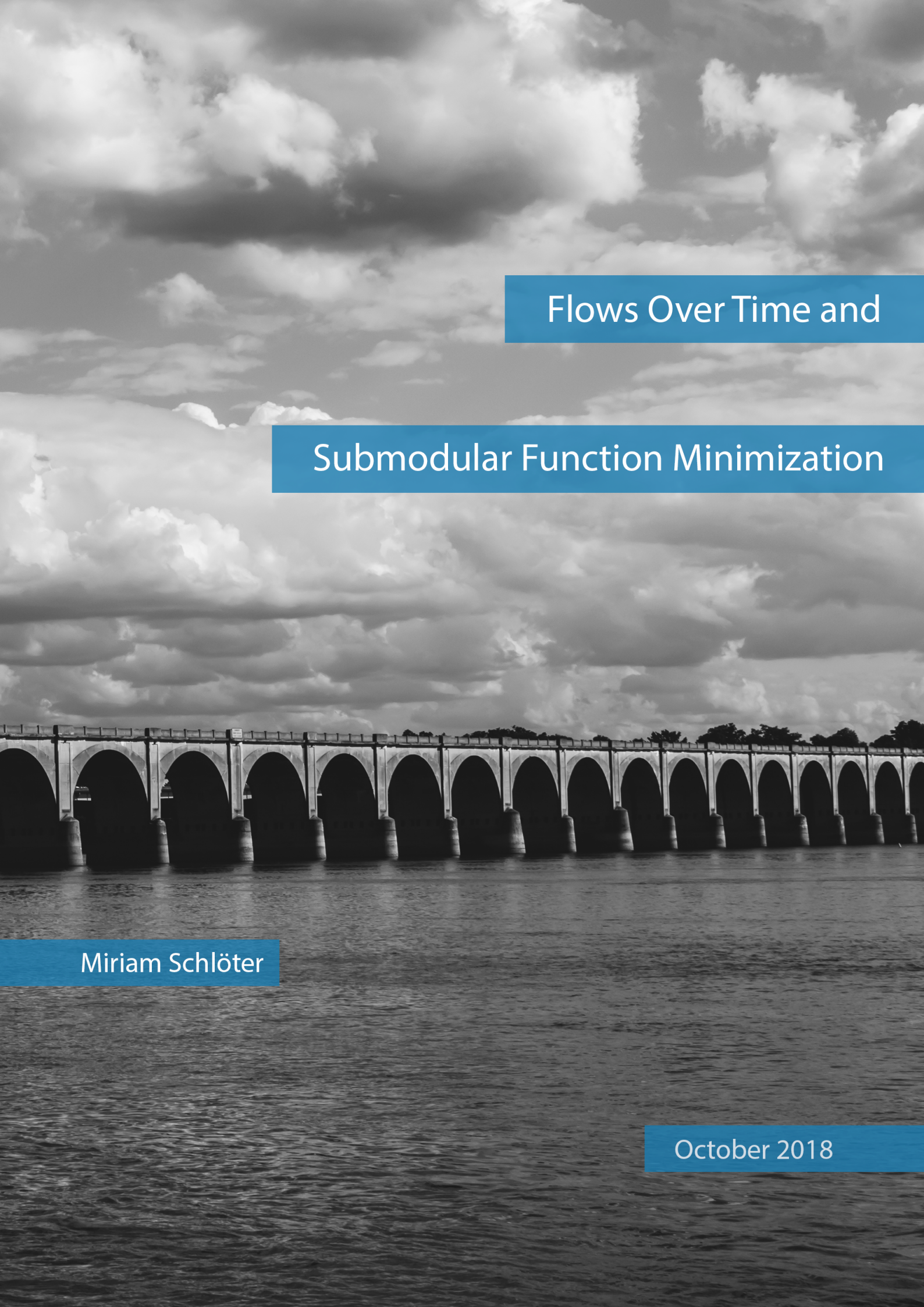


Flows Over Time and Submodular Function Minimization

Miriam Schlöter

October 2018

Flows Over Time and Submodular Function Minimization

vorgelegt von
M.Sc. Miriam Schlöter
geboren in Münster

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

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss

Vorsitzender:	Prof. Dr. John M. Sullivan
Gutachter:	Prof. Dr. Martin Skutella
	Prof. Dr. Britta Peis
	Prof. Dr. Thomas McCormick

Tag der wissenschaftlichen Aussprache: 16. April 2018

Berlin 2018

Miriam Schlöter
Flows Over Time and Submodular Function Minimization
PhD Thesis, October 14, 2018

