cycles and adding $\left\lfloor \frac{\ell_{short}}{\ell_C} \right\rfloor$ copies of C. Then $\ell_S - \varepsilon\Delta \le \ell_S - \ell_C \le \ell_{S'} \le \ell_S$ and $c_{S'} \le c_S$.*

Proof. Let S^* be the sequence obtained by removing all short cycles from S. S^* is then a sequence without any short cycles and length $\ell_{S^*} = \ell_S - \ell_{short}$. Now we add $\left\lfloor \frac{\ell_{short}}{\ell_C} \right\rfloor$ copies of C and gain S'. It follows that

$$\begin{aligned}
\ell_{S'} &= \ell_{S^*} + \left\lfloor \frac{\ell_{short}}{\ell_C} \right\rfloor \ell_C = \ell_S - \ell_{short} + \left\lfloor \frac{\ell_{short}}{\ell_C} \right\rfloor \ell_C \\
&\le \ell_S - \ell_{short} + \frac{\ell_{short}}{\ell_C}\ell_C \\
&= \ell_S \ .
\end{aligned}$$

Since C is a short cycle, $\ell_C \le n \cdot \varepsilon \frac{\Delta}{n} = \varepsilon\Delta$ holds, yielding

$$\begin{aligned}
\ell_{S'} &= \ell_S - \ell_{short} + \left\lfloor \frac{\ell_{short}}{\ell_C} \right\rfloor \ell_C \\
&\ge \ell_S - \ell_{short} + \frac{\ell_{short}}{\ell_C}\ell_C - \ell_C \\
&= \ell_S - \ell_C \\
&\ge \ell_S - \varepsilon\Delta \ .
\end{aligned}$$

Now let $C_1, \dots, C_k$ be the removed short cycles. Due to C having maximum length-to-cost-ratio and therefore minimum cost-to-length-ratio, it follows that the cost of S' is bounded by

$$\begin{aligned}
c_{S'} &= c_S - \sum_{i=1}^{k} \ell_{C_i}\frac{c_{C_i}}{\ell_{C_i}} + \left\lfloor \frac{\ell_{short}}{\ell_C} \right\rfloor c_C \\
&\le c_S - \sum_{i=1}^{k} \ell_{C_i}\frac{c_C}{\ell_C} + \ell_{short}\frac{c_C}{\ell_C} \\
&= c_S \ .
\end{aligned}$$

□

Recall that we are looking for an s-t-sequence of lower or equal cost than the cost of an optimal solution, but with relaxed length constraints. Based on our observations, each optimal solution to the problem yields a corresponding nice sequence of lower or equal cost in the length interval between $\Delta - \ell^* - \varepsilon\Delta$ and Δ. Thus, it is sufficient to describe an algorithm that computes an optimal solution in the set of nice sequences within this length interval. In the following, we describe an algorithm that does so, but with a slightly larger set of allowed sequences. This can only improve the objective, however.

Computing the best-ratio-cycles for each node. For each node $v \in V(G)$, we compute the best-ratio short cycle containing v and refer to it as C_v. We can compute the cycle with the best ratio by using Megiddo's framework [77].

Defining rounded lengths. In order to compute the rest of the sequence, we will now introduce rounded lengths. Define $\delta := \varepsilon^2\Delta/n^2$ and rounded arc lengths ℓ' by $\ell'_a := \lfloor \ell_a/\delta \rfloor \delta$ for all $a \in A(G)$.

Computing the optimal sequences with regard to rounded lengths for each node. We use the rounded arc lengths in a dynamic program. Let $S(v,w,x,y)$ be a v-w-sequence of minimum cost with length x (with regard to ℓ') that contains y arcs. We denote the cost of $S(v,w,x,y)$ by $c(v,w,x,y)$. Notice that storing these values requires only $O(n \cdot n \cdot n^2\varepsilon^{-2} \cdot (n + n^2\varepsilon^{-1}))$ space, as there are only n nodes, $n^2\varepsilon^{-2}$ different lengths and at most $n + n^2\varepsilon^{-1}$ arcs in the sequences we are looking for (see Observation 5.15). We initialize the dynamic program with $c(v,w,x,y) := 0$ for $v = w$, $x = 0$, $y = 0$ and $c(v,w,x,y) := \infty$ for either $x = 0$ or $y = 0$. For $x \in \{0, \delta, \dots, \lfloor n^2/\varepsilon^2 \rfloor \delta\}$ and $y \in \{0, 1, \dots, \lfloor n + n^2\varepsilon^{-1} \rfloor\}$ we compute

$$c(v,w,x,y) := \min \left\{ c(v,u,x-\ell'_a, y-1) + c_a \;\middle|\; a = (u,w) \in A(G), x - \ell'_a \geq 0 \right\} .$$

We now define $C(v,w,x) := \min\left\{ c(v,w,x,y) \mid y \in \{0,1,\dots,\lfloor n + n^2\varepsilon^{-1} \rfloor\}\right\}$ as the minimum cost of a v-w-sequence with length x (with regard to ℓ') and at most y arcs. The sequences corresponding to the costs can be computed with appropriate bookkeeping. This dynamic program will either compute a sequence that does not use short cycles or at least a sequence that does not use more arcs than a nice sequence potentially would – which is also fine for our purposes.

Bounding the rounding error. The rounding error for an arc a is bounded by δ. Our sequences have at most $n + n^2/\varepsilon$ arcs. Therefore the rounding error for such a sequence S is at most $(n + n^2/\varepsilon)\delta = (\varepsilon + \varepsilon^2/n)\Delta$. Therefore we can bound the length of S with regard to the rounded lengths by $\ell_S - (\varepsilon + \varepsilon^2/n)\Delta \leq \ell'_S \leq \ell_S$.

Putting the parts together. For this computation, we test all possible start-nodes v for short cycles and all ways to distribute length between the s-v-, v-t-sequences and the short cycles. There are at most $O((n^2\varepsilon^{-2})^2) = O(n^4\varepsilon^{-4})$ ways to distribute the length. If the s-v- and v-t-sequence do not have sufficient length to match the length constraint, the remaining length is achieved by using the best-ratio cycle at v. We define $\mathbb{L} := \{0, \delta, \dots, \lfloor n^2/\varepsilon^2 \rfloor \delta\}$ for brevity. Since we compute s-v- and v-t-sequences with regard to ℓ', we incur a rounding error of $2(\varepsilon + \varepsilon^2/n)$ there. Thus, we have to look for sequences with a length of at least $L := (1 - 3\varepsilon - 2\varepsilon^2/n)\Delta - \ell^*$ with regard to ℓ' in order to consider all sequences with a length of at least $\Delta - \ell^* - \varepsilon\Delta$ with regard to ℓ. We can compute an s-t-sequence S with cost at most the cost of an optimal solution by

$$\min \left\{ C(s, v, \ell'_s) + c_{C_v} \left\lfloor \frac{L - \ell'_s - \ell'_t}{\ell_{C_v}} \right\rfloor + C(v, t, \ell'_t) \,\middle|\, v \in V(G), \ell'_s, \ell'_t \in \mathbb{L}, \ell'_s + \ell'_t \leq L \right\}$$

and using bookkeeping. The minimum can be computed in time $O(n^5\varepsilon^{-4})$. Thus, we can find a sequence S with cost at most the cost of an optimal solution and

$$\left(1 - 3\varepsilon - 2\varepsilon^2/n\right) \Delta - \ell^* \leq \ell_S \leq \left(1 + 2\varepsilon + 2\varepsilon^2/n\right) \Delta \ .$$

Theorem 5.17. *For a given instance I of the restricted minimum cost s-t-sequence problem and $\varepsilon > 0$, we can compute in time polynomial in the input size and ε^{-1} an arc sequence whose cost is at most the cost of an optimal solution to I and whose length satisfies $(1 - \varepsilon)\Delta - \ell^* \leq \ell_S \leq (1 + \varepsilon)\Delta$.*

5.3.2 The Maximum Multi-Commodity Flow over Time Problem

We will now use sequence condensation to obtain an FPTAS to the maximum multi-commodity flows over time without intermediate storage. We study this problem for the sake of simplicity, as this problem has no costs, and we assume that all supplies and demands are infinite. We will see in the next chapter that costs or finite supplies and demands can be handled as well, but they increase the notational complexity somewhat.

LP-Formulations. Using the discrete time model, we can formulate the problem as an LP in arc variables. Since we have infinite supplies and demands, we can assume that each commodity $i \in K$ has a single source s^i and a single sink t^i and we will refer to their corresponding counterparts in the time expansion by s'^i and t'^i. For brevity, we define $\mathbb{T} := \{0, \dots, T - 1\}$. A variable $x^i_{a,\theta}$ describes the flow rate of commodity i into arc a at time step θ which is equivalent to the flow on the copy of a starting in time layer θ in the time-expanded network.

$$\begin{array}{llcll}
\max & \sum\limits_{i \in K} \sum\limits_{a \in \delta^-_{t'_i}} \sum\limits_{\theta \in \mathbb{T}} x^i_{a,\theta}, & & & \\
\text{s.t.} & \sum\limits_{i \in K} x^i_{a,\theta} & \leq & u_a & \text{for all } a \in A(G), \theta \in \mathbb{T}, \\
& \sum\limits_{a \in \delta^-_v} x^i_{a,\theta-\tau_a} - \sum\limits_{a \in \delta^+_v} x^i_{a,\theta} & = & 0 & \text{for all } i \in K, v \in V(G) \setminus \left\{s'^i, t'^i\right\}, \theta \in \mathbb{T}, \\
& x^i_{a,\theta} & \geq & 0 & \text{for all } a \in A(G), i \in K, \theta \in \mathbb{T}.
\end{array}$$

It is not difficult to see that due to flow conservation, all flow contributing to the objective function is sent from a source to a sink along some sequence of arcs. We define $\mathcal{S}$ to be the set of all s'^i-t'^i-sequences for all $i \in K$. Then there is always an optimal solution to the maximum multi-commodity flow over time problem that can be decomposed into sequences $\mathcal{S}^* \subseteq \mathcal{S}$. We want to split the sequences $S \in \mathcal{S}$ into smaller parts of nearly the same length $\Delta > 0$. Let τ^* be the maximum transit time of an arc. Then we can split S into subsequences of lengths between $\Delta - \tau^*$ and Δ. The length of the last subsequence can be anywhere between 0 and Δ, though, because we have to fit the remaining arcs there. This leads to the following definition.

$$\begin{aligned}\mathcal{S}^{\Delta}_{\Delta-\tau^*} := &\left\{ v\text{-}t'^i\text{-sequences } S \;\middle|\; i \in K, v \in V(G), \tau_S \le \Delta \right\}\\ &\cup \left\{ v\text{-}w\text{-sequences } S \mid i \in K, v, w \in V(G), \Delta - \tau^* \le \tau_S \le \Delta \right\}.\end{aligned}$$

Since optimal solutions can be decomposed into sequences in $\mathcal{S}^{\Delta}_{\Delta-\tau^*}$, we can also formulate an LP based on variables for all sequences in $\mathcal{S}^{\Delta}_{\Delta-\tau^*}$. Variable $x^i_{S,\theta}$ describes the flow rate of commodity i into sequence S at time step θ. Therefore, the following LP, which we will refer to as the *split-sequence LP*, can be used to compute an optimal solution to our problem:

$$\begin{array}{lllll}
\max & \sum\limits_{i\in K} \sum\limits_{S\in\delta^-_{t'^i}} \sum\limits_{\theta\in\mathbb{T}} x^i_{S,\theta-\tau_S}, & & & \\
\text{s.t.} & \sum\limits_{i\in K} \sum\limits_{S\in\mathcal{S}^{\Delta}_{\Delta-\tau^*}} \sum\limits_{\substack{j=1,\ldots,|S|:\\ a=a_j}} x^i_{S,\theta-\tau_{S[\to a_j]}} & \le & u_a & \text{for all } a \in A(G), \theta \in \mathbb{T},\\
& \sum\limits_{S\in\delta^-_v} x^i_{S,\theta-\tau_S} - \sum\limits_{S\in\delta^+_v} x^i_{S,\theta} & = & 0 & \text{for all } i\in K, v\in V(G)\setminus\{s'^i, t'^i\}, \theta\in\mathbb{T},\\
& x^i_{S,\theta} & \ge & 0 & \text{for all } S \in \mathcal{S}^{\Delta}_{\Delta-\tau^*}, i \in K, \theta \in \mathbb{T}.
\end{array}$$

This formulation of the problem has two advantages over the initial arc-based formulation. Firstly, all sequences in $S^{\Delta}_{\Delta-\tau^*}$ – with the exception of those ending in a sink – have a guaranteed length of $\Delta - \tau^*$. Since the time horizon T is an upper bound for the length of any sequence in $\mathcal{S}$, we can conclude that a sequence $S \in \mathcal{S}$ can be split into at most $\lceil T/(\Delta - \tau^*)\rceil + 1$ segments. Secondly, when rounding up the transit times of the sequence-segments, we know that the error introduced by the rounding is bounded by τ^* for all segments save for the last one, which ends in a sink. These two advantages together allow us to obtain a strong bound on the rounding error.

Solving the split-sequence LP. Unfortunately, the split-sequence LP cannot be solved directly in polynomial time due to its enormous size. We will approach this problem in two steps.

1. We round transit times to generate a number of discrete time steps that is polynomially bounded in the input size and ε^{-1}. This introduces a rounding error, which we bound in next paragraph.

2. The resulting LP has still exponentially many variables; therefore we will consider its dual, which has exponentially many constraints, but only polynomially many variables (in the input size and ε^{-1}). Finally, we argue that the dual separation problem can be approximated which yields an FPTAS for the original problem.

We begin by introducing a set $\mathcal{S}_L := \{S \text{ is a } v\text{-}w\text{-sequence with } \tau_S \geq L\}$ that be will necessary later due to the fact that we can only approximate the dual separation problem. We define $\overline{\mathcal{S}}_L^\Delta := \mathcal{S}_{\Delta-\tau^*}^\Delta \cup \mathcal{S}_L$ and replace $\mathcal{S}_{\Delta-\tau^*}^\Delta$ with $\overline{\mathcal{S}}_L^\Delta$ in the LP above. It is easy to see that this does not interfere with our ability to use this LP to solve our problem. We now round the transit times of the sequences in $\overline{\mathcal{S}}_L^\Delta$ up to the next multiple of Δ, i. e., $\tau'_S := \lceil \tau_S/\Delta \rceil \Delta$ for all $S \in \overline{\mathcal{S}}_L^\Delta$. Moreover, we define $T' := \lceil (1+\varepsilon)T/\Delta \rceil \Delta$ and $\mathbb{T}' := \{0, \Delta, \dots, T' - \Delta\}$. Using these transit times yields the following LP, which we will refer to as the *sequence-rounded LP*:

$$
\begin{array}{llll}
\max & \sum\limits_{i \in K} \sum\limits_{S \in \delta^-_{t'^i}} \sum\limits_{\theta \in \mathbb{T}'} x^i_{S,\theta-\tau'_S}, & & \\
\text{s.t.} & \sum\limits_{i \in K} \sum\limits_{S \in \overline{\mathcal{S}}_L^\Delta} \#(a,S) \cdot x^i_{S,\theta} & \leq \Delta\, u_a & \text{for all } a \in A(G), \theta \in \mathbb{T}', \\
& \sum\limits_{S \in \delta^-_v} x^i_{S,\theta-\tau'_S} - \sum\limits_{S \in \delta^+_v} x^i_{S,\theta} & = 0 & \text{for all } i \in K, v \in V(G)\backslash\{s'^i, t'^i\}, \theta \in \mathbb{T}', \\
& x^i_{S,\theta} & \geq 0 & \text{for all } S \in \overline{\mathcal{S}}_L^\Delta, i \in K, \theta \in \mathbb{T}'.
\end{array}
$$

Bounding the error of the sequence-rounded LP. In this section, we will analyze the error introduced by using sequence-rounded transit times. The lemmas and proofs used in this subsection built on and extend techniques introduced by Fleischer and Skutella [32]. We begin by bounding the error introduced by switching from the given transit times to the sequence-rounded transit times.

Lemma 5.18. *Let $S \in \mathcal{S}$ be a source-sink sequence and let τ be the normal transit times and τ' the sequence-rounded transit times. Then $\tau_S \leq \tau'_S \leq \tau_S + \varepsilon^2 T$, for $\Delta := \frac{\varepsilon^2 T}{4n}$ and $L := (1 - 1/4\varepsilon^2/(1 + 1/4\varepsilon^2))\Delta$.*

Proof. The rounding error for an sequence $S' \in \overline{\mathcal{S}}_L^\Delta$ is bounded by $\max(\tau^*, \Delta - L)$ for all sequences not ending in the sink of a commodity, for those ending in the sink it is bounded by Δ. W.l.o.g, we will now assume that $\Delta - L \geq \tau^*$, as we can decrease τ^* to $\Delta - L$ by splitting arcs – this requires at most $m\frac{T}{\Delta - L} \in O(\frac{nm}{\varepsilon^4})$ additional arcs for the L that we choose below. The rounding error for a source-sink-sequence decomposed into sequences of $\overline{\mathcal{S}}_L^\Delta$ is then bounded by

$$\left\lceil \frac{T}{L} \right\rceil (\Delta - L) + \Delta \leq \frac{T}{L}\Delta - T + 2\Delta - L\ .$$

We will now choose $\Delta := \frac{\varepsilon^2 T}{4n}$ and $L := (1 - \ell)\Delta$. Then

$$\frac{T}{L}\Delta - T + 2\Delta - L = \frac{\ell}{1-\ell} T + (1+\ell)\frac{\varepsilon^2 T}{4n}\ .$$

Now we choose $\ell := \frac{\frac{1}{4}\varepsilon^2}{1 + \frac{1}{4}\varepsilon^2}$ and get

$$\frac{\ell}{1-\ell} T + (1+\ell)\frac{\varepsilon^2 T}{4n} = \frac{1}{4}\varepsilon^2 T + \frac{1}{4n}\varepsilon^2 T + \frac{\frac{1}{4}\varepsilon^2}{1 + \frac{1}{4}\varepsilon^2}\frac{\varepsilon^2 T}{4n} \leq \varepsilon^2 T\ .$$

Thus, the rounding error is bounded by $\varepsilon^2 T$, i.e., a sequence is at most $\varepsilon^2 T$ longer when using the sequence-rounded transit times instead of the original ones. $\square$

This result enables us construct flows feasible with regard to the sequence-rounded transit times out of normal flows.

Lemma 5.19. *Let f be a multi-commodity flow over time, sending $|f|$ flow units within a time horizon of T. Then, for $\varepsilon > 0$, there exists a multi-commodity flow over time that is feasible with regard to the sequence-rounded transit times, sending $|f|/(1+\varepsilon)$ flow units within a time horizon of $(1+\varepsilon)T$.*

Proof. W.l.o.g., we can assume that f can be decomposed into s'^i-t'^i-sequences. Otherwise, there exists a flow of equal value with this property. All these s'^i-t'^i-sequences can be further decomposed into sequences of $\overline{\mathcal{S}}_L^\Delta$. We will now examine the effects of sending f with sequence-rounded transit times. This means that all flow particles of f still travel along the same sequences, as specified by the decomposition, but the time to traverse a sequence is now different than before. This does not affect flow conservation, since we just change the speed at which flow travels. However, we might violate capacities. Therefore, we define a smoothed flow based on the sequence-decomposition in order to lessen the effects of this:

$$f_S^{sm,i}(\theta) := \frac{1}{\varepsilon T} \int_{\theta-\varepsilon T}^{\theta} f_S^i(\xi)\ d\xi \quad \text{for all } S \in \overline{\mathcal{S}}_L^\Delta, i \in K, \theta \in [0,(1+\varepsilon)T)\ .$$

Due to the smoothing, this flow requires a time horizon that is εT longer than the original one, resulting in a time horizon of $(1+\varepsilon)T$ for f_{sm}. The flow value of each arc a at a point in time θ is then bounded by:

$$\begin{aligned} f_a^{sm}(\theta) :=& \sum_{i\in K} \sum_{S\in\overline{\mathcal{S}}_L^\Delta} \sum_{j=1,\dots,|S|:a=a_j} f_S^{sm,i}(\theta - \tau'_{S[\to a_j]}) \\ =& \sum_{i\in K} \sum_{S\in\overline{\mathcal{S}}_L^\Delta} \sum_{j=1,\dots,|S|:a=a_j} \frac{1}{\varepsilon T} \int_{\theta-\varepsilon T-\tau'_{S[\to a_j]}}^{\theta-\tau'_{S[\to a_j]}} f_S^i(\xi)\ d\xi\ . \end{aligned}$$

In the next step, we use the bound on the error introduced by the sequence-rounding which we have obtained in Lemma 5.18. This gives us the following estimation:

$$\begin{aligned} & \sum_{i\in K} \sum_{S\in\overline{\mathcal{S}}_L^\Delta} \sum_{j=1,\dots,|S|:a=a_j} \frac{1}{\varepsilon T} \int_{\theta-\varepsilon T-\tau'_{S[\to a_j]}}^{\theta-\tau'_{S[\to a_j]}} f_S^i(\xi)\ d\xi \\ \le & \frac{1}{\varepsilon T} \sum_{i\in K} \sum_{S\in\overline{\mathcal{S}}_L^\Delta} \sum_{j=1,\dots,|S|:a=a_j} \int_{\theta-\varepsilon T-\varepsilon^2 T-\tau_{S[\to a_j]}}^{\theta-\tau_{S[\to a_j]}} f_S^i(\xi)\ d\xi \\ = & \frac{1}{\varepsilon T} \int_{\theta-\varepsilon T-\varepsilon^2 T}^{\theta} \sum_{i\in K} \sum_{S\in\overline{\mathcal{S}}_L^\Delta} \sum_{j=1,\dots,|S|:a=a_j} f_S^i(\xi - \tau_{S[\to a_j]})\ d\xi. \end{aligned}$$

This can be further transformed to:

$$\begin{aligned}
&\frac{1}{\varepsilon T}\int_{\theta-\varepsilon T-\varepsilon^2 T}^{\theta}\sum_{i\in K}\sum_{S\in\overline{\mathcal{S}}_L^{\Delta}}\sum_{j=1,\ldots,|S|:a=a_j} f_S^i(\xi-\tau_{S[\to a_j]})\;d\xi\\
&=\frac{1}{\varepsilon T}\int_{\theta-\varepsilon T-\varepsilon^2 T}^{\theta} f_a(\xi)\;d\xi\\
&\le\frac{1}{\varepsilon T}\int_{\theta-\varepsilon T-\varepsilon^2 T}^{\theta} u_a\;d\xi=\frac{\varepsilon T+\varepsilon^2 T}{\varepsilon T}u_a=(1+\varepsilon)u_a\ .
\end{aligned}$$

Therefore, we can scale the smoothed flow by $\frac{1}{1+\varepsilon}$ and obtain a feasible flow, that sends $\frac{1}{1+\varepsilon}|f|$ flow units within a time horizon $(1+\varepsilon)T$:

$$f_S^{sc+sm,i}(\theta):=\frac{1}{1+\varepsilon}f_S^{sm,i}(,\theta)\quad\text{for all } S\in\overline{\mathcal{S}}_L^{\Delta}, i\in K,\theta\in[0,(1+\varepsilon)T)\ .$$

This concludes the proof. □

Furthermore, we can obtain a solution to the sequence-rounded LP out of any flow feasible with regard to the sequence-rounded transit times.