Acknowledgements

This thesis is the result of the last 3 years in the COGA research group at Technische Universität Berlin. I in particular enjoyed the nice atmosphere in the group, which made it easy to continue the research even if some results stubbornly refused to be proven.
First of all, I want to thank my advisor Martin Skutella for introducing me to the field of network flows over time which are the main topic of my thesis and for being a rich source of new ideas. I am very grateful for the independence and freedom 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 DFG for the funding they provided me. I am also very grateful to Britta Peis and Tom McCormick for their interest and willingness to take the second assessment of this thesis.
A special thanks goes to all the people with whom I had the fun to work on the "Elevator Paper": Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Julie Meißner and Kevin Schewior. Working on the "Elevator" has been, in the retrospect, one of the highlights during my PhD.
I also want to thank my various office mates during my time at COGA for always being willing to let work rest for some time and have some non-research related chats.
Finally, I want to thank Leif Naundorf, at first for reading huge parts of this thesis and for listening to many conference practice talks, but mostly for all the other things he does every day to help and support me.

Contents

1 Introduction

Many parts of our daily routine that we expect to "just work" rely on certain optimization processes that most of the time take place in a *network*. Switching on the light or – more importantly – the coffee machine first thing in the morning relies on a functioning network of power lines, and water can only flow through our showers because of the underlying network of pipes.

The newspaper that some of you still might pick up from your doorsteps every morning has already traveled a long way through different networks when it is opened at the breakfast table: from the printing facility it has been brought to a local distributor that organizes the newspaper carriers, one of whom has delivered the paper to your house. Obviously, multiple optimization problems need to be solved in order for the paper to find the way into your hands: to maximize the number of distributed papers and to minimize the personnel expenses, the route of the carrier has probably been optimized in such a way that they can deliver as many papers as possible during their shift. In the more likely case that you do not receive a printed newspaper anymore, you might read it in a digital form on a tablet or smartphone or you maybe just want to check the news (or the latest tweets of certain politicians) in your favorite social network. Either way, multiple networks and optimization processes on them are intertwined with the content you want to pop up on your displays. There is the figurative network of telecommunications cables and the more abstract digital network of the Internet that make sure that it only takes a fraction of a second for a news article to load or for the latest tweet to appear in your news feed. In case of a social network there is, as the name suggests, also your personal network of friends or people you follow (your filter bubble), which is optimized towards the objective of only presenting you news (or advertisements) that are probably of interest to you.

Once you are on your way to work after reading the latest news using the medium of your choice, you might have to navigate on your bike or in your car through the morning traffic in a street network or – if you use public transport to get to work – through a network of local trains. No matter which form of transportation you use, arriving at work relaxed and on time probably relies on an optimization process in the specific network. When riding your bike through a city, the most important objective is often not to find the shortest route, but the way with the smallest amount of motorized traffic. In your car your navigation system might lead you along the quickest way to work – which might be the route with the smallest rate of bike traffic. In order to find the best public transport connection, many people rely on a smartphone application that calculates the best connection with respect to a suitably chosen objective (e.g., finding a connection without any changes, the fastest route, or the route which requires the shortest walking distances towards and from the train stations).

The list of network optimization problems we rely on every day goes on and on. From the order we make at Amazon to the invitation to a birthday party we get via Facebook: optimization problems in networks are omni-present in our daily life.

In many network optimization problems, also in most of the problems we mentioned above, the goal is to transport a commodity through the network in an efficient manner, e.g., people, objects, or more abstract commodities like the information you receive over a social network, or the movie you want to stream. Often a huge amount of the same commodity needs to be transported at once, like the large number of newspapers that are delivered each morning, the amount of data that travels through the communications network when you stream the latest episode of your favorite TV show, or the electricity that is needed to power the local hospital.

Such transportation problems gave rise to the development of **static network flows** more than 90 years ago [Tol30]. An abstract mathematical **network** (or **graph**) consist of **nodes** that are for example used to model houses in the power network, street crossings in road networks, computers in telecommunications networks, train stations in the networks of public transportation, or user accounts in social networks. The nodes of a network are connected by **edges** – if the connection don't have a direction – or **arcs** – if the connections have a direction. Arcs can for instance model roads, cables, train tracks, or the friendship or follower relations in social networks. Additionally, the arcs often also have bounded capacities that give an upper bound on the amount of a commodity that can be transported along the specific arc: clearly, only a bounded number of cars fits on a road, the amount