Lemma 5.20. *Let f be a feasible multi-commodity flow over time with regard to the sequence-rounded transit times. Then there exists a solution x to the sequence rounded LP with equal value.*

Proof. Define x by

$$x_{a,\theta}^i:=\int_{\theta\Delta}^{(\theta+1)\Delta} f_a^i(\xi)\;d\xi\quad\text{for all } i\in K,a\in A(G),\theta\in\mathbb{T}'\ .$$

Since f is feasible with sequence-rounded transit times, x is feasible in the sequence-rounded LP. Due to f obeying flow-conservation, so does x. It is easy to see that the total flow value sent does not change, proving the claim. □

Finally, a solution to the sequence-rounded LP can be turned into a normal flow.

Lemma 5.21. *Let x be a solution to the sequence-rounded LP. Then, for any $\varepsilon>0$, one can compute a multi-commodity flow over time without intermediate storage that sends $|x|/(1+\varepsilon)$ units of flow within time horizon $(1+\varepsilon)T$.*

Proof. Consider the flow over time f corresponding to x and a decomposition $(f_S)_{S\in\overline{\mathcal{S}}_L^{\Delta}}$ of f. We will now analyze what happens if we send the flow of f using normal transit times. This means that all flow particles of f still travel along the same sequences, as specified by the decomposition, but the time to traverse a sequence is now different than before. This does not affect flow conservation, since we just change the speed at which flow particles travel. However, due to the different transit times, we might violate capacities. Therefore, we define a smoothed flow by

$$f_S^{sm,i}(\theta) := \frac{1}{\varepsilon T} \int_{\theta-\varepsilon T}^{\theta} f_S^i(\xi)\ d\xi \quad \text{for all } S \in \overline{\mathcal{S}}_L^{\Delta}, i \in K, \theta \in [0, (1+\varepsilon)T) \enspace .$$

in order to limit the capacity violations. The new flow requires εT more time, resulting in a time horizon of $(1+\varepsilon)T$. Now we need to examine how much capacities are violated by f^{sm} if send with the normal transit times:

$$\begin{aligned} f_a^{sm}(\theta) &= \sum_{i \in K} \sum_{S \in \overline{\mathcal{S}}_L^{\Delta}} \sum_{j=1,\dots,|S|:a=a_j} f_S^{sm,i}(\theta - \tau_{S[\to a_j]}) \\ &= \sum_{i \in K} \sum_{S \in \overline{\mathcal{S}}_L^{\Delta}} \sum_{j=1,\dots,|S|:a=a_j} \frac{1}{\varepsilon T} \int_{\theta-\varepsilon T-\tau_{S[\to a_j]}}^{\theta-\tau_{S[\to a_j]}} f_S^i(\xi)\ d\xi \enspace . \end{aligned}$$

At this point, we use the fact that the error introduced by the sequence rounding is at most $\varepsilon^2 T$ (see Lemma 5.18). We use this for the following estimation:

$$\begin{aligned} &\sum_{i \in K} \sum_{S \in \overline{\mathcal{S}}_L^{\Delta}} \sum_{j=1,\dots,|S|:a=a_j} \frac{1}{\varepsilon T} \int_{\theta-\varepsilon T-\tau_{S[\to a_j]}}^{\theta-\tau_{S[\to a_j]}} f_S^i(\xi)\ d\xi \\ \leq\ & \frac{1}{\varepsilon T} \sum_{i \in K} \sum_{S \in \overline{\mathcal{S}}_L^{\Delta}} \sum_{j=1,\dots,|S|:a=a_j} \int_{\theta-\varepsilon T-\tau'_{S[\to a_j]}}^{\theta-\tau'_{S[\to a_j]}+\varepsilon^2 T} f_S^i(\xi)\ d\xi \\ =\ & \frac{1}{\varepsilon T} \int_{\theta-\varepsilon T}^{\theta+\varepsilon^2 T} \sum_{i \in K} \sum_{S \in \overline{\mathcal{S}}_L^{\Delta}} \sum_{j=1,\dots,|S|:a=a_j} f_S^i(\xi - \tau'_{S[\to a_j]})\ d\xi \\ =\ & \frac{1}{\varepsilon T} \int_{\theta-\varepsilon T}^{\theta+\varepsilon^2 T} f_a(\xi)\ d\xi \\ \leq\ & \frac{1}{\varepsilon T} \int_{\theta-\varepsilon T}^{\theta+\varepsilon^2 T} u_a\ d\xi \\ =\ & \frac{\varepsilon T + \varepsilon^2 T}{\varepsilon T} u_a = (1+\varepsilon) u_a \enspace . \end{aligned}$$

Therefore, we can scale f^{sm} to get an flow, that sends $\frac{1}{1+\varepsilon}|x|$ flow units within a time horizon $(1+\varepsilon)T$ and obeys the capacities:

$$f_S^{sc+sm,i}(\theta) := \frac{1}{1+\varepsilon} f_S^{sm,i}(\theta) \quad \text{for all } S \in \overline{\mathcal{S}}_L^{\Delta}, i \in K, \theta \in [0, (1+\varepsilon)T) \enspace .$$

Since f is feasible, this flow also fulfills flow conservation making it feasible as well.

Consider an arbitrary $i \in K, v \in V(G) \setminus \{s^i, t^i\}, \theta \in [0, T)$:

$$\begin{aligned}
&\sum_{S\in\delta_v^-} f_S^{sc+sm,i}(\theta) - \sum_{S\in\delta_v^+} f_S^{sc+sm,i}(\theta)\\
&= \sum_{S\in\delta_v^-} \frac{1}{1+\varepsilon} f_S^{sm,i}(\theta) - \sum_{S\in\delta_v^+} \frac{1}{1+\varepsilon} f_S^{sm,i}(\theta)\\
&= \sum_{S\in\delta_v^-} \frac{1}{1+\varepsilon}\frac{1}{\varepsilon T}\int_{\theta-\varepsilon T}^{\theta} f_S^i(\xi)\; d\xi - \sum_{S\in\delta_v^+} \frac{1}{1+\varepsilon}\frac{1}{\varepsilon T}\int_{\theta-\varepsilon T}^{\theta} f_S^i(\xi)\; d\xi\\
&= \frac{1}{1+\varepsilon}\frac{1}{\varepsilon T}\int_{\theta-\varepsilon T}^{\theta} \sum_{S\in\delta_v^-} f_S^i(\xi) - \sum_{S\in\delta_v^+} f_S^i(\xi)\; d\xi\\
&= \frac{1}{1+\varepsilon}\frac{1}{\varepsilon T}\int_{\theta-\varepsilon T}^{\theta} 0\; d\xi = 0\ .
\end{aligned}$$

This concludes the proof. □

For the sake of completeness, we also discuss the computation of f^{sc+sm} and related functions used in previous lemmas.

Lemma 5.22. *f^{sc+sm} can be computed efficiently.*

Proof. Let x be a solution to the sequence-rounded LP that can be decomposed into sequences. There is no waiting in intermediate nodes – thus, each sequence can let flow wait at the source only. Therefore, each sequence S of the decomposition of x induces a sequence-flow over time f_S that sends x_S flow in a time interval of length Δ and zero flow otherwise; i. e. each f_S is a step-function with one non-zero-step. There are at most as many sequences as time-layers in the time-expanded network that correspond to the same sequence in the static network (recall that waiting at intermediate nodes is forbidden). Thus, the result is a piecewise constant function describing the flow for each sequence, that has at most as many steps as time-layers ($T' \setminus \Delta \in O(n\varepsilon^{-2})$). Thus, each of $f_S^{sc+sm,i}$ is efficiently computable and the number of such functions is polynomial in the input as well (since x is a flow in a network of polynomial size). □

Solving the Sequence-Rounded LP's Dual. It remains to show that we can solve the sequence-rounded LP efficiently. Recall that the sequence-rounded LP is

$$\begin{array}{llll}
\max & \sum_{i\in K}\sum_{S\in\delta_{t'^i}^-}\sum_{\theta\in\mathbb{T}'} x^i_{S,\theta-\tau'_S}, & & \\
\text{s.t.} & \sum_{i\in K}\sum_{S\in\overline{\mathcal{S}}_L^\Delta} \#(a,S)\cdot x^i_{S,\theta} & \leq\ \Delta\, u_a & \text{for all } a\in A(G), \theta\in\mathbb{T}',\\
& \sum_{S\in\delta_v^-} x^i_{S,\theta-\tau'_S} - \sum_{S\in\delta_v^+} x^i_{S,\theta} & =\ 0 & \text{for all } i\in K, v\in V(G)\setminus\{s'^i,t'^i\}, \theta\in\mathbb{T}',\\
& x^i_{S,\theta} & \geq\ 0 & \text{for all } S\in\overline{\mathcal{S}}_L^\Delta, i\in K, \theta\in\mathbb{T}'.
\end{array}$$

The dual of this LP has a variable $\alpha^i_{v,\theta}$ for every flow conservation constraint of a commodity $i \in K$, node $v \in V(G) \setminus \{s'^i, t'^i\}$, and time $\theta \in \mathbb{T}'$, as well as a variable $\beta_{a,\theta}$ for every capacity constraint of an arc $a \in A(G)$ and time $\theta \in \mathbb{T}'$. This leads us to the following observation.

Observation 5.23. *The dual of the sequence-rounded LP is*

$$\begin{array}{llll} \min & \sum\limits_{a \in A(G)} \sum\limits_{\theta \in \mathbb{T}'} \Delta\, u_a \beta_{a,\theta}, & & \\ \text{s.t.} & -\alpha^i_{v,\theta} + \alpha^i_{w,\theta+\tau'_S} + \sum\limits_{a \in A(G)} \#(a,S)\beta_{a,\theta} & \geq 1 & \text{for all } S \in \overline{\mathcal{S}}^{\Delta}_L, i \in K, \theta \in \mathbb{T}', \\ & \beta_{a,\theta} & \geq 0 & \text{for all } a \in A(G), \theta \in \mathbb{T}'. \end{array}$$

with $\alpha^i_{s'^i,\theta} = \alpha^i_{t'^i,\theta} = 0$ for all commodities $i \in K$ and $\theta \in \mathbb{T}'$, since these variables do not correspond to constraints in the primal LP and have just been added for notational brevity. The separation problem of this dual is to find a sequence $S \in \overline{\mathcal{S}}^{\Delta}_L$ with $\sum\limits_{a \in A} \#(a,S) \cdot \beta_{a,\theta} < 1 + \alpha^i_{v,\theta} - \alpha^i_{w,\theta+\tau'_S}$ for some v, w, i, θ.

Notice that the number of different combinations of v, w, i, θ is polynomially bounded such that we can enumerate them efficiently. Thus, the separation problem is to find a v-w-sequence S of minimum cost with regard to β and $S \in \overline{\mathcal{S}}^{\Delta}_L$. This is the restricted minimum cost s-t-sequence problem introduced in Chapter 5.3.1 with β as the costs and τ as the lengths and Δ as the threshold. As we have seen there, we can only approximately solve this separation problem in the sense that we get a solution with a length between $(1-\varepsilon)\Delta-\tau^*$ and $(1+\varepsilon)\Delta$. Recall that we have defined $\overline{\mathcal{S}}^{\Delta}_L$ as $\mathcal{S}^{\Delta}_{\Delta-\tau^*} \cup \mathcal{S}_L$ for this purpose. Until this point, choosing $\mathcal{S}_L = \emptyset$ would have been completely fine, we have not needed it so far. However, using the equivalence of separation and optimization [53], we can find a solution to a dual problem where $\mathcal{S}_L$ contains the additional sequences of length at least $(1-\varepsilon)\Delta - \tau^*$ found by the ellipsoid method. This leads to $L := (1-\varepsilon)\Delta - \tau^*$ which is also in line with the value of L we have used in the analysis before. We conclude this section by stating our main result.

Theorem 5.24. *Let I be an instance of the maximum multi-commodity flow over time problem without intermediate storage and let OPT be the value of an optimal solution to it. There exists an* FPTAS *that, for any given $\varepsilon > 0$, computes a solution with value at least $OPT/(1+\varepsilon)$ and time horizon at most $(1+\varepsilon)T$.*

5.3.3 Extensions

Our approach above is based on the fact that we can efficiently approximate the separation problem of the dual LP. We will now show that this is also the case for the multi-commodity flow over time problem without storage and the minimum cost multi-commodity flow over time problem.

We focus now on the minimum cost multi-commodity flow over time problem without intermediate storage, as this problem is a generalization of the multi-commodity flow over time problem without intermediate storage. The case with intermediate storage can be done similarly.

We begin by considering an arc based LP formulation that is again based on a time-expanded network. Since we are using a time expansion, we can introduce a super-source s'^i and -sink t'^i for each commodity $i \in K$, even though our supplies and demands may be finite. For this purpose, we define additional nodes s^i_v for each $v \in V(G)$ with supply $b^i_v > 0$ and t^i_v for each node $v \in V(G)$ with demand $b^i_v < 0$. Then we define new supplies and demands b' by $b'^i_{s'^i} := \sum_{v\in V: b^i_v>0} b^i_v$, $b'^i_{t'^i} := \sum_{v\in V: b^i_v<0} b^i_v$ and $b'^i_v = 0$ for the rest of the nodes. After that, we connect the super-nodes by arcs $a_{s^i_v} = (s'^i, s^i_v)$ and $a_{t^i_v} = (t^i_v, t'^i)$ with the supply/demand-nodes. The capacity of the new arcs is determined by the supply and demand of the nodes they are connected to, i. e. $u_{a_{s^i_v}} = |b^i_v|$, $u_{a_{t^i_v}} = |b^i_v|$. These arcs have zero transit time and zero cost. The supply nodes s^i_v and demand nodes t^i_v are then connected with all copies of their node v in the time-expanded network by arcs with infinite capacity, zero transit time and zero cost. This gives us a network where only two nodes per commodity have non-zero supply and demand, which makes the structure of the LP much nicer. Also, these nodes have only outgoing and only incoming arcs, respectively.

In order to make the LP more similar to the last problem, we transform it into a maximization problem. For brevity, we define $\mathbb{V} := \left\{ s'^i, t'^i \;\middle|\; i \in K \right\}$. We get the following LP based on arc variables. Notice that the arcs incident to nodes of $\mathbb{V}$ exist only once, i. e. they are outside the copied structure.

$$
\begin{array}{lllll}
\max & -\sum\limits_{i\in K}\sum\limits_{a\in A(G)}\sum\limits_{\theta\in\mathbb{T}} c_a \cdot x^i_{a,\theta}, & & & \\
\text{s.t.} & \sum\limits_{i\in K} x^i_{a,\theta} & \le & u_a & \text{for all } a \in A(G), \theta \in \mathbb{T}, \\
& \sum\limits_{a\in\delta^+_v} x^i_{a,\theta} - \sum\limits_{a\in\delta^-_v} x^i_{a,\theta-\tau_a} & = & 0 & \text{for all } i \in K, v \in V(G)\setminus\mathbb{V}, \theta \in \mathbb{T}, \\
& \sum\limits_{a\in\delta^+_v} x^i_a - \sum\limits_{a\in\delta^-_v} x^i_a & = & b'^i_v & \text{for all } i \in K, v \in \mathbb{V}, \theta \in \mathbb{T}, \\
& x^i_{a,\theta} & \ge & 0 & \text{for all } a \in A(G), i \in K, \theta \in \mathbb{T}.
\end{array}
$$

Notice that the changes compared to the maximum multi-commodity flow over time problem without intermediate storage are mostly in the objective function and the right side of the flow conservation constraints. Again we can consider a sequence-rounded LP.

$$
\begin{array}{lllll}
\max & -\sum\limits_{i\in K}\sum\limits_{S\in\overline{\mathcal{S}}^\Delta_L}\sum\limits_{\theta\in\mathbb{T}'} c_S \cdot x^i_{S,\theta}, & & & \\
\text{s.t.} & \sum\limits_{i\in K}\sum\limits_{S\in\overline{\mathcal{S}}^\Delta_L} \#(a,S)\cdot x^i_{S,\theta} & \le & \Delta u_a & \text{for all } a \in A(G), \theta \in \mathbb{T}', \\
& \sum\limits_{S\in\delta^+_v} x^i_{S,\theta} - \sum\limits_{a\in\delta^-_v} x^i_{S,\theta-\tau'_S} & = & 0 & \text{for all } i \in K, v \in V(G)\setminus\mathbb{V}, \theta \in \mathbb{T}', \\
& \sum\limits_{S\in\delta^+_v} x^i_S - \sum\limits_{S\in\delta^-_v} x^i_S & = & b'^i_v & \text{for all } i \in K, v \in \mathbb{V}, \theta \in \mathbb{T}', \\
& x^i_{S,\theta}, x^i_S & \ge & 0 & \text{for all } S \in \overline{\mathcal{S}}^\Delta_L, i \in K, \theta \in \mathbb{T}'.
\end{array}
$$

The dual of this sequence-rounded LP is:

$$\begin{array}{llll}
\min & \sum_{a\in A(G),\theta\in\mathbb{T}'} \Delta u_a \beta_{a,\theta} + \sum_{i\in K, v\in\mathbb{V},\theta\in\mathbb{T}'} b'^i_v \alpha^i_{v,\theta}, & & \\
\text{s.t.} & \alpha^i_{v,\theta} - \alpha^i_{w,\theta+\tau'_S} + \sum_{a\in A(G)} \#(a,S)\beta_{a,\theta} & \geq -c_S & \text{for all } S \in \overline{\mathcal{S}}^{\Delta}_L, i \in K, \theta \in \mathbb{T}', \\
& \beta_{a,\theta} & \geq 0 & \text{for all } a \in A(G), \theta \in \mathbb{T}'.
\end{array}$$

with $\alpha^i_{s'^i,\theta} = \alpha^i_{t'^i,\theta} = 0$ for all commodities $i \in K$ and $\theta \in \mathbb{T}'$, since these variables do not correspond to constraints in the primal LP and have just been added for notational brevity. Notice that $c_S = \sum_{a\in A(G)} \#(a,S)c_a$. Therefore, we can write the constraint for all $S \in \overline{\mathcal{S}}^{\Delta}_L, i \in K, \theta \in \mathbb{T}'$ as

$$\sum_{a\in A(G)} \#(a,S)\cdot(\beta_{a,\theta} + c_a) \geq -\alpha^i_{v,\theta} + \alpha^i_{w,\theta+\tau'_S}.$$

For non-negative costs, this leads once again to a restricted minimum cost s-t-sequence problem as the separation problem.

Theorem 5.25. *Let I be an instance of the minimum cost multi-commodity flow over time without storage problem with non-negative arc costs and let OPT be the value of an optimal solution to it. There exists an* FPTAS *that, for any given $\varepsilon > 0$, finds a solution of value at most OPT, with time horizon at most $(1+\varepsilon)T$ that fulfills a $(1+\varepsilon)^{-1}$ fraction of the given supplies and demands.*

We mention that for the multi-commodity flow over time problems with storage at intermediate nodes considered in [32], Fleischer and Skutella obtained a stronger approximation result where only the time horizon is increased by a factor $1+\varepsilon$ but the original, optimal flow values are sent. In our setting without storage, however, we also need to approximate the flow values and can only send a $1/(1+\varepsilon)$ fraction. This is due to the fact that for the case where intermediate storage is allowed, increasing the time horizon by a factor λ allows to increase flow values by a factor of λ (see Lemma 5.4). Such a result does not hold without storage, see for example our discussion in Chapter 3.2.4. We can apply Lemma 5.4 to the minimum multi-commodity cost flow over time problem with storage, though, in order to send all supplies and demands.

6 Temporal Repetition

In this chapter, we present our algorithmic results that work on the original graph and do not require time expansion. More specifically, we study the special case of generalized maximum flows over time with proportional losses. That means that on any arc with a transit time of τ, we lose the same fraction γ of flow. We construct a successive shortest path variant for this setting which works on the original network, but requires a pseudo-polynomial number of iterations in the worst case. Based upon this algorithm, we create an FPTAS for this problem. This FPTAS is a value-approximation instead of the time-approximations that we saw in the last chapter. This is a relatively rare feature for flow over time approximations, as they usually approximate time. Another peculiarity of this FPTAS is that the error ε appears only logarithmically as $\log \varepsilon^{-1}$ in the runtime of the FPTAS – making it extremely cheap to obtain very accurate approximations. This was published in Groß and Skutella [51].

6.1 Generalized Maximum Flows over Time with Proportional Losses

Here, we consider the case of lossy and proportional gains, i. e., $\gamma \equiv 2^{c \cdot \tau}$, for some constant $c < 0$. This means that in each time unit the same percentage of the remaining flow value is lost. This is motivated by problems where goods cannot be transported reliably, e. g., due to leakage or evaporation. In many applications, this loss depends on the time spent in the transportation network, and many processes of growth or decay in nature evolve over time according to an exponential function. This concept of proportional transit times and gains is analogous to the setting of minimum cost flows over time, where Fleischer and Skutella [31] considered the special case of costs proportional to transit times. We will focus on the setting of a single source and sink here, which we will usually refer to as s and t. We assume that source and sink have infinite supplies and demands, respectively. We split our results for this case into two parts.

- First, we show that the generalized maximum flow over time problem can be solved in the static network by a successive shortest path technique. While successive shortest path based algorithms can require pseudo-polynomially iterations, the important point of this algorithm is that it does not require time expansion like so many other algorithms do.
- Second, we employ the algorithm from the first part to obtain a value-FPTAS for the generalized maximum flow over time problem with proportional losses. An

interesting point here is that this is a value-FPTAS, whereas most flow over time FPTASs are time-FPTAS. That we can approximate this problem by value is essentially due to the fact that we have the algorithm from the first part, which does not require time expansion.

The work in this chapter was first published in Groß and Skutella [51].

6.1.1 A Variant of the Successive Shortest Path Algorithm

Due to the work of Onaga [82, 83] it is known that augmenting flow successively along highest gain s-t-paths solves the generalized maximum flow problem with a single source s and a single sink t. More precisely, Onaga's algorithm for lossy networks proceeds as follows. Begin with the zero-flow and the corresponding residual network. If no source-sink path exists in this residual network, terminate. Otherwise, augment flow along a source-sink path of maximum gain and continue with the resulting flow and residual network. Thus, applying Onaga's algorithm in the time-expanded network solves the generalized maximum flow over time problem – at the cost of potentially requiring pseudo-polynomially many augmentations in the pseudo-polynomially large time-expanded network.

We will now present an algorithm capable of solving the special case described above using only the original – not time-expanded – network. The idea of this algorithm is to employ a strategy similar to Onaga's in the original network and use this as a foundation to construct a flow over time solving the special case. In order to do this, we need some temporally repeated structures that we can work with.

Temporally repeated structures in a time expansion. We begin by introducing some notations; more precisely, we introduce a slightly non-standard way of building a time-expanded network from copies of the original network. Let $(G, u, \tau, \gamma, s, t, T)$ be a (residual) network, let $\gamma_{v\to w}$ be the maximum gain of a v-w-path in G and $\tau_{v\to w}$ the length of a shortest v-w-path (with respect to transit times) in G. Notice that $\tau_{v\to w} := \frac{1}{c}\log\gamma_{v\to w}$.

We begin by subdividing a time-expanded network G^T into parts using the original graph G. We do this by identifying copies of G in G^T – we pick a copy s_θ of the source s in G^T, and add arcs and nodes panning out from s_θ along shortest paths. We call the resulting copy of G a θ-copy θG, for some $\theta \in \{0,\dots,T-\tau_{s\to t}-1\}$. θ-copies are somewhat simpler to define in highest gain networks, which are equivalent to shortest path networks in our setting. In these networks, all paths from s to a node v have the same gain and transit time. The *θ-copy* θG of G for is then defined by:

$$\begin{aligned} V(\theta G) &:= \left\{ v_\xi \in V(G^T) \;\middle|\; v \in V(G), \xi = \theta + \tau_{s\to v} \right\}, \\ A(\theta G) &:= \left\{ a_\xi \in A(G^T) \;\middle|\; a = (v,w) \in A(G), \xi = \theta + \tau_{s\to v} \right\}. \end{aligned}$$

Figure 6.1 shows a highest gain network and a θ-copy in its time expansion.

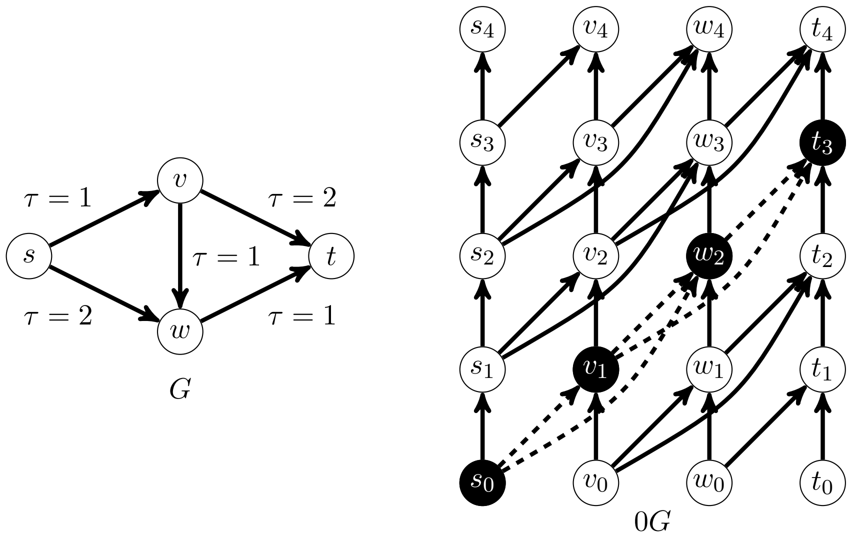


Figure 6.1: A highest gain network G and its time-expanded network G^5, with the θ-copy $0G$ marked by dashed arcs and black nodes.

For more general graphs, define the θ-copy θG of G for some $\theta \in \{0, \dots, T - \tau_{s\to t} - 1\}$ to be the following subgraph of the time-expanded network G^T:

$$A(\theta G) := \left\{ a_\xi \in A(G^T) \;\middle|\; a = (v,w) \in A(G), \xi = \theta + \tau_{s\to v}, \xi + \tau_a + \tau_{w\to t} < T \right\},$$
$$V(\theta G) := \bigcup_{a=(v,w)\in A(\theta G)} \{v, w\}\,.$$

For the special case mentioned above these two definitions coincide. Figure 6.2 shows an example of a network and a θ-copy in its time expansion.

We continue to define unions of adjacent θ-copies together with the holdover arcs that connect them. A union of the θ-copies from θ to θ' for $0 \le \theta < \theta' \le T - \tau_{s\to t} - 1$ is then called $[\theta, \theta']G$, and defined as:

$$V([\theta,\theta']G) := \bigcup_{\xi=\theta}^{\theta'} V(\xi G),$$
$$A([\theta,\theta']G) := \bigcup_{\xi=\theta}^{\theta'} A(\xi G) \cup \bigcup_{v\in V(G)} \bigcup_{\xi=\theta+\tau_{s\to v}}^{\theta'+\tau_{s\to v}-1} \{(v_\xi, v_{\xi+1})\}\,.$$

Figure 6.3 shows a network and its $[0, 1]$-copies in the time expansion.

We are particularly interested in θ-copies in which we can send flow from the source to the sink, which are the copies from 0 to $T - 1 - \tau_{s\to t}$, if $T - 1 - \tau_{s\to t} < T - 1$. All later θ-copies start too late to have flow arriving before the time horizon. Therefore, we define for brevity $\overline{G} := [0, T - \tau_{s\to t} - 1]G$ if $\tau_{s\to t} < T - 1$ and as the empty graph otherwise.

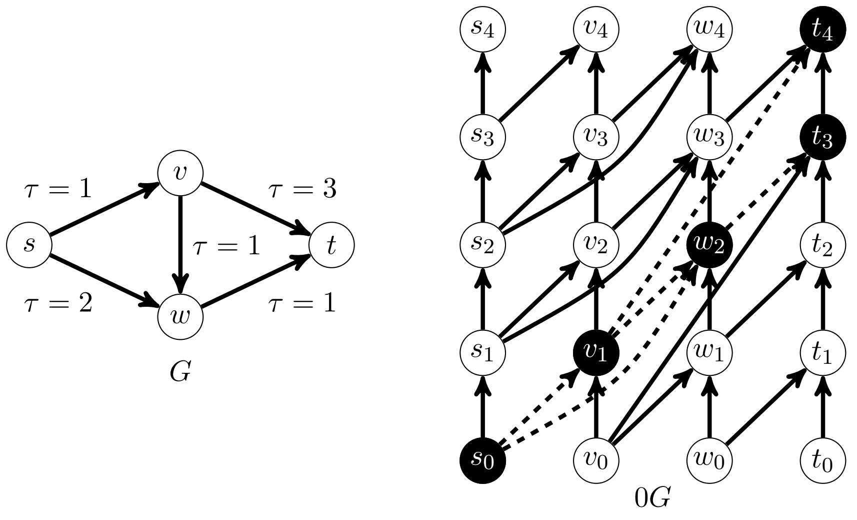


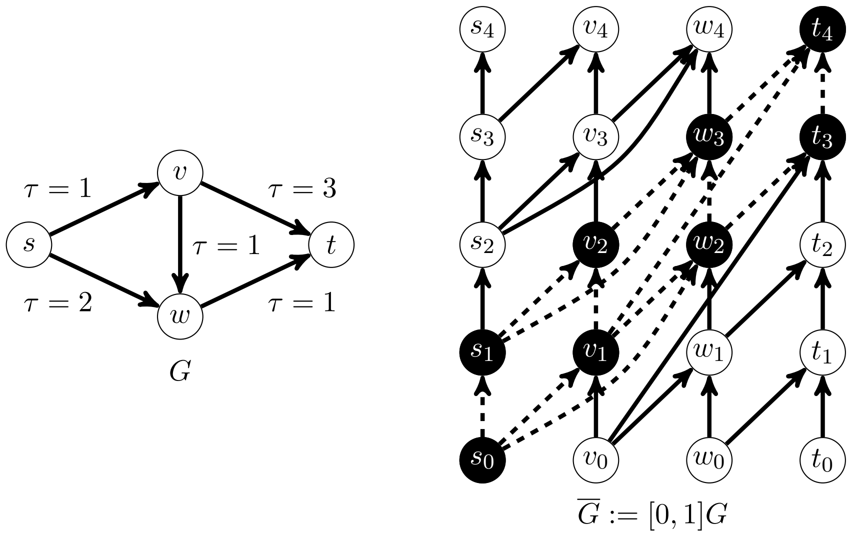
Figure 6.2: A network G and its time-expanded network G^5, with the θ-copy $0G$ marked by dashed arcs and black nodes.

If we consider a normal time-expanded network without the θ-copy structure, we can always remove all arcs and nodes that are not part of s'-t'-paths (with s', t' being the source and sink of the time-expanded network, see Definitions 4.1 and 4.2). Furthermore, if we consider time-expanded residual networks, we can simplify the network slightly further. In this case, the residual network G_f^T corresponding to a flow f can have reverse holdover arcs at source and sink. These reverse holdover arcs do not help to construct new s'-t'-paths in the time-expanded-network or new flow-generating cycles with a path to t', since we already have holdover arcs of infinite capacity between all copies of the source and the sink, respectively. Thus, we can omit them without reducing the value of an optimal solution as well. We write $\widetilde{G_f^T}$ for the subnetwork of G_f^T created by removing nodes and arcs not on s'-t'-paths and reverse holdover arcs at source and sink. Figure 6.4 shows such a pruned network.

We will show later (Lemma 6.2) that $\overline{G_f} = \widetilde{G_f^T}$ holds for the flows we are considering, i. e., $\overline{G_f}$ is the subnetwork of G_f^T containing exactly the nodes and arcs of G_f^T that can be part of s'-t'-paths.

Flows in θ-copies. Now we want to extend the concept of θ-copies to flows. We begin by transferring a flow x in a normal graph to a flow in a θ-copy. Let us assume that x is a flow in a static unique gain network G. We define the *θ-flow* θx of x in θG for some $\theta \in \{0, \dots, T - \tau_{s\to t} - 1\}$ by $(\theta x)_{a_\xi} := x_a$ for all $a_\xi \in A(\theta G)$.

Similarly, we can define a transfer of x to a series of temporally adjacent θ-copies. We define the *$[\theta, \theta']$-flow* $[\theta, \theta']x$ of x in $[\theta, \theta']G$ for some $\theta, \theta' \in \{0, \dots, T - \tau_{s\to t} - 1\}$ with $\theta < \theta'$ by setting $([\theta, \theta']x)_a := x_a$ for all arcs $a = a_\xi$ in a specific θ-copy. For holdover arcs between nodes that are not copies of source or sink, we set $([\theta, \theta']x)_{(v_\xi, v_{\xi+1})} := 0$,



$\tau = 1$
$\tau = 3$
$\tau = 1$
$\tau = 2$
$\tau = 1$
s
v
w
t
G
s_0 s_1 s_2 s_3 s_4
v_0 v_1 v_2 v_3 v_4
w_0 w_1 w_2 w_3 w_4
t_0 t_1 t_2 t_3 t_4
$\overline{G} := [0,1]G$

Figure 6.3: A network G and its time-expanded network G^5, with the $[0,1]$-copies $[0,1]G$ marked by dashed arcs and black nodes.

$v \in V(G) \setminus \{s,t\}$. For holdover arcs between copies of the source s, we define

$$([\theta,\theta']x)_a := \begin{cases} (\theta' - \theta + 1)|x| & a = (s_\xi, s_{\xi+1}), \xi < \theta, \\ (\theta' - \xi)|x| & a = (s_\xi, s_{\xi+1}), \theta \leq \xi < \theta', \\ 0 & a = (s_\xi, s_{\xi+1}), \theta' \leq \xi. \end{cases}$$

Similarly, we define for holdover arcs between copies of the sink T:

$$([\theta,\theta']x)_a := \begin{cases} 0 & a = (t_\xi, t_{\xi+1}), \xi \leq \theta + \tau_{s\to t}, \\ (\xi - \theta - \tau_{s\to t} + 1)|x| & a = (t_\xi, t_{\xi+1}), \theta + \tau_{s\to t} < \xi < \theta' + \tau_{s\to t}, \\ (\theta' - \theta + 1)|x| & a = (t_\xi, t_{\xi+1}), \xi \geq \theta' + \tau_{s\to t}. \end{cases}$$

Again, we define for brevity $\overline{x} := [0, T - \tau_{s\to t} - 1]x$ for a flow x in G, if $\tau_{s\to t} < T - 1$ and as the zero flow otherwise.

The algorithm. Informally spoken, the idea of our algorithm is to start with the zero flow, compute a maximum flow in the highest-gain / shortest-path subnetwork of the static residual network, augment this flow and repeat this process until no s-t-path exists in the static residual network. We then use the augmented maximum flows to construct an optimal solution to our problem, by sending each flow as long as possible into the network (i. e., temporally repeated). We will use the notations introduced above to describe the construction of the flow over time in the last step of our algorithm, and show its optimality.

Algorithm 6.1. *Let $I = (G, u, \tau, \gamma, s, t, T)$ be an instance of the generalized maximum flow over time problem.*

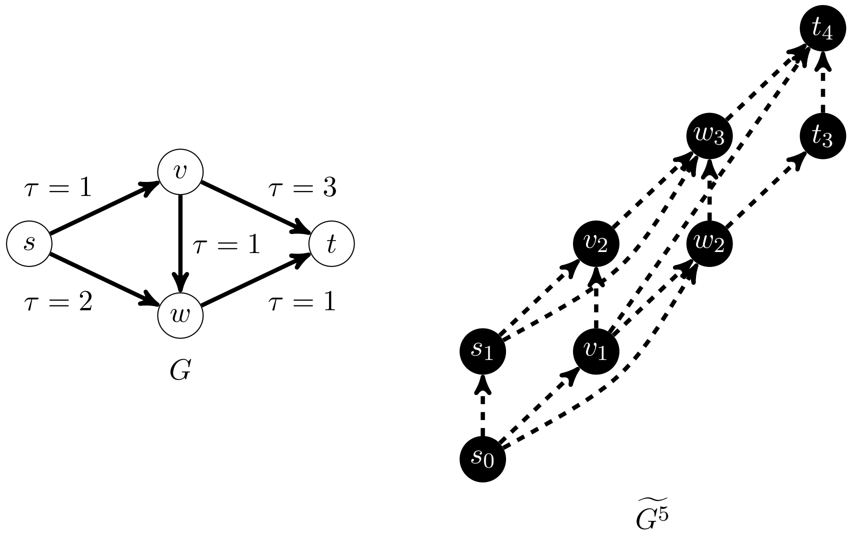


Figure 6.4: A network G and its pruned time-expanded network $\widetilde{G^5}$.

1. *Begin with $i := 0$ and the static zero-flow $x_0 :\equiv 0$.*
2. *If no s-t-path exists in G_{x_i} or if $\tau_{s\to t} \geq T$ in G_{x_i}, then set $k := i - 1$ and go to step 6.*
3. *Restrict the static residual network G_{x_i} to the network G'_{x_i} containing only paths of maximum gain and compute a generalized maximum flow x'_i in G'_{x_i}.*
4. *Define x_{i+1} by adding x'_i to x_i as follows: $(x_{i+1})_a := (x_i)_a + (x'_i)_a + \gamma_a^{-1}(x'_i)_{\overleftarrow{a}}$ for all $a \in A(G)$. Notice that x_{i+1} is a feasible flow in G, since x'_i is a feasible flow in a restricted residual network of x_i.*
5. *Set $i := i + 1$ and go to step 2.*
6. *Construct a generalized flow over time f defined by $f = \sum_{j=0}^{k} \overline{x'_j}$.*

Correctness. We will prove the correctness of Algorithm 6.1 by comparing it to Onaga's algorithm applied to the time-expanded network. For $i = 0, \dots, k{+}1$, define a generalized flow over time $f_i := \sum_{j=0}^{i-1} \overline{x'_j}$. In particular, f_0 is the zero flow over time and $f = f_{k+1}$. The strategy for our proof is to show that in every iteration $\overline{x'_i}$ is a flow along highest gain paths in $G^T_{f_i}$. Then successively adding $\overline{x'_0}, \dots, \overline{x'_k}$ produces the same result as Onaga's algorithm applied to the time-expanded network, showing correctness of our algorithm. The following lemma turns out to be helpful in proving the correctness of Algorithm 6.1.

Lemma 6.2. *For each $i = 0, \dots, k + 1$, it holds that $\overline{G_{x_i}} = \widetilde{G^T_{f_i}}$, i. e., the copied static network is equal to the pruned time-expanded network after each iteration of the algorithm.*

Proof. The proof uses induction on i. For $i = 0$, we get $\overline{G_{x_0}} = \overline{G} = \widetilde{G^T} = \widetilde{G^T_{f_0}}$. Now we assume that $\overline{G_{x_i}} = \widetilde{G^T_{f_i}}$ holds for some i. In iteration i, the algorithm adds $\overline{x'_i}$ to f_i and x'_i to x_i, resulting in f_{i+1} and x_{i+1}, respectively. Due to x'_i being a maximum flow in the highest gain network of G_{x_i} it follows that $\overline{x'_i}$ is a maximum flow in the highest gain network of $\overline{G_{x_i}}$.

Sending x'_i in G_{x_i} causes all copies θG_{x_i} of $\overline{G_{x_i}}$ to be changed as well. These changes due to sending flow are the same that sending $\overline{x'_i}$ in $\overline{G_{x_i}}$ causes. In contrast to $\overline{G_{x_i}}$, $\overline{G_{x_{i+1}}}$ might also gain or lose arcs that either become again part of an s'-t'-path or are no longer part of an s'-t'-path. However, note that both x'_i and $\overline{x'_i}$ are flows in the highest gain network of G_{x_i} and $\overline{G_{x_i}}$, respectively. This implies that sending these flows does not increase the gain of source-node- or node-sink-paths. By proportionality of gains and transit times it follows that the transit time of source-node- or node-sink-paths does not decrease; thus, no arcs are gained this way. Similarly, all arcs lost due to no longer being on an s'-t'-path in $\overline{G_{x_{i+1}}}$ are by definition also removed in $\widetilde{G^T_{f_{i+1}}}$. Thus, $\overline{G_{x_{i+1}}} = \widetilde{G^T_{f_{i+1}}}$. This concludes the proof. □

Theorem 6.3. *Algorithm 6.1 computes a generalized maximum flow over time.*

Proof. Making use of Lemma 6.2 we show that, for $i = k + 1$ (i. e., after the final iteration), there is no s'-t'-path in G^T_f. Since the algorithm terminates, there is no s-t-path in G_{x_i} and therefore no s'-t'-path in $\overline{G_{x_i}} = \widetilde{G^T_f}$. As G^T_f has no more s'-t'-paths then $\widetilde{G^T_f}$ the result follows. Note that our instance has no flow-generating cycles to begin with, due to $c < 0$, and sending flow along highest gain paths does not generate new flow-generating cycles (see Onaga [82, 83]).