of information that you can send our receive via the Internet is bounded through the transmission rate at which signals can travel along the fiber optic (or copper) cables of the telecommunications network you are connected to (a fiber optic cable can reach – by using a technique of sending signals at 4 different wavelength – a transmission rate of 100 Gbit/s, while a copper cable only allows for 10Gbit/s), the number of posts or tweets that can be sent simultaneously through a social network is on the one hand also bounded by the transmission rate of the underlying telecommunications network, but on the other hand also by the properties of the used server infrastructure.
Usually, the commodities are supposed to be transported from special nodes (e.g., certain storage areas) – the **sources** – to certain other nodes – the **sinks** – where for example the customers are located. A **flow** in such a mathematical network can be viewed as a *transportation plan* that tells us how a specific transportation problem can be solved. Such a transportation plan has to fulfill two properties in order to be **feasible**: it is never allowed to exceed the capacity of an arc (*capacity constraint*), and for every node (except for the sources and sinks) the same amount of the transported good that has traveled into this node also has to travel out of this node (*flow conservation*). For example a delivery plan in which a newspaper is just dropped at a street crossing, or an information signal is lost at a switch that connects fiber optical cables would not be valid, as well as a public transportation plan which requires more than one train to drive on the same track at the same time. If **supplies** and **demands** are given, it is usually the goal that each source sends exactly its supply while each sink receives exactly its demand. For instance, the newspaper carrier should deliver each paper in order to satisfy the supply of his employer while also making sure that every person with a subscription of the paper receives one.
Flows in networks have been an active field of research during the past 90 years. From the beginning this research was strongly connected with or motivated by real life problems. In fact, the whole research area of network flows was basically born out of the military driven question of how to most efficiently cut off the Soviet railway network from "the west" [HR55]. In spite of its strong connection to practical applications, classical network flows often turn out to be too *static* to be useful when wanting to model certain scenarios. Which properties classical networks lack, can best be understood when imagining that our daily morning routine does not go as planned: Imagine, you open your door in the morning and your newspaper – due to some planning mistake – has not yet arrived preventing you from reading the news during your breakfast. Or, in case you are a more digital person, maybe the server of the newspaper's website is so slow that you cannot download the articles you want to read on time, or the social network of your choice is experiencing such a high amount of traffic that you cannot refresh your timeline or news feed. Once you are on your way to work without having read the paper or having checked your news online, you might be stuck in a traffic jam or miss your connecting train so that you do not arrive at work on time. This unsatisfactory day could end with your Amazon order not arriving during the promised time window. As you see, all of these transportation problems as well as many other transportation problems – besides their objective of transporting goods, people or information through a network – also need to meet a more or less fixed deadline. Classical static network flows cannot be used to solve transportation problems that need to meet a deadline because the underlying model does not take into account that it usually *takes time to transport a commodity along an arc*. There are networks in which the *delay* of a commodity resulting from the transit time along an arc seems to be negligible. For example, a signal can travel along a fiber optics cable at a velocity of around 69% of the speed of light [Fin] and thus it might only need fractions of a second to traverse the cable. For instance, a signal traveling along a straight cable spanned from Berlin to New York would need about 30.9 ms to traverse this distance. There are applications in which such a short travel time could be neglected, in some cases however tiny periods of time can make a huge difference: for example, in high frequency trading, where an advantage of a few milliseconds due to a faster fiber network can result in winning millions of dollars instead of losing this amount of money, financial firms go to surprising lengths to make the fiber network they use only a tiniest bit faster than the networks of their competitors [Tov].
In many other applications it is immediately clear that the travel time along arcs should be taken into account. One prominent example is the network optimization problem of evacuation planning: When devising a strategy of how to evacuate people from a burning building or out of a stadium after a bomb threat, it is vital for the life of the people to carefully incorporate the time needed to walk certain distances into the evacuation plan. Failing to do so might result in failing to rescue all the people before the (in this case very fixed) time horizon, i.e., before the building burns down or the bomb detonates. Thus, just assuming that people in a dangerous situation magically walk along