Thus, our algorithm starts with the zero flow over time f_0, similar to Onaga's algorithm. In every iteration, flow is only being augmented along highest gain paths in $\overline{G_{x_i}}$ and by Lemma 6.2 this equals $\widetilde{G^T_{f_i}}$ as well. Since this is exactly what Onaga's algorithm does, the correctness of our algorithm follows. Note that Lemma 6.2 also guarantees that f is a feasible generalized flow over time, due to each $\overline{x'_i}$ being a flow in $\overline{G'_{x_i}}$ which is a restricted residual network of $\widetilde{G^T_{f_i}}$. This concludes the proof. □

Running time. We conclude this section by examining the running time of Algorithm 6.1. Let $n := |V(G)|$, $m := |A(G)|$ and $U := \max_{a \in A(G)} u_a$ for a problem instance $(G, u, \tau, \gamma, s, t, T)$. In our case, a highest gain path can be found in $O(nm)$ time using Moore-Bellman-Ford's algorithm (see Bellman [12], Ford [37], Moore [79]) or in $O(m + n \log n)$ by applying Dijkstra's algorithm [24] with Fibonacci heaps (see Fredman and Tarjan [40]) and reduced costs. For both algorithms, $\tau_a = \frac{1}{c} \log \gamma_a$ is being used as a cost function. Both algorithms are capable of computing the highest-gain network as well. A generalized maximum flow in the highest-gain network can then be computed by a standard maximum flow algorithm. Since there are at most T time steps, there can be at most T iterations. The running time of an iteration is dominated by the maximum flow computation, yielding a running time of $O(maxflow \cdot T)$, where $O(maxflow)$ is the running time of the maximum flow algorithm. Orlin [87] described a maximum flow algorithm with a running time of $O(nm)$, resulting in a running time of $O(nm \cdot T)$ for our algorithm.

For special cases, this runtime can be improved further. Beygang, Krumke, and Zeck [14] recently studied static generalized maximum flows in series-parallel networks and discovered that a greedy-strategy that chooses always the highest-gain path in the original – not residual – network is sufficient for finding an optimal solution. This can be carried over to our setting and can be used for bounding the number of paths used. Since each augmentation saturates an arc, there can be at most m iterations, yielding a polynomial-time algorithm.

6.1.2 Turning the Algorithm into an FPTAS

In this section, we will see that Algorithm 6.1 can be terminated early to obtain an approximate solution. In fact, the algorithm can be turned into an FPTAS.

Theorem 6.4. *Let OPT be the value of an optimal solution to a generalized maximum flow over time problem instance $I = (G, u, \tau, \gamma, s, t, T)$, $\varepsilon > 0$, and $U := \max_{a \in A} u_a$. Algorithm 6.1 has found a solution of value at least $OPT - \varepsilon$ after all paths of length $\leq \tau^*$ with*

$$\tau^* := -\tfrac{1}{c}(\log \tfrac{1}{\varepsilon} + \log m + \log U + 2 \log T)$$

have been processed (recall that the lengths of the paths used by the algorithm are monotonically increasing). For a constant $c < 0$ and using a maximum flow based approach as proposed in Chapter 6.1.1, this leads to a running time of $O(maxflow \cdot (\log \varepsilon^{-1} + \log U + \log T))$ for a solution of value at least $OPT - \varepsilon$.

Proof. Algorithm 6.1 computes an optimal solution for the generalized maximum flow over time problem. For $i \in \{0, \ldots, T-1\}$, let a_i denote the amount of flow sent into paths of length (transit time) i in a time step $\theta \in \{0, \ldots, T-i-1\}$ by the algorithm. Note that the algorithm does not use storage; the gain factor of a path of length i is therefore $2^{c \cdot i}$ and flow is sent into a path of length i for $T - i$ time steps. The length of the paths used by Algorithm 6.1 increases monotonically. After all paths of length $\leq i$ have been found, only paths of length $i+1, \ldots, T-1$ remain. The amount of flow arriving at the sink by such paths equals

$$\sum_{j=i+1}^{T-1} a_j \cdot 2^{c \cdot j} \cdot (T-j) \ .$$

Due to $c < 0$ it follows that $\max\left\{2^{c(i+1)}, \ldots, 2^{c(T-1)}\right\} = 2^{c(i+1)}$. Furthermore, it is clear that $T - j \leq T$. The amount of flow sent into paths in one time step is bounded by $m \cdot U$. This yields

$$\sum_{j=i+1}^{T-1} a_j \cdot 2^{c \cdot j} \cdot (T-j) \leq \sum_{j=i+1}^{T-1} m \cdot U \cdot 2^{c(i+1)} \cdot T \leq m \cdot U \cdot 2^{c(i+1)} \cdot T^2 \ .$$

For $i = -\frac{1}{c}(\log \frac{1}{\varepsilon} + \log m + \log U + 2 \log T) - 1$, the right hand side equals ε. □

The above theorem allows an approximation within a constant value ε. For an FPTAS, we need to approximate OPT within a factor of $(1-\varepsilon)$ or a value of εOPT. This can be done by a slight modification of Theorem 6.4.

Theorem 6.5. *Let OPT be the value of an optimal solution for a generalized maximum flow over time problem instance $I = (G, u, \tau, \gamma, s, t, T)$, $\varepsilon > 0$ and $U := \max_{a \in A} u_a$. Algorithm 6.1 has found a solution of value at least $(1-\varepsilon)OPT$ after τ^* iterations with*

$$\tau^* := \lceil -\tfrac{1}{c}(\log \tfrac{1}{\varepsilon} + \log m + \log U + 2\log T) \rceil$$

For a constant $c < 0$ and using a maximum flow based approach as proposed in Chapter 6.1.1, this leads to a running time of $O(maxflow \cdot (\log \varepsilon^{-1} + \log U + \log T))$ for a solution of value at least $(1-\varepsilon)OPT$.

Proof. We use the arguments and notations of the proof of Theorem 6.4. Let i_0 be the length of the first iteration's paths. We assume w.l.o.g that the minimum capacity in the network is 1. Consequently, at least 2^{ci_0} flow is sent in the first iteration. We now want to determine τ^* such that the amount of flow sent to the sink by later iterations is $\leq \varepsilon OPT$.

$$\sum_{j=i_0+\tau^*}^{T-1} a_j \cdot 2^{c \cdot j} \cdot (T-j) \leq m \cdot U \cdot 2^{c(i_0+\tau^*)} \cdot T^2 \leq \varepsilon 2^{ci_0} \leq \varepsilon OPT$$

This inequality can be transformed to

$$m \cdot U \cdot 2^{c\tau^*} \cdot T^2 \leq \varepsilon \quad \Leftrightarrow \quad \tau^* \geq -\tfrac{1}{c}(\log \tfrac{1}{\varepsilon} + \log m + \log U + 2\log T)$$

This concludes the proof. □

7 Complexity

In this chapter, we present our results in the area of complexity theory. First, we show an NP-hardness result for generalized maximum flows over time with arbitrary or lossy gains. This reduction builds on the reduction of Klinz and Woeginger [71]. We will see that this reduction even yields a non-approximability result in terms of flow value for them.

Second, we consider the special case of generalized flows over time with lossy gains, that are also proportional to transit times. Here, we show that maximal flows are also earliest arrival flows – together with the very recent results by Disser and Skutella [25], this should lead to an NP-hardness result for this special case as well. Both results in this chapter have first been published in Groß and Skutella [51].

7.1 Generalized Maximum Flows over Time

In this chapter, we study the complexity of the generalized maximum flow over time problem. We classify the generalized maximum flow over time problem based on the gains used into the cases of

- arbitrary gains $\gamma_a \in \mathbb{R}_{\geq 0}$,
- lossy gains $\gamma_a \in [0, 1]$,
- and lossy and proportional gains $\gamma_a = 2^{c\tau_a}$ for some constant $c < 0$.

The results presented here have first been published in Groß and Skutella [51].

7.1.1 Arbitrary and Lossy Gains

We begin with the cases of arbitrary and lossy gains. Here, we will adapt the reduction of Klinz and Woeginger [71] to the generalized maximum flow over time problem, showing that this problem is weakly NP-hard. In fact, this reduction will yield even a non-approximability result for value-approximations.

Theorem 7.1. *There is neither a polynomial-time algorithm nor a polynomial-time value-approximation algorithm for the generalized maximum flow over time problem, unless* P = NP*. This holds even on series-parallel graphs.*

Proof. Let

$$I_P = (x_1, \dots, x_n), \quad \sum_{i=1}^{n} x_i = 2L, \quad x_1, \dots, x_n, L \in \mathbb{N}$$

be an instance of the PARTITION problem. We define a directed graph G by

$$V(G) := \{v_0, v_1, \ldots, v_{n+1}\},$$
$$A(G) := \underbrace{\{a_i = (v_i, v_{i+1}) \mid i = 1, \ldots, n\}}_{:=A_1} \cup \underbrace{\{a'_i = (v_i, v_{i+1}) \mid i = 1, \ldots, n\}}_{:=A_2} \cup \{(v_0, v_1)\}.$$

Furthermore, we define capacities and transit times by

$$u_a := \begin{cases} 1 & a = (v_0, v_1), \\ \infty & \text{else}, \end{cases} \qquad \tau_a := \begin{cases} x_i & a = a_i \in A_1, \\ 0 & a \in A_2 \cup \{(v_0, v_1)\}, \end{cases}$$

for all $a \in A(G)$. Gain factors γ_a, $a \in A(G)$ are defined to be proportional to the transit times, i. e., $\gamma_a = 2^{c\tau_a}$ for a constant $c > 0$ that we will fix later. Supplies and demands are defined by $b_{v_0} = +\infty$, $b_{v_{n+1}} = -\infty$ and $b_v = 0$ for all other nodes $v \in V(G) \setminus \{v_0, v_{n+1}\}$. The time horizon is $T := L + 1$. Note that the gain factors are encoded implicitly by the transit times; the resulting instance of the generalized maximum flow over time problem is therefore polynomial in the size of the partition instance. Figure 7.1 depicts this instance.

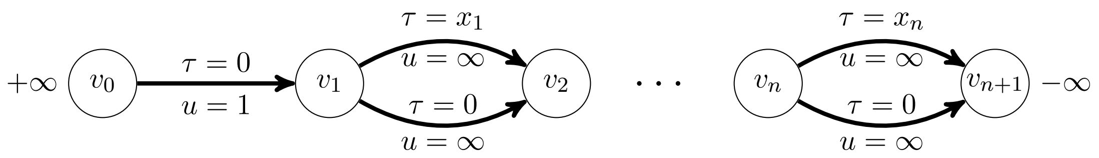


Figure 7.1: A generalized maximum flow over time instance obtained from an instance for the PARTITION problem. Capacities, transit times and supplies and demands are as annotated, gains are implicitly given by transit times via $\gamma \equiv 2^{c\tau}$. Supplies and demands not specified are zero.

YES-instance $\rightarrow$ at least 2^{cL} units flow can be sent. If I_P is a YES-instance with a solution $I \subset \{1, 2, \ldots, n\}$, then there is a generalized flow over time that sends at least 2^{cL} units of flow from v_0 to v_{n+1} within the time horizon T. This is, for example, achieved by sending one unit of flow during the time interval $[0, 1)$ into the path induced by the arc set $\{a_i \mid i \in I\} \cup \{a'_i \mid i \notin I\}$. This path has length L and gain factor 2^{cL} yielding 2^{cL} flow units arriving at the sink in the time interval $[L, L + 1)$.

NO-instance $\rightarrow$ at most $O(2^{c(L-1)})$ units flow can be sent. If I_P is a NO-instance, then there exists no path of length L. Thus, due to the time horizon of $L + 1$, there exists no path with length at least L that is able to carry flow to the sink in time. Flow can leave the source at a rate of at most one of at any point in time $\theta \in [0, L + 1)$, and $2^{c \cdot \min\{L-1, L-\theta\}}$ is the highest gain factor possible for flow leaving the source at time θ. This yields an upper bound of

$$\int_0^1 2^{c(L-1)}\, d\theta + \int_1^{L+1} 2^{c(L-\lfloor\theta\rfloor)}\, d\theta = 2^{c(L-1)} + \sum_{\theta=1}^{L} 2^{c(L-\theta)} = \frac{2^{cL} - 1}{2^c - 1} + 2^{c(L-1)}$$

on the amount of flow that can be sent into the sink in a NO-instance. Notice that, for c sufficiently large, this amount is strictly smaller than the lower bound 2^{cL} on the maximum flow value for a YES-instance. The size of the gap depending on c is:

$$\frac{2^{cL}}{\frac{2^{cL}-1}{2^c-1}+2^{c(L-1)}} = \frac{2^{cL}(2^c-1)}{2^{cL}-1+(2^c-1)2^{c(L-1)}} = \frac{2^c-1}{2-\frac{1}{2^c}-\frac{1}{2^{cL}}} \geq \frac{2^c-1}{2} \ .$$

Thus, we can make the gap arbitrarily large by choosing c appropriately. Therefore any polynomial-time value-approximation algorithm for the generalized maximum flow over time problem is able to solve the NP-hard PARTITION problem. □

A similar result can be shown for lossy networks, i. e., networks where all gain factors are ≤ 1. We can use the same reduction as above, but we define gains γ by

$$\gamma_a := \begin{cases} 1 & a \in A_1 \cup \{(v_0, v_1)\}, \\ 2^{-cx_i} & a = a_i' \in A_2 \end{cases} \qquad \text{for all } a \in A(G)$$

for some constant $c > 0$. Notice that a v_0-v_{n+1}-path with a transit time of θ has a gain of $2^{c(2L-\theta)}$. Using these gains, we can send at least 2^{-cL} units of flow in a YES-instance, using the same argumentation as above. In NO-instances, we can still send flow into source-sink paths with a rate of at most 1 at each point in time. However, the highest gain factor possible for a path used at time $\theta \in [0, L+1)$ is now $2^{-c(2L-\min\{L-1,L-\theta\})}$. This gives us

$$\begin{aligned} \int_0^1 2^{-c(L+1)}\, d\theta + \int_1^{L+1} 2^{-c(L+\lfloor\theta\rfloor)}\, d\theta &= 2^{-c(L+1)} + 2^{-cL}\sum_{\theta=1}^{L} 2^{-c\theta} \\ &= 2^{-c(L+1)} + 2^{-cL}\left(\frac{1-2^{-cL}}{2^c-1}\right) \in O(2^{-c(L+1)}) \end{aligned}$$

as an upper bound for the amount of flow that can be sent. This produces a similar gap as above with at least 2^{-cL} flow for YES-instances and at most $O(2^{-c(L+1)})$ flow for NO-instances.

Note that Theorem 7.1 still holds if we encode a number n as $n = m \cdot 2^e$ and use binary encoding for m and e. This is common when using floating-point numbers, for instance. This allows us to encode a number n that is a power of 2 in space $O(\log\log n)$; it is easy to check that the hardness proof still holds in this case. It is an interesting open question whether the theorem holds when a number n in the input has to be encoded in space $O(\log n)$.

7.1.2 Lossy and Proportional Gains

In this section, we will study the complexity of generalized maximum flow over time problems with lossy and proportional gains, i. e., gains of the form $\gamma \equiv 2^{-c\tau}$ for some constant $c > 0$. Furthermore, we focus on instances with a single source s and sink t with unlimited supply and demand. We will show that under these conditions, all generalized

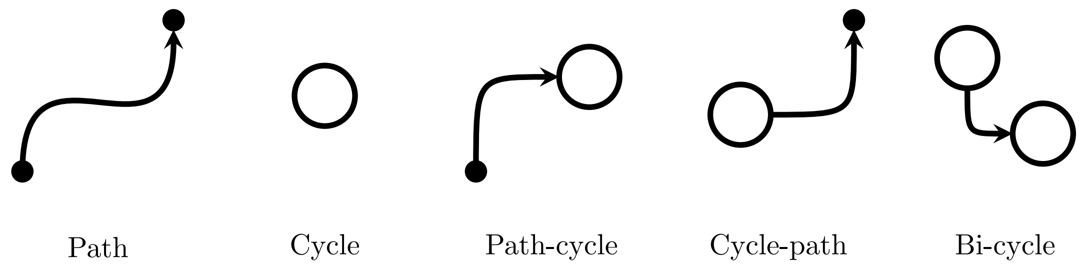


Figure 7.2: The five types of elementary generalized flows.

maximum flows over time share the same arrival pattern if holdover at intermediate nodes is not used, and therefore are earliest arrival flows as well.

Theorem 7.2. *Let $I = (G, u, \tau, \gamma, s, t, T)$ be an instance of the generalized maximum flow over time problem such that $\gamma \equiv 2^{c\tau}$ for some constant $c < 0$. Let f, f' be generalized maximum flows over time for I that do not use holdover in any node except for s and t. Then f and f' have the same arrival pattern.*

Proof. Assume that f and f' fulfill the requirements of the theorem but have different arrival patterns. Consider the time-expanded network G^T and treat f and f' as generalized static flows in G^T. We can define $f - f'$ to be a generalized flow in the residual network $G^T_{f'}$ of f' as follows:

$$(f - f')_a := \begin{cases} \max\{f_a - f'_a, 0\} & a \in A(G^T), \\ \max\left\{(f'_{\overleftarrow{a}} - f_{\overleftarrow{a}}) \cdot \gamma_{\overleftarrow{a}}, 0\right\} & \overleftarrow{a} \in A(G^T), \end{cases} \quad \text{for all } a \in A(G^T_{f'}).$$

Generalized flows can be decomposed into five types of elementary generalized flows according to Theorem 2.2 (see also Figure 7.2). Such an elementary flow g in $G^T_{f'}$ can be added to f' in G^T in the following way:

$$(f' + g)_a := f'_a + g_a - g_{\overleftarrow{a}} \gamma_{\overleftarrow{a}} \quad \text{for all } a \in A(G^T).$$

Thus we can conclude that the decomposition does not contain flows along paths or cycle-paths since they could be added to f' increasing the total flow sent, which would yield a contradiction to the maximality of f'. Similarly, the decomposition cannot contain flows along path-cycles or bi-cycles that contain a copy of the sink in the time expansion, since such a flow would contain a path or a cycle-path which we could use to increase the flow value of f'. All other kinds of flows along path-cycles or bi-cycles cannot affect the arrival pattern when added to f'. Due to our choice of f and f', one of these flows needs to change the arrival pattern, however. This leaves flow along unit-gain cycles as the only possibility left. Here, we can confine ourselves to flows along unit-gain cycles sending flow along holdover or reverse holdover arcs at t – if no form of holdover is used, the arrival pattern does not change due to flow conservation. Notice that such a flow along a unit-gain cycle does not use holdover at s since it would otherwise contain a flow along a path.

We now partition the arcs used by a flow along a unit-gain cycle into the set of all used holdover arcs H, the set of all used reverse holdover arcs R and the set of all other arcs O. This leads to:

$$1 = \gamma_C = \underbrace{\prod_{a \in H} \gamma_a}_{=1} \cdot \underbrace{\prod_{a \in R} \gamma_a}_{=1} \cdot \prod_{a \in O} \gamma_a \quad \Longrightarrow \quad \prod_{a \in O} \gamma_a = 1 \ .$$

Each arc $a \in O$ corresponds to an arc at a certain point in time, each arc $a \in H$ corresponds to waiting one time unit at a node and each arc $a \in R$ corresponds to canceling waiting time at a node for one time unit. The unit-gain cycle C itself must have transit time 0, yielding

$$0 = \tau_C = |H| - |R| + \sum_{a \in O} \tau_a \ .$$

Proportionality of transit times and gain factors give us

$$1 = \prod_{a \in O} \gamma_a = \prod_{a \in O} 2^{c\tau_a} = 2^{c \sum_{a \in O} \tau_a} \quad \Longrightarrow \quad \sum_{a \in O} \tau_a = 0 \quad \Longrightarrow \quad |H| = |R| \ .$$

Thus we know now that flows along unit-gain cycles using holdover / reverse holdover at t must use an equal number of holdover and reverse holdover arcs. We will complete the proof by showing that flow along a unit-gain cycle that uses an equal number of holdover and reverse holdover arcs at t and none at s allows us to reroute flow along a flow-generating cycle in contradiction to the maximality of f'. Figure 7.3 depicts this rerouting.

Let f^* be such a flow along a cycle. By assumption, there is a holdover arc used by f^* to let flow wait in t from time layer α to $\alpha + 1$ and a reverse holdover arc used by f^* to cancel waiting in t from time layer β to $\beta - 1$. There are two possible cases, depending on which happens earlier in time. We will refer to the case where $\alpha < \beta$ as Case 1, the other as Case 2. Figure 7.3 shows these two cases. Due to the fact that holdover arcs at the sink have unlimited capacity, we can reroute the flow by letting the flow wait at t from α to $\beta - 1$ in the first case and β to $\alpha + 1$ in the second case. The resulting cycles are flow-generating, since they use holdover but no reverse holdover; the rest of the cycle must therefore have negative transit time, which implies flow-generation by proportionality of transit times and gain factors. This shows the contradiction and completes the proof. □

If we consider a slightly different model of flow-conservation, where flow can only be safely stored at source and sink and is subject to the same proportional loss when waiting in a node that occurs when traversing arcs, we can give a stronger version of the previous theorem.

Theorem 7.3. *Let $I = (G, u, \tau, \gamma, s, t, T)$ be an instance of the generalized maximum flow over time problem such that $\gamma \equiv 2^{c\tau}$ for some constant $c < 0$ and holdover at a*

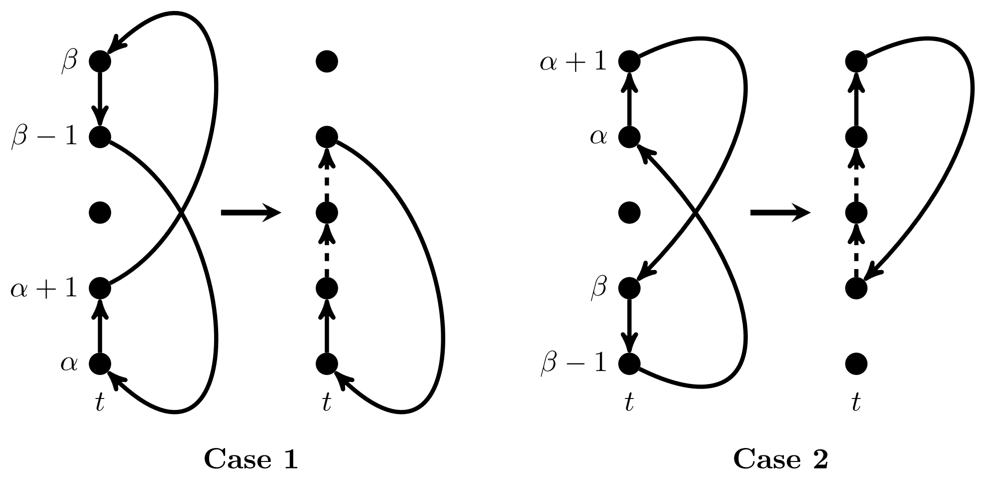


Figure 7.3: Rerouting to construct flow-generating cycles out of unit-gain cycles.

node $v \in V(G) \setminus \{s,t\}$ has a loss factor of 2^c for each time unit spent waiting. Let f, f' be generalized maximum flows over time for I. Then f and f' have the same arrival pattern.

Proof. The arguments of the previous proof hold here as well, if we make a single modification: instead of partitioning the arcs of the flows along unit-gain cycles into holdover, reverse holdover and normal arcs, we partition the arcs now into holdover arcs at the sink, reverse holdover at the sink and all other arcs (i.e. normal arcs and holdover arcs at intermediate nodes). This is due to the fact that holdover arcs in nodes other than source and sink are now allowed, in contrast to the previous theorem. These holdover arcs behave the same as normal arcs in terms of proportional loss and are therefore grouped with them. The argumentation of the previous proof can then be applied to show this claim as well. □

8 Orientations and Flows over Time

In this final chapter, we study the problem of orienting an undirected graph such that the orientation is as beneficial as possible for a subsequent flow over time computation. This chapter is split into two parts. In the first part, we study prices of orientations for various problems. We consider both the flow price of orientation as well as the time price of orientation and hybrid prices which combine these two concepts.

In the second part, we discuss the computational complexity of finding good orientations. Here, we will see several non-approximability results for various contraflow problems. The results in this chapter are an as of yet unpublished work of Arulselvan, Groß and Skutella [7].

8.1 The Price of Orientation

In this chapter, we study the price of orientation. As we discussed before, there are two different ways to pay the price – either in flow value, or by the time horizon. We begin by studying the price of orientation for the maximum contraflow over time and quickest contraflow problems. First, we analyze the flow price of orientation, followed by the time price of orientation. We conclude this chapter by considering paying the price partly in flow value and partly in time horizon, and by discussing prices of orientations for multi-commodity and earliest arrival problems.

8.1.1 Price in Terms of Flow Value

In this subsection, we will examine the flow price of orientation. We will see that orientation can cost us two thirds of the flow value in some instances, but not more. This is assuming that we have multiple sources and sinks with finite supplies and demands.

Theorem 8.1. *Let $N = (G, u, b, \tau, T)$ be an undirected network over time, in which B units of flow can be sent within the time horizon T. Then there exists an orientation $\overrightarrow{N}$ of N in which at least $B/3$ units of flow can be sent within time horizon T.*

Proof. The idea of this proof is to simplify the instance, such that a temporally repeated solution can be found. Such a solution gives us an orientation that we can use, if the simplification does not cost us too much in terms of flow value. We will achieve this by simulating the balances using additional edges and capacities, creating a maximum flow over time problem which permits a temporally repeated solution – these solutions use edges only in a single direction, which is exactly the property that we need (Ford and Fulkerson [39]). Then we show that the resulting maximum flow over time problem is close enough to the original problem for our claim to follow.

Simulating the balances. We achieve this by adding a super-source s and a super-sink t to the network, resulting in an undirected network over time $N' = (G', u', \tau', s, t, T)$ with

$$\begin{aligned} V(G') &:= V(G) \cup \{s, t\}, \\ E(G') &:= E(G) \cup \{\{s, s_+\} \mid s_+ \in S_+\} \cup \{\{s_-, t\} \mid s_- \in S_-\}, \end{aligned}$$

$$u'_e := \begin{cases} u_e & e \in E(G), \\ \infty & \text{else}, \end{cases} \qquad \tau'_e := \begin{cases} \tau_e & e \in E(G), \\ 0 & \text{else}. \end{cases}$$

We refer to the newly introduced edges of $E(G') \setminus E(G)$ as *auxiliary* edges. Furthermore, we sometimes refer to an auxiliary edge by the unique terminal node it is adjacent to and write u_v for $u_e, e = (s, v)$ and so on. An illustration of this construction can be found in Figure 8.1.

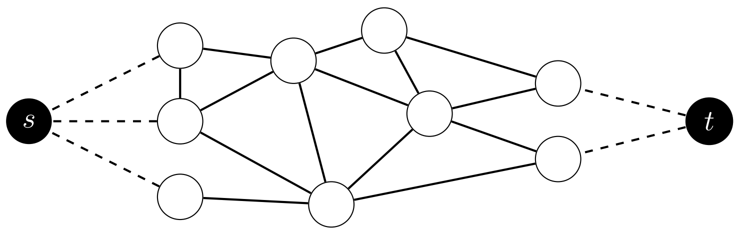


Figure 8.1: The modified network consisting of the original network (white), the super-terminals (black) and the dashed auxiliary edges.

The network N' describes a maximum flow over time problem which has an optimal solution that is a temporally repeated flow, which uses each edge only in one direction during the whole time interval $[0, T)$. Thus, there is an orientation $\overrightarrow{N'}$ such that the value of a maximum flow over time in N' is the same as in $\overrightarrow{N'}$. However, an optimal solution for N' will generally be infeasible for N, since there are no balances in N'.

Thus, we need to modify N' such that balances of N are respected – but without using actual balances. This leaves us the option to modify the capacities of the auxiliary edges. In the next step, we will show that we can always find capacities that enforce that the balances constraints are satisfied and have nice properties for bounding the loss in flow value incurred by the capacity modification. These properties are then used in the last step to complete the proof.

Enforcing balances by capacities for auxiliary edges. In this step, we show that we can choose capacities for the auxiliary edges in such a way that there is a maximum flow over time in the resulting network that respects the original balances. Choosing finite capacities for some of the auxiliary edges will – in general – reduce the maximum flow value that can be sent, though. In order to bound this loss of flow later on, we need capacities with nice properties that can always be found. We make the following claim.

Lemma 8.2. *There are capacities u''_e that differ from u'_e only for the auxiliary edges, such that the network $N'' = (G', u'', \tau', s, t, T)$ has a temporally repeated maximum flow* *over time f with the following properties:*

- *The balances of the nodes in the original setting are respected:*

$$|f_v| := \int_0^T f_v(\theta)\ d\theta = \int_0^T f_{(s,v)}(\theta)\ d\theta \leq |b_v| \quad \forall v \in S_+ \cup S_-,$$

- *and that terminals without tightly fulfilled balances have auxiliary edges with unbounded capacity*

$$|f_v| < |b_v| \Rightarrow u_v = \infty \quad \forall v \in S_+ \cup S_-.$$

Proof. The idea of this proof is to start with unbounded capacities and iteratively modify the capacities based on the balance and amount of flow currently going through a node, until we have capacities satisfying our needs. In order to show that such capacities exist, we apply Brouwer's fixed-point theorem on the modification function to show the existence of a fix point. By construction of the modification function, this will imply the existence of the capacities.

Prerequisites for using Brouwer's fixed point-theorem. We begin by defining $U := B = \sum_{v \in S_+} b_v$ as an upper bound for the capacity of auxiliary edges and we will treat U and ∞ interchangeably from now on. This allows us to consider capacities in the interval $[0, U]$, which is convex and compact, instead of $[0, \infty)$. This will be necessary for applying Brouwer's fixed point theorem later on.

Now assume that we have some capacities $u \in [0, U]^{S_+ \cup S_-}$ for the auxiliary edges. Since we leave the capacities for all other edges unchanged, we identify the capacities for the auxiliary edges with the capacities for all edges. Compute a maximum flow over time $f(u)$ for (G', u, τ', s, t, T) by using Ford and Fulkerson's reduction to a static minimum cost flow. For this proof, we need to ensure that small changes in u result in small changes in $f(u)$, i. e., we need continuity. Thus, we will now specify that we compute the minimum cost flow by using successive computations of shortest s-t-paths. In case there are multiple shortest paths in an iteration, we consider the shortest path graph, and choose a path in this graph by using a depth-first-search that uses the order of edges in the adjacency list of the graph as a tie-breaker. The path decomposition of the minimum cost flow deletes paths in the same way. This guarantees us that we choose paths consistently, leading to the continuity that we need.

Defining the modification function. In order to obtain capacities for a maximum flow over time that respects the balances, we define a function $h : [0, U]^{S_+ \cup S_-} \to [0, U]^{S_+ \cup S_-}$ which will reduce the capacities of the auxiliary edges, if balances are not respected:

$$(h(u))_v := \min \left\{ U, \frac{b_v}{|f_v(u)|} u_v \right\} \quad \forall v \in S_+ \cup S_-.$$

We define $\frac{b_v}{|f_v(u)|}$ to be infinity for $|f_v(u)| = 0$. $|f_v(u)|$ refers to the amount of flow going through the auxiliary edge of terminal $v \in S_+ \cup S_-$, if f is the solution of the maximum flow over time algorithm described above with u as the capacities for the auxiliary edges. Due to our rigid specification in the maximum flow computation, $|f_v(u)|$ is continuous, and therefore h is continuous as well.

Using Brouwer's fixed-point theorem. Thus, h is continuous over a convex, compact subset of $\mathbb{R}^{S_+ \cup S_-}$. By Brouwer's fixed-point theorem h has a fixed point $\overline{u}$ with $h(\overline{u}) = \overline{u}$, meaning that for every $v \in S_+ \cup S_-$ either $\overline{u}_v = U$ or $\overline{u}_v = \frac{b_v}{|f_v(\overline{u})|}\overline{u}_v \Leftrightarrow b_v = |f_v(\overline{u})|$ holds, which is exactly what we require of our capacities. □

We can now choose capacities u'' in accordance to Lemma 8.2, and thereby gain a maximum flow over time problem instance $N'' = (G', u'', \tau', s, t, T)$, that has a temporally repeated optimal solution which does not violate the original balances. What is left to do is to analyze by how much the values of optimal solutions for N and N'' are apart.

Bounding the difference in flow value between N and N''. We now want to show that we can send at least $B/3$ flow units in the network N'' with the auxiliary capacities of the previous step. For the purpose of this analysis, we partition the sources and sinks as follows:

$$S_+^1 := \left\{s_+ \in S_+ \mid u_{s_+} < \infty\right\}, S_+^2 := \left\{s_+ \in S_+ \mid u_{s_+} = \infty\right\},$$
$$S_-^1 := \left\{s_- \in S_- \mid u_{s_-} < \infty\right\}, S_-^2 := \left\{s_- \in S_- \mid u_{s_-} = \infty\right\}.$$

The partitioning is also shown in Figure 8.2.

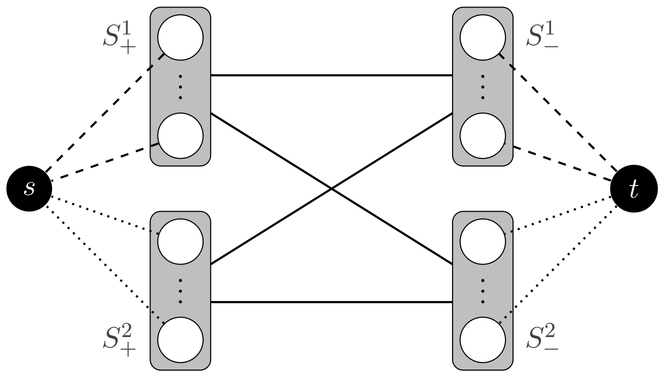


Figure 8.2: The partitioning based on the capacities of the auxiliary edges. Dashed edges have finite capacity, dotted edges have infinite capacities.

Now let f be a temporally repeated maximum flow in N'' that does not violate balances. Notice that the auxiliary edges to terminals in S_+^2 and S_-^2, respectively, have infinite capacity and that the supply / demand of nodes in S_+^1 and S_-^1 is fully utilized. Thus, $|f| \geq \max\left\{b(S_+^1), b(S_-^1)\right\}$.

Should $b(S_+^1) \geq B/3$ or $b(S_-^1) \geq B/3$ hold, we would be done – so let us assume that $b(S_+^1) < B/3$ and $b(S_-^1) < B/3$. It follows that $b(S_+^2) \geq 2/3B$ and $b(S_-^2) \geq 2/3B$ must hold in this case. Now consider the network N' with the terminals of S_+^1 and S_-^1

removed, leaving only the terminals of S_+^2 and S_-^2. We call this network $N'(S_+^2, S_-^2)$. Let $|f'|$ be the value of a maximum flow over time in $N'(S_+^2, S_-^2)$. Since B units of flow can be sent in N (and therefore N' as well), we must be able to send at least $B/3$ units in $N'(S_+^2, S_-^2)$. This is due to the fact that $b(S_+^2) \geq 2/3B, b(S_-^2) \geq 2/3B$ – even if $B/3$ of these supplies and demands were going to S_-^1 and coming from S_+^1, respectively, this leaves at least $B/3$ units that must be send from S_+^2 to S_-^2. Thus, $B/3 \leq |f'|$. Since the capacities of the auxiliary edges of S_+^2 and S_-^2 are infinite, we can send these $B/3$ flow units in N'' as well, proving this part of the claim.

Thus, we have shown that a transshipment over time problem can be transformed into a maximum flow over time problem with auxiliary edges and capacities. If these edges and capacities fulfill the requirements of Lemma 8.2, we can transfer solutions for the maximum flow problem to the transshipment problem such that at least one third of the total supplies of the transshipment problem can be send in the flow problem. Finally, the proof of Lemma 8.2 shows that such capacities do always exist, completing the proof. □

Notice that the algorithm described in the proof is not necessarily efficient – it relies on Brouwer's fixed-point theorem, and finding a Brouwer fixed-point is known to be PPAD-complete [89] and exponential lower bounds for the common classes of algorithms for this problem are known [63]. Since the algorithm is efficient aside from finding a Brouwer fixed-point, our problem is at least not harder than finding a Brouwer fixed-point. Thus, our problem is probably not FNP-complete (with FNP being the functional analogon of NP) as PPAD-completeness indicates that a problem is not FNP-complete [89]. However, it is possible that the fixed-point can efficiently be found for the specific function we are interested in. One problem for finding such an algorithm is however, that changing the capacity of one auxiliary edge does not only modify the amount of flow through its associated terminal but through other terminals as well – and this change in flow value can be an increase or decrease, making monotonicity arguments problematic.

Now we have an upper bound for the flow price of orientation and it turns out that this bound is tight. In order to show this, we can construct a network with three sources and sinks where each source has to send flow to a specific sink (due to capacities and transit times) but the network topology prevents flow from more than one source-sink pair being able to be send in any orientation.

Theorem 8.3. *For any $\varepsilon > 0$, there are undirected networks over time $N = (G, u, b, \tau, T)$ in which B units of flow can be sent, but at most $B/3 + \varepsilon$ units of flow can be sent in any orientation $\overrightarrow{N}$ of N.*

Proof. Consider the family of undirected networks over time $N_{\delta,\varepsilon} = (G, u, b, \tau, T + \varepsilon)$ depicted in Figure 8.3, for some $\varepsilon > 0, \delta \in (0, 1)$. For $\varepsilon \leq \delta T$, we cannot send flow from s_1 to t_2 within the time horizon, and we can only send ε/T flow from s_1 to t_1. Thus, we have to orient $\{v_3, v_4\}$ as (v_3, v_4) or lose the supply of s_1 in the case of $\varepsilon \to 0$. Orienting $\{v_3, v_4\}$ as (v_3, v_4) causes us to lose the demands of t_1 and t_2, though, resulting in only one third of the flow being able to be sent.

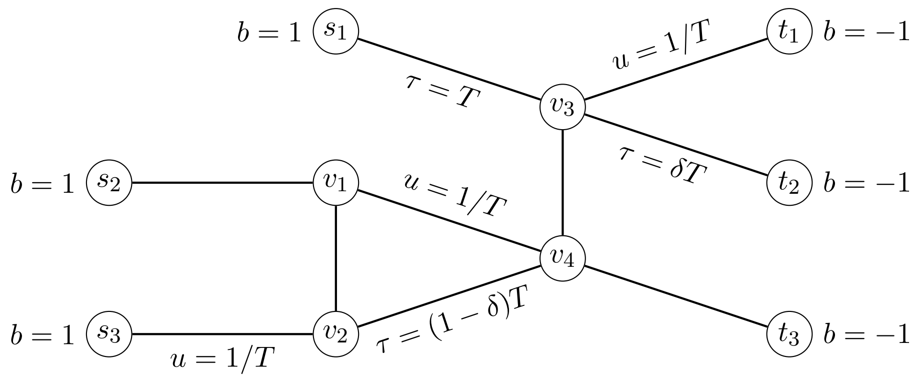


Figure 8.3: The undirected network over time $N_{\delta,\varepsilon}$. Not specified transit times and balances are 0 and not specified capacities are infinite.

Therefore let us now orient $\{v_3, v_4\}$ as (v_4, v_3). Supply from s_3 needs to go through $\{s_3, v_2\}$ at a rate of at most $1/T$. Thus, if we were to route flow through $\{s_3, v_2\}$ and $\{v_2, v_4\}$ we can send at most $(T + \varepsilon - (1-\delta)T)/T = \varepsilon/T + \delta$ to v_4 (and the sinks) within the time horizon. For $\delta, \varepsilon \to 0$ this converges to 0 as well. Since we already lost the supply of s_1, we need the supply of s_3 if we want to send significantly more than one unit of flow. Therefore, we would have to orient $\{v_1, v_2\}$ as (v_2, v_1) to accomplish this. However, due to the capacity of $1/T$ on $\{v_1, v_4\}$ we can send at most $1 + \varepsilon/T$ flow through this edge, and one unit of this flow comes from s_3, leaving only ε/T units for flow from s_2. Thus, for $\delta, \varepsilon \to 0$ the flow we can send converges to one.

In the undirected network, we can send all supplies. The supply from s_1 is sent to t_3, using $\{v_3, v_4\}$ at time T. The supply from s_2 is sent to t_2, via $\{v_1, v_2\}$ at time 0, $\{v_2, v_4\}$ and $\{v_4, v_3\}$ at time $(1-\delta)T$. The supply from s_3 is sent to t_1 by $\{v_2, v_1\}$, $\{v_1, v_4\}$ and $\{v_4, v_3\}$ during the time interval $(0, T)$. This completes the proof. □

With these theorems, we have a tight bound for the flow price of orientation in networks with arbitrarily many sources and sinks. In the case of a single source and sink, we have a maximum flow over time problem and we can always find an orientation in which we can send as much flow as in the undirected network (Ford and Fulkerson [39]). This leaves the question about networks with either a single source or a single sink open. However, if we use the knowledge that only one source (or sink) exists in the analysis done in the proofs of Theorem 8.1 and Theorem 8.3, we achieve a tight factor of 2 in these cases.

Theorem 8.4. *Let $N = (G, u, b, \tau, T)$ be an undirected network over time with a single source or sink, in which B units of flow can be sent within the time horizon T. Then there exists an orientation $\overrightarrow{N}$ of N in which at least $B/2$ units of flow can be sent within time horizon T, and there are undirected networks over time for which this bound is tight.*

Proof. For this proof, we can use most of the argumentation of the proof of Theorem 8.1. The differences start only in the last part, where the differences in flow value between the original network N and the network with capacitated auxiliary edges N'' is considered. In the proof of Theorem 8.1, we partitioned the sources and sinks, but now we have either a single source or a single sink which does not need to be partitioned. Let us assume now that we have a single sink, the case with a single source follows analogously. We partition the sources as follows.

$$S_+^1 := \left\{ s_+ \in S_+ \mid u_{s_+} < \infty \right\}, S_+^2 := \left\{ s_+ \in S_+ \mid u_{s_+} = \infty \right\},$$

We assume $b(S_+^1) < B/2$, because otherwise there is nothing to show. This implies that $b(S_+^2) \geq B/2$, however. We can now consider the network N' with the sources S_+^1 removed and refer to the resulting network as $N'(S_+^2, S_-)$. Since B units of flow can be sent in N (and therefore N' as well), we must be able to send at least $b(S_+^2)$ units in $N'(S_+^2, S_-)$, since we still have all sinks available. Because of $b(S_+^2) \geq B/2$, this proves the first part of the claim. For the second part, the lower bound, consider the construction from Theorem 8.3.

If we restrict the network described there to s_2, s_3, v_1, v_2, v_4 and set the balance of v_4 to -2, we can apply the same argumentation as in Theorem 8.3 to get a proof for the case of a single sink. For the case of a single source, we can do something similar, but we have to perform more changes. The result can be seen in Figure 8.4; the argumentation is analogous to Theorem 8.3. □

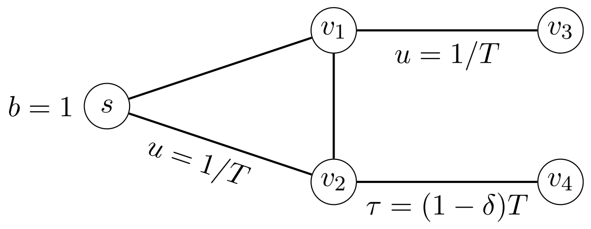


Figure 8.4: An undirected network with a single source where every orientation can send at most one half of the flow possible in the undirected setting. Not specified transit times and balances are 0 and not specified capacities are infinite.

8.1.2 Price in Terms of the Time Horizon

Now that we have tight bounds for the flow price of orientation with single and multiple sources and sinks, let us now consider the time price of orientation. Here, we examine by how much we need to extend the time horizon in order to send as much flow in a favorable orientation as in the undirected network. It turns out that there are instances for which we have to increase the time horizon by a factor that is linear in the number of nodes. This is due to the fact that we have to send everything, which can force us to send some flow along very long detours – this is similar to what occurs in [50]. For this reason it is not a good idea to pay the price of orientation in time alone.

Theorem 8.5. *There are undirected networks over time $N = (G, u, b, \tau, T)$ with either a single source or a single sink in which B units of flow can be sent within a time horizon* *of T, but it takes a time horizon of at least $(n-1)/4 \cdot T$ to send B units of flow in any orientation $\overrightarrow{N}$ of N. This bound also holds if G is a tree with multiple sources and sinks.*

Proof. We define a family of undirected networks over time N_k by

$$\begin{aligned} V(N_k) &:= \{s_0, v_0, v_k, t_k\} \cup \{\, s_i, t_i, v_i, w_i \mid i = 1, \dots, k-1\}\,, \\ E(N_k) &:= \{\{s_0, v_0\}, \{v_{k-1}, v_k\}, \{v_k, t_k\}\} \\ &\qquad \cup \{\, \{s_i, w_i\}, \{t_i, w_i\}, \{w_i, v_i\}, \{v_i, v_{i-1}\} \mid i = 1, \dots, k-1\}\,. \end{aligned}$$

We define capacities, transit times and balances for this network by

$$u_e := \begin{cases} (nT)^{i-1} & e = \{w_i, t_i\}\,, \\ \infty & \text{else,} \end{cases} \qquad \tau_e := \begin{cases} T & e = \{v_i, v_{i-1}\}\,, \\ 0 & \text{else,} \end{cases}$$

$$b_v := \begin{cases} (nT)^i & v = s_i, \\ -\sum_{i=0}^{k-1} (nT)^i & v = t, \\ 0 & \text{else.} \end{cases}$$

Figure 8.5 depicts such a network N_k. It is possible to fulfill all supplies and demands in

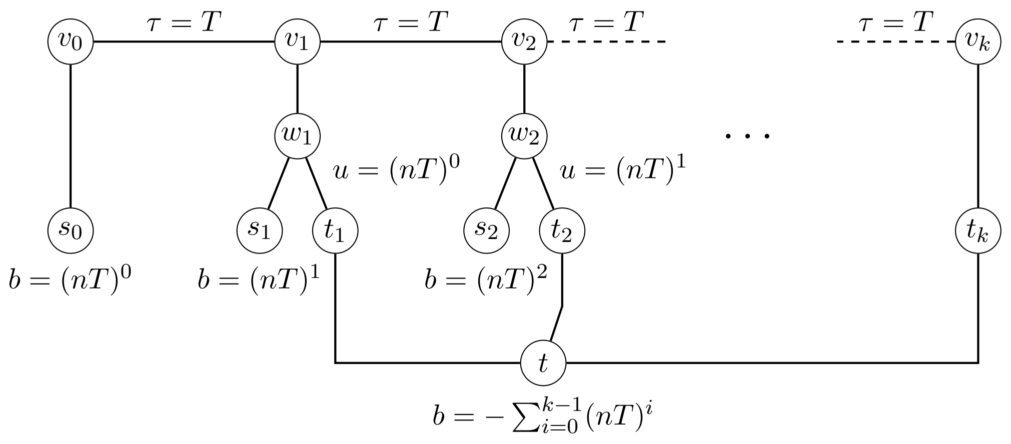


Figure 8.5: An undirected network with a single sink where every orientation requires a time horizon that is larger by a factor of at least $(n-1)/4$ compared to the undirected setting. Not specified transit times and balances are 0 and not specified capacities are infinite.

time $T+1$ in the undirected network, if we route the supply of source s_i through v_i, v_{i+1}, w_{i+1} and t_{i+1} to t. However, this requires using the $\{v_i, w_i\}$-edges in both directions. If we orient a $\{v_i, w_i\}$ edge as (v_i, w_i), we can only route the supply of s_i via w_i and t_i to t, which requires nT time units, due to the supply of s_i and the capacity of $\{w_i, t_i\}$. If we orient all $\{v_i, w_i\}$ edges as (w_i, v_i), we have to route the supply from s_0 via $v_0, v_1, \dots, v_k$ and t_k to t, which requires kT time units. By construction of the network, we have $k = (n-1)/4$, which proves the claimed factor.

A similar construction can be employed in networks with a single source and multiple sinks (see Figure 8.6). If we want to show the result for graphs G that are trees, we can remove t and shift the demand to the nodes t_i, $i = 1, \dots, k$ and give node t_i a demand of $-(nT)^{i-1}$. □

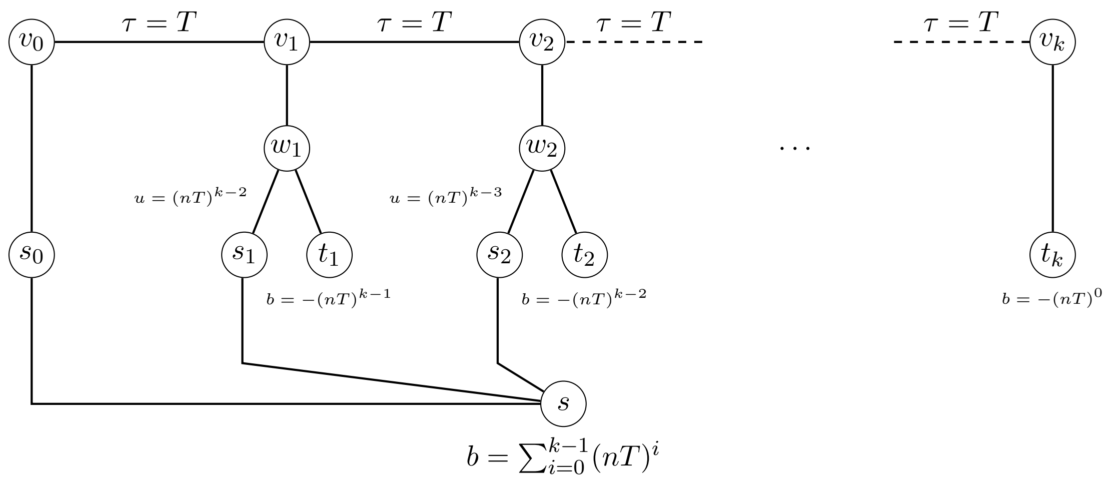


Figure 8.6: An undirected network with a single source where every orientation requires a time horizon that is larger by a factor of at least $(n-1)/4$ compared to the undirected setting. Not specified transit times and balances are 0 and not specified capacities are infinite.

8.1.3 Price in Terms of Flow and Time Horizon

We have seen now that the price of orientation is 3 with regard to the flow value, and $\Omega(n)$ with regard to the time horizon. We can improve on these bounds if we allow to pay the price of orientation partly in terms of flow value and partly in terms of the time horizon. This is possible by combining the reduction to maximum flows over time from Theorem 8.1 with the concept of temporally averaged flows (see, e. g., Fleischer and Skutella [32]).

Theorem 8.6. *Let $N = (G, u, b, \tau, T)$ be an undirected network over time, in which B units of flow can be sent within the time horizon T. Then there exists an orientation $\overrightarrow{N}$ of N in which at least $B/2$ units of flow can be sent within time horizon $2T$. The orientation and a transshipment over time with this property can be obtained in polynomial time.*

Proof. In order to prove this claim, we will create a modified network with a larger time horizon in which we can send a temporally repeated flow which uses each edge in only one direction. This gives us then an orientation with the desired properties. Consider the network $N' = (G', u', b', \tau', 2T)$ defined by

$$V(G') := V(G) \cup \{s, t\}\,, \quad E(G') := E(G) \cup \{\{s, v\} \mid b_v > 0\} \cup \{\{v, t\} \mid b_v < 0\}\,,$$

$$u'_e := \begin{cases} \frac{b_v}{T} & e = \{s, v\}\,, \\ \frac{-b_v}{T} & e = \{v, t\}\,, \\ u_e & \text{else,} \end{cases} \quad \tau'_e := \begin{cases} \tau_e & e \in E(G), \\ 0 & \text{else,} \end{cases} \quad b'_v := \begin{cases} 0 & v \in V(G), \\ B & v = s, \\ -B & v = t. \end{cases}$$

An illustration can be found in Figure 8.7. We know that there is a transshipment over

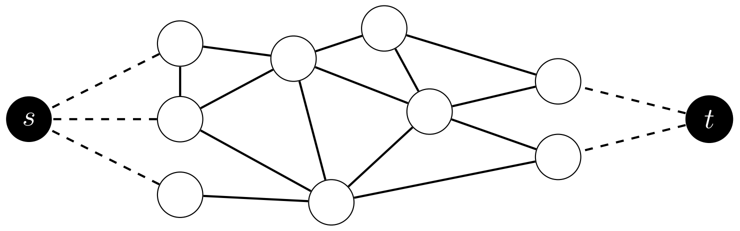


Figure 8.7: The modified network consisting of the original network (white) and the newly introduced nodes (black) and auxiliary edges (dashed).

time f that sends B flow units within time T in N. We can decompose this transshipment into flow along a family of paths $\mathcal{P}$ with $\tau_P < T$ for all $P \in \mathcal{P}$ and interpret f as sending flow into paths $P \in \mathcal{P}$ at a rate of $f_P(\theta)$ at time θ. Now consider a transshipment over time f' that is defined by sending flow into the same paths as f, but at an averaged rate of $f'_P(\theta) := \frac{1}{T} \int_0^T f_P(\xi)\, d\xi$ for a path P and a time $\theta \in [0, T)$. Since all paths $P \in \mathcal{P}$ have $\tau_P < T$, f' sends its flow within a time horizon of $2T$. f' sends B flow units as well, since we just averaged flow rates and the averaging guarantees that the capacities of the edges $e \in E(G') \setminus E(G)$ are not violated. We conclude that a maximum flow over time in $N'' := (G', u', \tau', s, t, 2T)$ has a value of at least B.

Now we compute a maximum flow over time in N'' using the Ford-Fulkerson algorithm [39]. This algorithm computes a temporally repeated maximum flow over time f'' which uses each edge in only one direction. We can transform f'' into a transshipment over time f^* for N by cutting off the edges of $E(G') \setminus E(G)$. Due to $u'_{\{s,v\}} = b_v/T$, $u'_{\{v,t\}} = -b_v/T$ and the time horizon of $2T$, the resulting flow over time f^* satisfies supplies and demands b'' with $0 \leq b''_v \leq 2b_v$ for $v \in V(G)$ with $b_v > 0$ and $0 \geq b''_v \geq 2b_v$ for $v \in V(G)$ with $b_v < 0$. Thus, $1/2 f^*$ sends at least $B/2$ flow units in $2T$ time and uses each edge in only one direction without violating the balances b. Furthermore, this can be done in polynomial time, since the transformation and the Ford-Fulkerson algorithm are polynomial. This concludes the proof. $\square$

8.1.4 Price in Trees

We have seen that in general networks there is always a price of orientation to be paid. This raises the question whether there are families of networks where we do not have to pay a price of orientation – i. e., networks where the price of orientation is 1. However, as the next example shows, even very simple families of networks like trees have prices of orientation greater than 1.

Theorem 8.7. *There are undirected networks over time N in which B units of flow can be sent within the time horizon T and the underlying graph G is a tree but any orientation $\overrightarrow{N}$ of N can send at most $B/2$ units of flow within time horizon T.*

Proof. Consider the instance depicted in Figure 8.8 with a time horizon of $T+1$. In the undirected network, we can send flow from b to d and e to a within the time horizon by using $\{b, c\}$ in both directions. In any orientation, we either send only 1 flow unit or we require a larger time horizon, as we have to route flow from e to d if we want to use both sinks. □

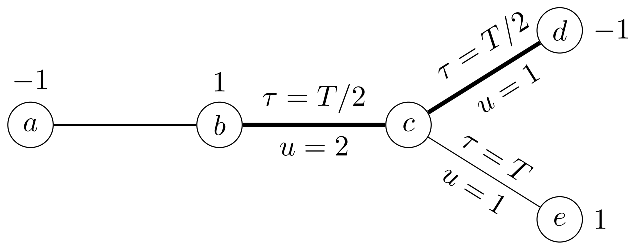


Figure 8.8: A tree network where the value price of orientation is 2.

Notice that we can use the construction in Figure 8.6 to show a time price of orientation of $\Omega(n)$ for trees, as we can remove the super-source from the network and shift the supplies away from the super-source.

8.1.5 Price for Earliest Arrival Flows

We now have tight bounds for the flow and time price of orientation for maximum or quickest flows over time. However, for application in evacuations, it would be nice if we could analyze the price of orientation for earliest arrival flows as well, as they provide guarantees for flow being sent at all points in time. Unfortunately, we can create instances where not even approximate earliest arrival contraflows exist, because the trade-off between different orientations becomes too high.

Theorem 8.8. *There are undirected networks over time $N = (G, u, b, \tau)$ for which an earliest arrival flow exists, but that do not allow for an earliest arrival contraflow. This also holds for α-time- and β-value-approximative earliest arrival contraflows for $\alpha < T/4$ and $\beta < U/2$, where T and U are the largest transit time and capacity in the network, respectively.*

Proof. Consider the network depicted in Figure 8.9. We can orient the edge $\{v_1, v_2\}$ as (v_2, v_1) and have flow arriving with a rate of U starting at time T. However, we have no flow arriving before time $T/2+1$ using this orientation. If we use the orientation (v_1, v_2) instead, we can have flow arrive at time 2, but at a rate of 1 instead of U, and the flow

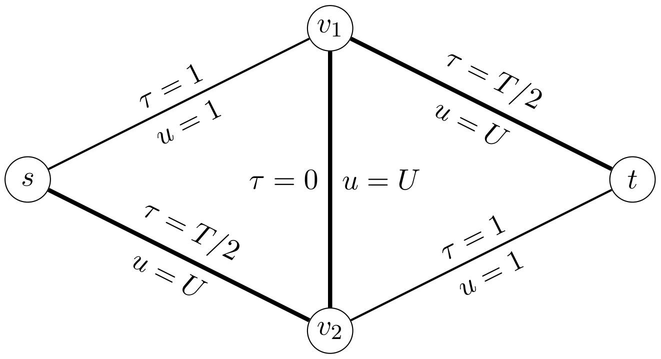


Figure 8.9: An undirected network with source s and sink t and capacities and transit times as specified.

rate can increase to not more than 2 after time $T/2+1$. For $U \gg T$, this trade-off makes it impossible to find an earliest arrival contraflow.

For α-time-approximative earliest arrival flows, we can choose $U = 2(T^2+T)+2$. This yields an instance where no $\alpha < T/4$-approximation is possible. Using the orientation (v_1, v_2), we cannot send U flow units before time T^2+T. However, using the orientation (v_2, v_1), we could have sent them by time $T + 1$. Sending flow at a rate of U using the orientation (v_2, v_1) results in no flow units being sent until time $T/2 + 1$, but flow could have been sent as early as time 2 using the other orientation. This yields the non-approximability result for α-time-approximations.

For β-value-approximations, we need to use the orientation (v_1, v_2) to have some flow arrive starting at time 2. However, using the other orientation allows us to send flow at a rate of U, yielding a ratio that converges to U compared to a rate of 2 for the (v_1, v_2) orientation, which concludes the proof. □

8.1.6 Price for Multi-commodity Flows

Another interesting setting is to consider the price of orientation in multi-commodity flows. In the case of multi-commodity flows we will usually not be able to send as much flow in an oriented network as we can in the unoriented one, as the following theorem shows.

Theorem 8.9. *There are undirected networks over time $N = (G, u, b, \tau, T)$ with multiple commodities for which B flow units can be sent within a time horizon T but any orientation $\overrightarrow{N}$ of N cannot send B flow units even with infinite time.*

Proof. Consider the instance depicted in Figure 8.10. We can orient $\{v, w\}$ as either (v, w) or (w, v) and in each case we lose all the flow of one of the two commodities. In the undirected network they can use $\{v, w\}$ one after the other but this is not possible in any orientation. □

Thus, we have to expect that we will lose flow from some commodities in an orientation. In fact, we may lose all flow of an specific commodity (as we have seen in Figure 8.10), so

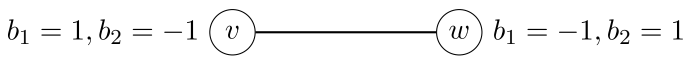


Figure 8.10: An undirected network with two commodities where no orientation can send both commodities.

the price of orientation for the concurrent multi-commodity flow problem is unbounded. On the other hand, we can bound the price of orientation for the (maximum) multi-commodity flow problem by picking the commodity with the most supplies, discard all other commodities and solve the single commodity version of the problem for the remaining commodity. By Theorem 8.1, this guarantees a price of orientation of $3k$, where k is the number of commodities. The next theorem gives a lower bound on the flow price of orientation for this problem.

Theorem 8.10. *There are undirected networks over time $N = (G, u, b, \tau, T)$ with multiple commodities for which B flow units can be sent within a time horizon T but any orientation $\overrightarrow{N}$ of N can send at most $B/(3k)$ flow units within time T.*

8.2 Complexity

In this final section, we study the computational complexity of various contraflow problems. We begin with the quickest contraflow problem, where time-approximations better than 2 are not possible, unless $\mathsf{P} = \mathsf{NP}$.

Theorem 8.11. *The quickest contraflow problem cannot be time-approximated better than a factor of* 2*, unless* $\mathsf{P} = \mathsf{NP}$*.*

Proof. Rebennack et al. [96] showed the NP-hardness of this problem. Their reduction technique can also be used to show a non-approximability result, if we modify the transit times used in their reduction. We give a brief sketch of their reduction technique, which is based on the SAT problem. We construct an instance for the quickest contraflow problem from an instance for the 3SAT problem with ℓ clauses $c_1, \dots, c_\ell$ over k variables $x_1, \dots, x_k$ as follows.

- For each clause c_i, we create a source c_1^+ and a sink c_1^- with a supply and demand of 1 and -1, respectively.
- For each variable x_i, we create four nodes: x_i^1 and x_i^2 for its unnegated literal, $\bar{x}_i^1$ and $\bar{x}_i^2$ for its negated literal. These nodes get neither supplies nor demands. Furthermore, we create a source s_i and a sink t_i with a supply and demand of 1, respectively. Finally, we create edges $\{x_i^1, x_i^2\}$, $\{\bar{x}_i^1, \bar{x}_i^2\}$, $\{s_i, x_i^2\}$, $\{s_i, \bar{x}_i^2\}$ with a transit time of 0 and edges $\{t_i, x_i^1\}$, $\{t_i, \bar{x}_i^1\}$ with a transit time of 1.
- For each clause $c_i = x_{i_1} \vee x_{i_2} \vee \bar{x}_{i_3}$ we create edges $\{c_i^+, x_{i_1}^1\}, \{c_i^+, x_{i_2}^1\}, \{c_i^+, \bar{x}_{i_3}^1\}$ with a transit time of 1 and edges $\{c_i^-, x_{i_1}^2\}, \{c_i^-, x_{i_2}^2\}, \{c_i^-, \bar{x}_{i_3}^2\}$ with a transit time of 0.

All capacities are infinite. Figure 8.11 depicts such a construction.

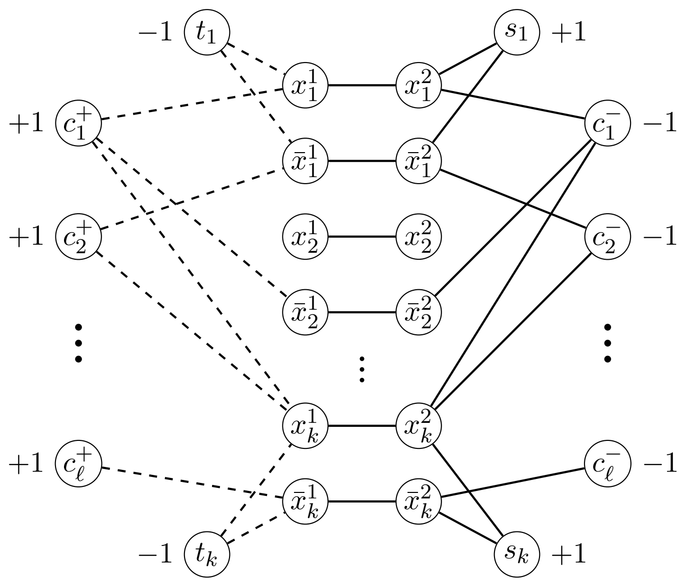


Figure 8.11: A quickest contraflow instance derived from a SAT instance. Edges with a transit time of 1 are dashed, edges with a transit time of 0 are solid. s_2 and t_2 and several clause-edges are not shown.

YES-Instance $\rightarrow$ Routable in time 1. We derive an orientation from an assignment for the SAT problem that fulfills all clauses. If variable x_i is set to 1 in the assignment, we orient $\{x_i^1, x_i^2\}$ as (x_i^1, x_i^2) and $\{\bar{x}_i^1, \bar{x}_i^2\}$ as $(\bar{x}_i^2, \bar{x}_i^1)$. Otherwise, we orient $\{x_i^1, x_i^2\}$ as (x_i^2, x_i^1) and $\{\bar{x}_i^1, \bar{x}_i^2\}$ as $(\bar{x}_i^1, \bar{x}_i^2)$. All other edges are oriented away from the sources or towards the sinks, respectively. A clause source c_i^+ with a fulfilled literal x_i can send 1 flow unit along $c_i^+ \rightarrow x_i^1 \rightarrow x_i^2 \rightarrow c_i^-$, and each variable source s_i can send 1 flow unit to its sink via $s_i \rightarrow x_i^2 \rightarrow x_i^1 \rightarrow t_i$ if $x_i = 0$ and $s_i \rightarrow \bar{x}_i^2 \rightarrow \bar{x}_i^1 \rightarrow t_i$ otherwise. This takes 1 time unit.

NO-Instance $\rightarrow$ Not routable in time < 2. If we want to send everything in a time < 2, we can only use paths containing at most one edge with a transit time of 1. It follows that supply from the clause sources needs to go to a clause sink, via an (x_i^1, x_i^2) or $(\bar{x}_i^1, \bar{x}_i^2)$ edge. Similar, each variable sink t_i needs to get its flow from a variable source and requires an (x_i^2, x_i^1) or $(\bar{x}_i^2, \bar{x}_i^1)$ oriented edge, if we want to be faster than 2. Having both edges oriented as (x_i^2, x_i^1) and $(\bar{x}_i^2, \bar{x}_i^1)$ does not help more than having only one of them oriented that way – we will now assume without loss of generality, that only one of the edges is oriented that way. We can derive an assignment from the orientation of these edges. If we have (x_i^2, x_i^1) in our orientation, we set $x_i = 0$ and $x_i = 1$ otherwise. However, no assignment fulfills all clauses, therefore we have to send clause supplies to variable demands, which takes 2 time units.

Thus, if we are able to time-approximate the quickest contraflow problem within a factor of 2, we can distinguish between YES and NO instances of the 3SAT problem. $\square$

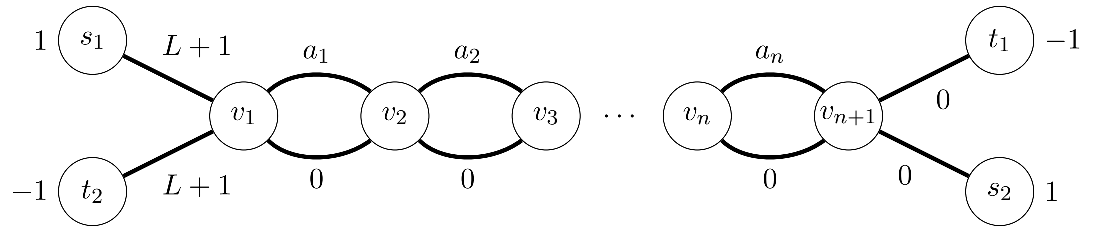


Figure 8.12: An instance of the maximum contraflow over time problem, derived from a PARTITION instance. All capacities are one, transit times are as specified on the edges.

We can show a similar result for the maximum contraflow over time problem. Here, we use a reduction from PARTITION similar to the reduction used by Klinz and Woeginger [71] for minimum cost flows over time.

Theorem 8.12. *The maximum contraflow over time problem cannot be approximated by value better than a factor of* 2*, unless* $\mathsf{P} = \mathsf{NP}$*.*

Proof. The following reduction is inspired by Klinz and Woeginger [71]. Consider an instance of the PARTITION problem, given by integers $a_1, \ldots, a_n$ with $\sum_{i=1}^{n} a_i = 2L$ for some integer $L > 0$. We create an instance for the maximum contraflow over time problem as follows:

- We create $n+1$ nodes $v_1, \ldots, v_{n+1}$, two sources s_1, s_2 and two terminals t_1, t_2. The sources have each a supply of 1, the sinks each a demand of -1.
- We create $2n$ edges $e_i = \{v_i, v_{i+1}\}$, $e_i' = \{v_i, v_{i+1}\}$ for $i = 1, \ldots, n$ with a transit time of a_i for e_i and a transit time of 0 for e_i', edges $\{s_1, v_1\}$, $\{t_2, v_1\}$ with a transit times of $L+1$ and edges $\{s_2, v_{n+1}\}$, $\{t_1, v_{n+2}\}$ with a transit times of 0. All edges have unit capacities.
- We set the time horizon to $2L + 2$.

The resulting instance is depicted in Figure 8.12.

YES-Instance → total flow value of 2. If the PARTITION instance is a YES-instance, there exists a subset of indices $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} A_i = L$. Thus, we can orient the edges so that two disjoint v_1-v_{n+1}- and v_{n+1}-v_1-paths with a length of L are created, which gives us two disjoint paths from s_1 to t_1 and s_2 to t_2 with transit time $2L + 1$, respectively, that are sufficient to send all supplies within the time horizon of $2L + 2$.

NO-Instance → total flow value of 1. Notice that we cannot send any flow from s_1 to t_2 within the time horizon of $2L + 2$. If the PARTITION instance is a NO-instance, then we cannot get a v_1-v_{n+1}- and a v_{n+1}-v_1-path with a length L, so only the demands of one of the two demands can be fulfilled. □

Clearly, the results of Theorem 8.11 and 8.12 carry over to case of multiple commodities as well. However, for multiple commodities, we can show even stronger results for some problems. We begin with the maximum multi-commodity concurrent contraflow over time problem, where we can show that it is not approximable by time or value, unless $\mathsf{P} = \mathsf{NP}$.

Theorem 8.13. *The maximum multi-commodity concurrent contraflow over time problem cannot be approximated by time or value, unless* $\mathsf{P} = \mathsf{NP}$. *This holds even in the case with zero transit times.*

Proof. Consider an instance of 3SAT, given by a set of ℓ clauses $C = \{c_1, \dots, c_\ell\}$ on k variables $x_1, \dots, x_k$. We create a corresponding instance of the maximum multi-commodity concurrent contraflow over time problem as follows:

- For each clause c_i we create a node c_i,
- for each variable x_i, create nodes x_i^1, x_i^2, $\overline{x}_i^1$, $\overline{x}_i^2$,x_i^-,$\overline{x}_i^-$, d_i^-, $\overline{d}_i^-$ and d_i^+.
- For a clause $c_i = x_{i_1} \vee x_{i_2} \vee \overline{x}_{i_3}$ we create edges $\{c_i, x_{i_1}^1\}$, $\{c_i, x_{i_2}^1\}$ and $\{c_i, \overline{x}_{i_3}^1\}$,
- for each variable x_i, create edges $\{d_i^+, x_i^2\}$, $\{d_i^+, \overline{x}_i^2\}$, $\{x_i^1, d_i^-\}$, $\left\{\overline{x}_i^1, \overline{d}_i^-\right\}$, $\{x_i^2, x_i^-\}$, $\{\overline{x}_i^2, \overline{x}_i^-\}$, $\{x_i^1, x_i^2\}$ and $\{\overline{x}_i^1, \overline{x}_i^2\}$.
- Capacities are set to ℓ for each edge and transit times to 0.
- There is a commodity for each variable x_i, with a supply of 2 at d_i^+ and demands of -1 at d_i^-, $\overline{d}_i^-$. Furthermore, there is a commodity for each clause $c_i = x_{i_1} \vee x_{i_2} \vee \overline{x}_{i_3}$, with a supply of 3 at the clause node c_i and a demand of -1 at $x_{i_1}^-$, $x_{i_2}^-$ and $\overline{x}_{i_3}^-$.

Notice that the resulting network – an example of which is depicted in Figure 8.13 – has $\ell + 9k$ nodes and $3\ell + 8k$ edges, which is polynomial in the size of the 3-SAT instance.

YES-Instance → $\frac{1}{3}$-concurrent flow value. If the 3SAT instance is a YES-instance, then there is a variable assignment $x_i \in \{0,1\}$, $i = 1, \dots, k$ fulfilling all clauses. We use this assignment to define an orientation of the edges in our network. If a variable x_i is assigned a value of 0, we orient the edge $\{x_i^1, x_i^2\}$ as (x_i^2, x_i^1) and $\{\overline{x}_i^1, \overline{x}_i^2\}$ as $(\overline{x}_i^1, \overline{x}_i^2)$; if x_i is assigned the value 1, we orient edge $\{\overline{x}_i^1, \overline{x}_i^2\}$ as $(\overline{x}_i^2, \overline{x}_i^1)$ and $\{x_i^1, x_i^2\}$ as (x_i^1, x_i^2). All other edges are oriented away from sources and towards sinks. Notice that:

- Each clause commodity of a clause $c_i = x_{i_1} \vee x_{i_2} \vee \overline{x}_{i_3}$ can send flow to the nodes $x_{i_1}^1, x_{i_2}^1, \overline{x}_{i_3}^1$.
- Each clause is satisfied by our assignment, so there is a literal in each clause that is true.
- For this literal x_i, there is an edge directed from x_i^1 to x_i^2 (or $\overline{x}_i^1$ to $\overline{x}_i^2$, respectively).

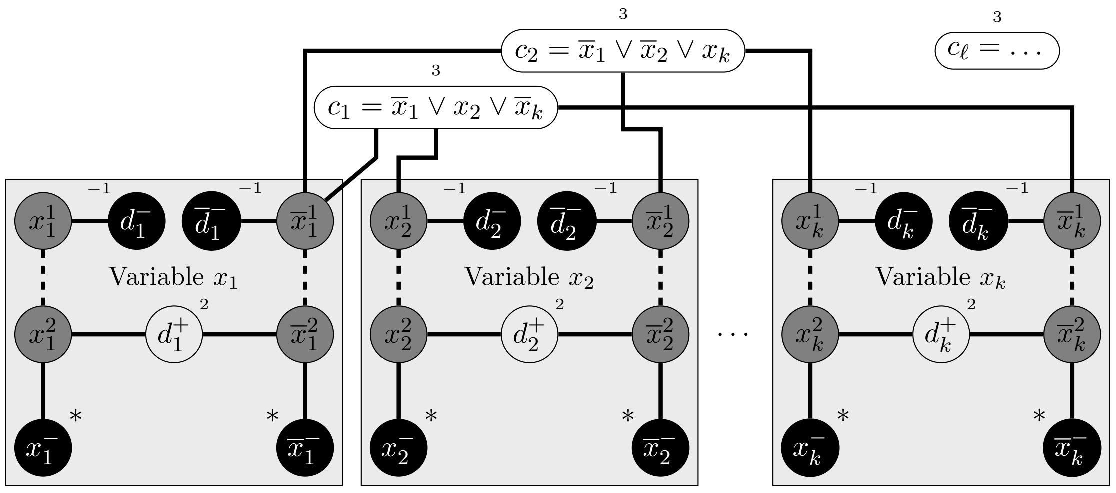


Figure 8.13: A maximum multi-commodity concurrent contraflow over time problem instance derived from a 3SAT instance. The dashed edges are important for encoding SAT assignments.

- By construction of the instance, there is a demand for this clause commodity in x_i^-, which can be reached from x_i^2 (or $\bar{x}_i^-$ and $\bar{x}_i^2$, respectively).

Therefore we can fulfill as much demand of a clause commodity as it has satisfied literals in our assignment, which is at least 1. Thus, we have a concurrent flow value of $\frac{1}{3}$ for these commodities. Now we need to consider the variable commodities. Since our assignment can only set x_i to either 1 or 0, one of the edges $\{x_i^1, x_i^2\}$ and $\{\overline{x}_i^1, \overline{x}_i^2\}$ has been oriented as (x_i^2, x_i^1) or $(\overline{x}_i^2, \overline{x}_i^1)$, respectively, in each variable block. This creates a path to send one flow unit of each variable commodity, giving us a concurrent flow value of $\frac{1}{2}$ for them, yielding a total concurrent flow value of $\frac{1}{3}$.

Positive concurrent flow value → YES-Instance. In order to have a positive concurrent flow value, at least one of the edges $\{x_i^1, x_i^2\}$ and $\{\overline{x}_i^1, \overline{x}_i^2\}$ needs to be oriented as (x_i^2, x_i^1) or $(\overline{x}_i^2, \overline{x}_i^1)$ in each variable block – otherwise there is no way to route any flow from the variable commodity. Thus, we can define an assignment by setting $x_i = 0$ if $\{x_i^1, x_i^2\}$ has been oriented as (x_i^2, x_i^1) and $x_i = 1$ otherwise. Notice that if $\{x_i^1, x_i^2\}$ is oriented as (x_i^2, x_i^1), there is no flow reaching x_i^- (the edges adjacent to d_i^+ cannot be used to reach x_i^-, or no flow of the variable commodity could be sent). But since we have a positive concurrent flow value, there is flow from every clause commodity reaching one of its sinks. Such flow has – by construction of the network – to travel through the block of one of the variables contained in the clause. More specifically, it has to traverse the (x_i^1, x_i^2) or $(\overline{x}_i^1, \overline{x}_i^2)$ edge, depending on whether the variable appears negated in the clause or not. Thus, this flow travels through an edge representing an fulfilled literal of its clause in the assignment derived from the edge orientation. Therefore, the instance has a satisfying assignment, making it a YES-instance.

Thus, NO-instances have a concurrent flow value of 0, whereas YES-instances have a concurrent flow value of 1/3, proving that value-approximations are not possible. Since our argumentation holds for arbitrarily large time horizons, it follows that time-approximation is not possible either. □

We can use a similar construction to show a non-approximability result for the quickest multi-commodity contraflow problem.

Theorem 8.14. *The quickest multi-commodity contraflow problem cannot be approximated by time with a factor better than $\Theta(\sqrt{U})$ with U being the largest capacity, unless* $\mathsf{P} = \mathsf{NP}$. *This holds even in the case of zero transit times.*

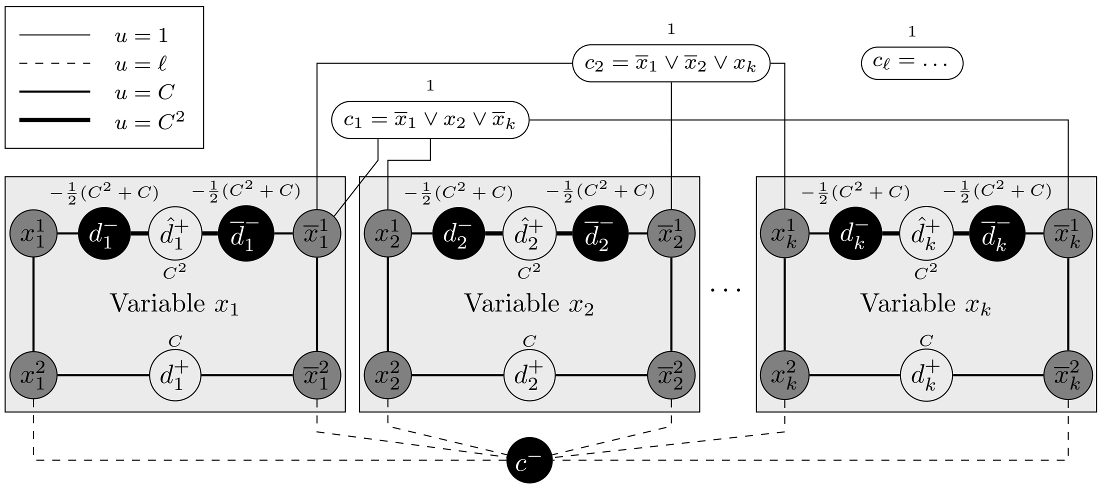
Proof. Consider an instance of 3SAT, given by a set of ℓ clauses $C = \{c_1, \dots, c_\ell\}$ on k variables $x_1, \dots, x_k$. We create a corresponding instance of the quickest multi-commodity contraflow problem as follows:

- We create a super-sink c^- and for each clause c_i we create a node c_i,
- for each variable x_i, we create nodes x_i^1, x_i^2, $\overline{x}_i^1$, $\overline{x}_i^2$, d_i^-, $\overline{d}_i^-$, d_i^+ and $\hat{d}_i^+$.
- For each clause $c_i = x_{i_1} \vee x_{i_2} \vee \overline{x}_{i_3}$ we create edges $\{c_i, x_{i_1}^1\}$, $\{c_i, x_{i_2}^1\}$ and $\{c_i, \overline{x}_{i_3}^1\}$,
- for each variable x_i, we create edges $\{d_i^+, x_i^2\}$, $\{d_i^+, \overline{x}_i^2\}$, $\{x_i^1, d_i^-\}$, $\left\{\overline{x}_i^1, \overline{d}_i^-\right\}$, $\{x_i^2, c^-\}$, $\{\overline{x}_i^2, c^-\}$, $\{x_i^1, x_i^2\}$, $\{\overline{x}_i^1, \overline{x}_i^2\}$, $\left\{\hat{d}_i^+, d_i^-\right\}$ and $\left\{\hat{d}_i^+, \overline{d}_i^-\right\}$.
- Capacities are set to C^2 for edges leaving $\hat{d}_i^+$, to C for the other edges completely inside a variable block, to 1 for edges incident to the source of a clause commodity and ℓ for all other edges.
- For each clause c_i, there is a commodity with supply of 1 at node c_i and a demand of -1 at c^-. For each variable x_i, there is a commodity with a supply of C at d_i^+, a supply of C^2 at $\hat{d}_i^+$, and a demand of $-\frac{1}{2}(C^2 + C)$ at d_i^- and $\overline{d}_i^-$.

Notice that the resulting network – an example of which is depicted in Figure 8.14 – has $\ell + 8k + 1$ nodes and $3\ell + 10k$ edges, which is polynomial in the size of the 3SAT instance.

YES-Instance → 1 time unit required. If the 3SAT instance is a YES-instance, then there is a variable assignment $x_i \in \{0, 1\}$, $i = 1, \dots, k$ fulfilling all clauses. We use this assignment to define an orientation of the edges in our network. If a variable x_i is assigned a value of 0, we orient the edge $\{x_i^1, x_i^2\}$ as (x_i^2, x_i^1) and $\{\overline{x}_i^1, \overline{x}_i^2\}$ as $(\overline{x}_i^1, \overline{x}_i^2)$; if x_i is assigned the value 1, we orient edge $\{\overline{x}_i^1, \overline{x}_i^2\}$ as $(\overline{x}_i^2, \overline{x}_i^1)$ and $\{x_i^1, x_i^2\}$ as (x_i^1, x_i^2). All other edges are oriented away from sources and towards sinks. Notice that:

- Each clause commodity of a clause $c_i = x_{i_1} \vee x_{i_2} \vee \overline{x}_{i_3}$ can send flow to the nodes $x_{i_1}^1, x_{i_2}^1, \overline{x}_{i_3}^1$.



$u = 1$
$u = \ell$
$u = C$
$u = C^2$
$c_2 = \overline{x}_1 \vee \overline{x}_2 \vee x_k$
$c_1 = \overline{x}_1 \vee x_2 \vee \overline{x}_k$
$c_\ell = \dots$
1
$-\frac{1}{2}(C^2 + C)$
x_1^1 d_1^- $\hat{d}_1^+$ $\overline{d}_1^-$ $\overline{x}_1^1$
C^2
Variable x_1
C
x_1^2 d_1^+ $\overline{x}_1^2$
Variable x_2
Variable x_k
c^-

Figure 8.14: An instance of the quickest multi-commodity concurrent flow problem derived from a 3SAT instance. Supplies and demands are zero unless specified otherwise, capacities are encoded by edge type.

- Each clause is satisfied by our assignment, so there is a literal in each clause that is true.
- For this literal x_i, there is an edge directed from x_i^1 to x_i^2 (or $\overline{x}_i^1$ to $\overline{x}_i^2$, respectively).
- By construction of the instance, there is a demand for this clause commodity in c^-, which can be reached from x_i^2 / $\overline{x}_i^2$.

Therefore we can fulfill the demand of a clause commodity if it has satisfied literals in our assignment, which it does. Now we need to consider the variable commodities. Since our assignment can only set x_i to either 1 or 0, one of the edges $\{x_i^1, x_i^2\}$ and $\{\overline{x}_i^1, \overline{x}_i^2\}$ has been oriented as (x_i^2, x_i^1) or $(\overline{x}_i^2, \overline{x}_i^1)$, respectively, in each variable block. This creates a path to send C flow units from d_i^+ to one of its sinks, and the remaining demands can be covered by supply from $\hat{d}_i^+$. Since the transit times are zero, all of this can be done in 1 time unit.

NO-instance → $\Theta(C)$ time units required. Any orientation that orients $\{x_i^1, x_i^2\}$ as (x_i^1, x_i^2) and $\{\overline{x}_i^1, \overline{x}_i^2\}$ as $(\overline{x}_i^2, \overline{x}_i^1)$ or $\{x_i^1, x_i^2\}$ as (x_i^2, x_i^1) and $\{\overline{x}_i^1, \overline{x}_i^2\}$ as $(\overline{x}_i^1, \overline{x}_i^2)$ can be interpreted as a variable assignment (see above). However, in a NO-instance, there is no variable assignment that satisfies all clauses. Thus, if all clause commodities could be fulfilled in such an orientation, there would have to be a clause c_i with a literal x_j, such that the edge $\left\{\overline{x}_j^1, \overline{x}_j^2\right\}$ is used to get to the clause sink. If we do this, we need to switch either the direction of one of the outgoing edges of $\hat{d}_j^+$ in order to get to $\left\{\overline{x}_j^1, \overline{x}_j^2\right\}$ from the entry point of the clause commodity into the variable block. However, this causes $1/2C^2 - 1/2C$ units of flow from the variable commodity to be send through edges with capacities of at most C, which requires at least $\Theta(C)$ time units.

If we orient both $\{x_i^1, x_i^2\}$ as (x_i^2, x_i^1) and $\{\overline{x}_i^1, \overline{x}_i^2\}$ as $(\overline{x}_i^2, \overline{x}_i^1)$, we cannot use the corresponding variable block to route flow from clause commodities, which does not help us at all.

The only option left would be to orient both $\{x_i^1, x_i^2\}$ as (x_i^1, x_i^2) and $\{\overline{x}_i^1, \overline{x}_i^2\}$ as $(\overline{x}_i^1, \overline{x}_i^2)$. However, this means that we have to route the C units of supply from d_i^+ over c^- and back to the variable block through a clause source, which requires at least C time units because of the capacities.

Thus, we have a gap of C between YES- and NO-instances, which proves the claim. $\square$

Bibliography

[1] A. Agapi, S. Soudan, M. Pasin, P. V.-B. Primet, and T. Kielmann. Optimizing deadline-driven bulk data transfers in overlay networks. In *Proceedings of 18th Internatonal Conference on Computer Communications and Networks (ICCCN '09)*, pages 1–8, 2009.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[3] G. Andreatta, L. De Giovanni, and G. Salamaso. Fleet quickest routing on grids: A polynomial algorithm. *International Journal of Pure and Applied Mathematics*, 62:419–432, 2010.

[4] J. E. Aronson. A survey of dynamic network flows. *Annals of Operations Research*, 20:1–66, 1989.

[5] J. E. Aronson and B. D. Chen. A forward network simplex algorithm for solving multiperiod network flow problems. *Naval Research Logistics Quarterly*, 33:445–467, 1986.

[6] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, 2009.

[7] A. Arulselvan, M. Groß, and M. Skutella. The price of orientation: On the effects of directing a network on flows over time. Unpublished.

[8] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger. Fast routing in very large public transportation networks using transfer patterns. In *Proceedings of the 18th European Symposium on Algorithms (ESA '10)*, pages 290–301. Springer, 2010.

[9] H. Bast and S. Storandt. Frequency data compression for public transportation network algorithms. In *Proceedings of the 6th Symposium on Combinatorial Search (SoCS '13)*, 2013.

[10] N. Baumann and E. Köhler. Approximating earliest arrival flows with flow-dependent transit times. *Discrete Applied Mathematics*, 155:161–171, 2007.

[11] N. Baumann and M. Skutella. Earliest arrival flows with multiple sources. *Mathematics of Operations Research*, 34:499–512, 2009.

[12] R. E. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[13] G. N. Berlin. The use of directed routes for assessing escape potential. *Fire Technology*, 14:126–135, 1978.

[14] K. Beygang, S. O. Krumke, and C. Zeck. Generalized max flow in series-parallel graphs. Report in Wirtschaftsmathematik 125, TU Kaiserslautern, 2010.

[15] D. R. Bish, E. P. Chamberlayne, and H. A. Rakha. Optimizing network flows with congestion-based flow reductions. *Networks and Spatial Economics*, pages 1–24, 2012.

[16] M. Braun and S. Winter. Ad-hoc solution of the multicommodity-flow-over-time problem. *IEEE Transactions on Intelligent Transportation Systems*, 10:658–667, 2009.

[17] L. E. J. Brouwer. Über Abbildung von Mannigfaltigkeiten. *Mathematische Annalen*, 71:97–115, 1911.

[18] R. E. Burkard, K. Dlaska, and B. Klinz. The quickest flow problem. *Zeitschrift für Operations Research – Methods and Models of Operations Research*, 37:31–58, 1993.

[19] R. E. Burkard, K. Feldbacher, B. Klinz, and G. J. Woeginger. Minimum-cost strong network orientation problems: Classification, complexity, and algorithms. *Networks*, 33:57–70, 1999.

[20] L. G. Chalmet, R. L. Francis, and P. B. Saunders. Network models for building evacuation. *Management Science*, 28:86–105, 1982.

[21] B. Cho and I. Gupta. New algorithms for planning bulk transfer via internet and shipping networks. In *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS '10)*, pages 305–314, 2010.

[22] W. Choi, H. W. Hamacher, and S. Tufekci. Modeling of building evacuation problems by network flows with side constraints. *European Journal of Operational Research*, 35:98–110, 1988.

[23] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.

[24] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[25] Y. Disser and M. Skutella. In defense of the simplex algorithm's worst-case behavior. *ArXiv e-prints*, November 2013.

[26] D. Dressler, G. Flötteröd, G. Lämmel, K. Nagel, and M. Skutella. Optimal evacuation solutions for large-scale scenarios. In B. Hu, K. Morasch, S. Pickl, and M. Siegle, editors, *Operations Research Proceedings 2010*, Operations Research Proceedings, pages 239–244. Springer Berlin Heidelberg, 2011.

[27] D. Dressler, M. Groß, J.-P. Kappmeier, T. Kelter, J. Kulbatzki, D. Plümpe, G. Schlechter, M. Schmidt, M. Skutella, and S. Temme. On the use of network flow techniques for assigning evacuees to exits. *Procedia Engineering*, 3:205–215, 2010.

[28] D. Eppstein, Z. Galil, and G. F. Italiano. Dynamic graph algorithms. In M. J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 8. CRC Press, 1999.

[29] F. Fischer and C. Helmberg. Dynamic graph generation for the shortest path problem in time expanded networks. *Mathematical Programming*, pages 1–41, 2012.

[30] L. Fleischer and M. Skutella. The quickest multicommodity flow problem. In W. J. Cook and A. S. Schulz, editors, *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO '02)*, volume 2337 of *Lecture Notes in Computer Science*, pages 36–53. Springer, 2002.

[31] L. Fleischer and M. Skutella. Minimum cost flows over time without intermediate storage. In *Proceedings of the 14th ACM–SIAM Symposium on Discrete Algorithms (SODA '03)*, pages 66–75, Baltimore, 2003.

[32] L. Fleischer and M. Skutella. Quickest flows over time. *SIAM Journal on Computing*, 36:1600–1630, 2007.

[33] L. Fleischer and É. Tardos. Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters*, 23:71–80, 1998.

[34] L. K. Fleischer. Faster algorithms for the quickest transshipment problem. *SIAM Journal on Optimization*, 12:18–35, 2001.

[35] L. K. Fleischer and K. D. Wayne. Fast and simple approximation schemes for generalized flow. *Mathematical Programming*, 91:215–238, 2002.

[36] M. Fonoberova. Algorithms for finding optimal flows in dynamic networks. In S. Rebennack, P. M. Pardalos, M. V. F. Pereira, and N. A. Iliadis, editors, *Handbook of Power Systems II*, Energy Systems, pages 31–54. Springer, 2010.

[37] L. R. Ford. Network flow theory. Paper P-923, The Rand Corporation, 1956.

[38] L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.

[39] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[40] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.

[41] D. Gale. Transient flows in networks. *The Michigan Mathematical Journal*, 6:59–63, 1959.

[42] T. Gallai. Maximum-minimum Sätze über Graphen. *Acta Mathematica Academiae Scientiarum Hungaricae*, 9:395–434, 1958.

[43] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[44] R. J. Gaskins and J. M. A. Tanchoco. Flow path design for automated guided vehicle systems. *International Journal of Production Research*, 25:667–676, 1987.

[45] M. X. Goemans. Minimum bounded degree spanning trees. In *Proceedings of the 47th IEEE Symposium on Foundations of Computer Science (FOCS '06)*, pages 273–282, 2006.

[46] A. V. Goldberg, S. A. Plotkin, and É. Tardos. Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 16:351–379, 1991.

[47] D. Goldfarb and Z. Jin. A faster combinatorial algorithm for the generalized circulation problem. *Mathematics of Operations Research*, 21:529–539, 1996.

[48] D. Goldfarb, Z. Jin, and J. B. Orlin. Polynomial-time highest-gain augmenting path algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 22:793–802, 1997.

[49] M. Gondran and M. Minoux. *Graphs and Algorithms.* Wiley, New York, 1984.

[50] M. Groß, J.-P. W. Kappmeier, D. R. Schmidt, and M. Schmidt. Approximating earliest arrival flows in arbitrary networks. In L. Epstein and P. Ferragina, editors, *Algorithms – ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 551–562. Springer Berlin Heidelberg, 2012.

[51] M. Groß and M. Skutella. Generalized maximum flows over time. In P. Persiano and R. Solis-Oba, editors, *Proceedings of the 9th Workshop on Approximation and Online Algorithms (WAOA '11)*, volume 7164 of *Lecture Notes in Computer Science*, pages 247–260. Springer, 2012.

[52] M. Groß and M. Skutella. Maximum multicommodity flows over time without intermediate storage. In L. Epstein and P. Ferragina, editors, *Algorithms – ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 539–550. Springer Berlin Heidelberg, 2012.

[53] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, second corrected edition edition, 1993.

[54] B. Hajek and R. G. Ogier. Optimal dynamic routing in communication networks with continuous traffic. *Networks*, 14:457–487, 1984.

[55] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theoretical Computer Science*, 379:387–404, 2007.

[56] H. W. Hamacher and S. A. Tjandra. Mathematical modelling of evacuation problems – a state of the art. In M. Schreckenberg and S. D. Sharma, editors, *Pedestrian and Evacuation Dynamics*, pages 227–266, Berlin, 2002. Springer.

[57] Y. Han, X. Guan, and L. Shi. Optimization based method for supply location selection and routing in large-scale emergency material delivery. *IEEE Transactions on Automation Science and Engineering*, 8:683–693, 2011.

[58] G. Y. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–309, 1980.

[59] T. Harks, F. G. König, J. Matuschke, A. Richter, and J. Schulz. An integrated approach to tactical logistics network optimization. Preprint 034, Technische Universität Berlin, Institut für Mathematik, 2012.

[60] T. Hasuike, H. Katagiri, H. Tsubaki, and H. Tsuda. Tour planning for sightseeing with time-dependent satisfactions of activities and traveling times. *American Journal of Operations Research*, 3:369–379, 2013.

[61] M. Hausknecht, T.-C. Au, P. Stone, D. Fajardo, and T. Waller. Dynamic lane reversal in traffic management. In *14th International IEEE Conference on Intelligent Transportation Systems (ITSC '11)*, pages 1929–1934, 2011.

[62] S. Hernández, S. Peeta, and G. Kalafatas. A less-than-truckload carrier collaboration planning problem under dynamic capacities. *Transportation Research Part E: Logistics and Transportation Review*, 47:933–946, 2011.

[63] M. D. Hirsch, C. H. Papadimitriou, and S. A. Vavasis. Exponential lower bounds for finding Brouwer fix points. *Journal of Complexity*, 5:379–416, 1989.

[64] D. S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Co., Boston, 1997.

[65] B. Hoppe. *Efficient Dynamic Network Flow Algorithms.* PhD thesis, Cornell University, 1995.

[66] B. Hoppe and É. Tardos. Polynomial time algorithms for some evacuation problems. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 433–441, 1994.

[67] B. Hoppe and É. Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25:36–62, 2000.

[68] L. V. Kantorovich. Mathematical methods of organizing and planning production. Technical report, Publication House of the Leningrad State University, 1939. Translated in Management Science, 6:366–422, 1960.

[69] S. Kim and S. Shekhar. Contraflow network reconfiguration for evaluation planning: A summary of results. In *Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems (GIS '05)*, pages 250–259, 2005.

[70] B. Klinz and G. J. Woeginger. Minimum cost dynamic flows: The series-parallel case. In *Proceedings of the 4th Conference on Integer Programming and Combinatoral Optimization (IPCO '95)*, pages 329–343, 1995.

[71] B. Klinz and G. J. Woeginger. Minimum cost dynamic flows: The series parallel case. *Networks*, 43:153–162, 2004.

[72] E. Köhler, K. Langkau, and M. Skutella. Time-expanded graphs for flow-dependent transit times. In *Proceedings of the 10th European Symposium on Algorithms (ESA '02)*, pages 599–611. Springer, 2002.

[73] E. Köhler and M. Strehler. Traffic signal optimization using cyclically expanded networks. In T. Erlebach and M. Lübbecke, editors, *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10)*, volume 14, pages 114–129, 2010.

[74] E. Köhler and M. Strehler. Combining static and dynamic models for traffic signal optimization inherent load-dependent travel times in a cyclically time-expanded network model. *Procedia – Social and Behavioral Sciences*, 54:1125–1134, 2012.

[75] B. Korte and J. Vygen. *Combinatorial Optimization – Theory and Algorithms.* Springer, 2012.

[76] M. Lahmar, T. Assavapokee, and S. A. Ardekani. A dynamic transportation planning support system for hurricane evacuation. In *Proceedings of the 9th IEEE Intelligent Transportation Systems Conference (ITSC '06)*, pages 612–617, 2006.

[77] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.

[78] E. Minieka. Maximal, lexicographic, and dynamic network flows. *Operations Research*, 21:517–527, 1973.

[79] E. F. Moore. The shortest path through a maze. In *Proceedings of an International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.

[80] C. St. J. A. Nash-Williams. Well-balanced orientations of finite graphs and unobtrusive odd-vertex-pairings. In *Proceedings of the 3rd Waterloo Conference on Combinatorics (WCC '69)*, pages 133–149. Academic Press, New York, 1969.

[81] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, New York, 1988.

[82] K. Onaga. Dynamic programming of optimum flows in lossy communication nets. *IEEE Transactions on Circuit Theory*, 13:282–287, 1966.

[83] K. Onaga. Optimal flows in general communication networks. *Journal of the Franklin Institute*, 283:308–327, 1967.

[84] J. B. Orlin. Minimum convex cost dynamic network flows. *Mathematics of Operations Research*, 9:190–207, 1984.

[85] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41:338–350, 1993.

[86] J. B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118:237–251, 2009.

[87] J. B. Orlin. Max flows in $O(nm)$ time or better. In *Proceedings of the 45th ACM Symposium on the Theory of Computing (STOC '13)*, pages 765–774, 2013.

[88] M. S. Osman, B. Ram, J. Bhadury, P. Stanfield, L. Davis, and F. Samanlioglu. Optimization model for distributed routing for disaster area logistics. In *Proceedings of the 5th IEEE International Conference on Service Operations, Logistics and Informatics (SOLI '09)*, pages 278–283, 2009.

[89] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48:498–532, 1994.

[90] B. Peis and A. Wiese. Universal packet routing with arbitrary bandwidths and transit times. In O. Günlük and G. J. Woeginger, editors, *Proceedings of the 14th Conference on Integer Programming and Combinatoral Optimization (IPCO '10)*, volume 6655 of *Lecture Notes in Computer Science*, pages 362–375. Springer, 2011.

[91] A. B. Philpott. Continuous-time flows in networks. *Mathematics of Operations Research*, 15:640–661, 1990.

[92] W. B. Powell, P. Jaillet, and A. Odoni. Stochastic and dynamic networks and routing. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 141–295. Elsevier, 1995.

[93] M. Raayatpanah, H. Ghasvari, and A. Ebrahimnejad. Generalized water supply management over time. *Middle-East Journal of Scientific Research*, 15:383–388, 2013.

[94] T. Radzik. Faster algorithms for the generalized network flow problem. *Mathematics of Operations Research*, 23:69–100, 1998.

[95] T. Radzik. Improving time bounds on maximum generalised flow computations by contracting the network. *Theoretical Computer Science*, 312:75–94, 2004.

[96] S. Rebennack, A. Arulselvan, L. Elefteriadou, and P. M. Pardalos. Complexity analysis for maximum flow problems with arc reversals. *Journal of Combinatorial Optimization*, 19:200–216, 2010.

[97] M. Restrepo and D. P. Williamson. A simple gap-canceling algorithm for the generalized maximum flow problem. *Mathematical Programming*, 118:47–74, 2009.

[98] H. E. Robbins. A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, 46:281–283, 1939.

[99] K. Roy and C. J. Tomlin. Solving the aircraft routing problem using network flow algorithms. In *Proceedings of the 2007 American Control Conference (ACC '07)*, pages 3330–3335, 2007.

[100] S. Ruzika, H. Sperber, and M. Steiner. Earliest arrival flows on series-parallel graphs. *Networks*, 57:169–173, 2011.

[101] M. Schmidt and M. Skutella. Earliest arrival flows in networks with multiple sinks. *Discrete Applied Mathematics*, 164:320 – 327, 2014.

[102] A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency.* Springer, 2003.

[103] M. Singh and L. C. Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC '07)*, pages 661–670, 2007.

[104] M. Skutella. An introduction to network flows over time. In *Research Trends in Combinatorial Optimization*, pages 451–482. Springer, 2009.

[105] S. Soudan and P. V.-B. Primet. Mixing malleable and rigid bandwidth requests for optimizing network provisioning. In *Proceedings of the 21st International Teletraffic Congress (ITC '09)*, pages 1–8, 2009.

[106] S. Stiller and A. Wiese. Increasing speed scheduling and flow scheduling. In O. Cheong, K.-Y. Chwa, and K. Park, editors, *Algorithms and Computation*, volume 6507 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2010.

[107] S. Su, Y. Wang, S. Jiang, K. Shuang, and P. Xu. Efficient algorithms for scheduling multiple bulk data transfers in inter-datacenter networks. *International Journal of Communication Systems*, 2013.

[108] É. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34:250–256, 1986.

[109] S. A. Tjandra. Earliest arrival flow with time dependent capacity approach to the evacuation problems. Report in Wirtschaftsmathematik (WIMA Report) 75, Technical University of Kaiserslautern, 2001.

[110] S. A. Tjandra. *Dynamic network optimization with application to the evacuation problem.* PhD thesis, Technical University of Kaiserslautern, 2003.

[111] K. Truemper. On max flows with gains and pure min-cost flows. *SIAM Journal on Applied Mathematics*, 32:450–456, 1977.

[112] H. Tuydes and A. Ziliaskopoulos. Network re-design to optimize evacuation contraflow. In *Proceedings of the 83rd Annual Meeting of the Transportation Research Board*, Washington, DC, 2004.

[113] H. Tuydes and A. Ziliaskopoulos. Tabu-based heuristic approach for optimization of network evacuation contraflow. *Transportation Research Record: Journal of the Transportation Research Board*, 1964:157–168, 2006.

[114] V. V. Vazirani. *Approximation Algorithms.* Springer, 2004.

[115] M. A. Venkataramanan and K. A. Wilson. A branch-and-bound algorithm for flow-path design of automated guided vehicle systems. *Naval Research Logistics*, 38:431–445, 1991.

[116] K. D. Wayne. *Generalized Maximum Flow Algorithms.* PhD thesis, Cornell University, 1999.

[117] K. D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. *Mathematics of Operations Research*, 27:445–459, 2002.

[118] W. L. Wilkinson. An algorithm for universal maximal dynamic flows in a network. *Operations Research*, 19:1602–1612, 1971.

[119] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011.

[120] B. Wolshon. One-way-out: Contraflow freeway operation for hurricane evacuation. *Natural Hazards Review*, 2:105–112, 2001.

[121] B. Wolshon, E. Urbina, and M. Levitan. National review of hurricane evacuation plans and policies. Technical report, LSU Hurricane Center, Louisiana State University, Baton Rouge, Louisiana, 2002.

[122] N. Zadeh. A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 5:255–266, 1973.

Symbol Index

$\#(a, S)$	number of occurrences of a in S
$\overleftarrow{A(G)}$	set of reverse arcs of $A(G)$
$\overleftarrow{a}$	reverse arc of a
$a \in S$	a contained in S
$A(G)$	arc set of G
B^i_+	sum of all supplies of commodity i
B^i_-	sum of all demands of commodity i
b^i_v	supply or demand at node v from commodity i
B_+	sum of all supplies
B_-	sum of all demands
$c(f)$	cost of f
$c(x)$	cost of x
c_a	cost of a
$\delta^+_G(v)$	set of outgoing arcs of v in G
$\delta^-_G(v)$	set of incoming arcs of v in G
$\delta_G(v)$	set of incident arcs of v in G
$E(G)$	edge set of G
$\lvert f\rvert$	flow value of f
$\lvert f\rvert_\theta$	flow value of f until time θ
$\lvert f\rvert_{\text{conc}}$	concurrent flow value of f
$f^i_a(\theta)$	flow rate of commodity i into arc a at time θ
f_v	flow rate function of the auxiliary edge of v

$(G, u, \tau, \dots)$	graph G with capacities u, transit times τ, etc.
γ_a	gain of a
$\overrightarrow{G}$	orientation of G
$\overline{G}$	$[0, T - \tau_{s \to t} - 1]$-copies of G
$\widetilde{G_f^T}$	subnetwork of G_f^T created by removing nodes and arcs not on s'-t'-paths
G^T/Δ	time-condensed network
G^T	time-expanded network of G with time horizon T
G_x	residual network of G and x
$\gamma_{v \to w}$	the maximum gain of a v-w-path
K	commodity set
k	number of commodities
m	number of arcs or edges
n	number of nodes
N^T/Δ	time-condensed network
p	short for p_G if G can be inferred from the context
P^δ	path with delays
p_f	arrival pattern of f
p_G	earliest arrival pattern of G
$\mathcal{S}$	set of all s'^i-t'^i-sequences for all $i \in K$
$S[\to (a, k)]$	subsequence of S from the beginning up to but not including the k-th occurrence of a in S
$S[\to i]$	subsequence of S from the beginning up to but not including the i-th element of S
S_+^i	source set of commodity i
S_-^i	sink set of commodity i
S_+	source set of all commodities
S_-	sink set of all commodities

τ_a	transit time of a
T	time horizon
$\tau_{v\to w}$	the length of a shortest v-w-path (with respect to transit times)
$[\theta,\theta']G$	$[\theta,\theta']$-copies of G
$[\theta,\theta']x$	the maximum gain of a v-w-path
θG	θ-copy of G
θx	θ-flow of x
u_a	capacity of a
u_x	residual capacities of a flow x
$v \in S$	v is incident to arc or edge in S
$V(G)$	node set of G
$\lvert x\rvert$	flow value of x
$\lvert x\rvert_{\text{conc}}$	concurrent flow value of x
$\overline{x}$	$[0, T - \tau_{s\to t} - 1]x$
x_a^i	flow value of commodity i on a

Index