the planned evacuation roads without needing any time to do so (this is what a classical flow would imply), will most likely lead to the death of many people.
There is a recent example which shows that it can be most disturbing if time is not taken into account when distributing goods or information throughout a network. In January 2018 the Hawaii Emergency Management Agency mistakenly dispatched an alert warning of an incoming ballistic missile attack to cellphones across Hawaii. Not surprisingly, this alarm set off widespread panic throughout this state [Joc]. Within minutes of the alert people flocked to shelters, crowding highways in scenes of helplessness. Only minutes after the wrongly sent alert the agency and the governor began posting notices on Facebook and Twitter announcing the mistake. However, a flaw in the alert system delayed sending out the correction to cellphones, which finally happened 38 minutes after the emergency alert. The problem of sending the correction message to the inhabitants of Hawaii can clearly be modeled as a network problem in which the agency is the single source and the sinks are the people that need to be informed. The underlying network is a combination of different social networks and the cellphone network. When modeling the problem as a classical flow problem, the agency did a good job: the message was distributed to all people eventually. However, when taking time into account the agency failed miserably as one can be sure that 38 minutes is way past the deadline until when a correction message regarding a wrongfully sent ballistic attack alert should have been received by a major part of the population. The agency should have taken time into account.
A second aspect that cannot be modeled by classical networks flows is the **variance** of flow over time. For example, traffic is going to be much more intense during the rush hours in the morning and at night than in the middle of the day. In telecommunications networks there can be spikes in traffic, for example when the next season of a popular TV show is released. Power management obviously sees variation during the day and also during the season and social networks also should take the posting or tweeting habits of their users into account when optimizing their server infrastructure. Clearly, those two aspects – **delays** and **variance** – should be incorporated into the flow model in order to make it more applicable in practice. This was first done by Ford and Fulkerson 60 years ago [FF58] at the same time when also the first peak regarding research about classical (*static*) network flows occurred. Instead of fixing one flow value per arc, the so-called **dynamic flows** that they introduced allow for specifying a flow rate at which flow travels into an arc for each point in time. Thus, in this model the flow can vary over time. Furthermore, they also added transit times to the arcs of the considered network which model the time it takes for flow to completely traverse an arc in the network – flow that enters an arc at some point in time can only leave the arc after the specified transit time. Finally, they also introduced the term of the **time horizon** which acts as a deadline for the flow: all flow needs to have arrived at the sinks within the given time horizon. A **dynamic flow** in such a **dynamic network** network needs to fulfill all the properties of a static flow: the capacities of arcs need to be respected, and the overall amount of flow that has traveled into a specific node (except for the sources and sinks) immediately (or eventually, depending on the model) also has to leave this node. Additionally, when flowing through the network, the flow needs to respect the transit times of the arcs and it has additionally to fulfill the condition that after the time horizon *no flow remains in the dynamic network*. Consider again the wrongfully sent missile attack warning in Hawaii. If a time horizon had been incorporated into the dynamic network flow problem of distributing the correction message, a lot of distress for many people could have been prevented. In order to prevent misunderstanding, nowadays the term **flows over time** is used instead of dynamic flows, because the term *dynamic* is also often used in the context of graph problems where the input changes after having solved the problem and the objective is to adapt the solution efficiently to reflect the change in the input [Ber09]. Using flows over time is widely accepted in applications. Examples can be found in Aronson [AD86], Powell et al. [PJO95], Dressler et al. [Dre+11; Dre+10], Hamacher and Tjandra [HT02], Choi et al.[CHT88] and Berlin [Ber78].
Depending on the setting at hand, flow over time problems can have different objectives. The most straightforward flow over time problem is the **maximum flow over time problem** that has the objective to send as much flow as possible from a given set of sources to a given set of sinks within a specified time horizon. Closely related to this problem is the **quickest flow problem**. Given a dynamic network with a single source and a single sink with a fixed supply and demand, respectively, the goal of this problem is to find a flow over time that fulfills the given supply (and demand) as quickly as possible. One possible application of such a flow are special kinds of evacuation scenarios in which the people have to be rescued from a single endangered area at the source to a safe area

at the sink (see for example [CFS82; CHT88; HT02]). However, in many evacuation scenarios it is not sufficient to be restricted to dynamic networks with a single source and a single sink. This is where the **quickest transshipment problem** comes into play. It consists of a dynamic network with *multiple* sources and sinks, each of which has a bounded supply or demand, respectively. The goal is to find a flow over time that satisfies all supplies and demands *as fast as possible*. Clearly, quickest transshipments are at least suited to model one important aspect of a successful evacuation strategy: they rescue *all* the people from the endangered locations at the sinks to multiple safe areas at the sources as quickly as possible. As an example one can think of a cruise ship that is at risk of sinking. Of course, the goal is to bring all the passengers from their cabins into the safety of the life boats *before the ship sinks*. Also the distribution of the correction of the wrongly sent warning in Hawaii could have been modeled as a quickest transshipment problem. This way it would have been ensured that all people would have received the correction over some medium as quickly as possible. This would have spared many people a lot of fear and worry. The first and so far only known efficient algorithm for solving quickest transshipment problems has been devised by Hoppe and Tardos [HT00] in 1995.

Relying on quickest transshipments is, however, not sufficient for many evacuation scenarios. Consider again the case of the sinking cruise ship. It might be known that the ship will sink eventually, but it is usually not known when exactly this will happen. Only focusing on the minimal time horizon necessary to rescue *all* passengers will not lead to a satisfactory evacuation strategy if the ship sinks before all people could be rescued. In such a setting one should try to optimize multiple objectives, namely to maximize the number of people that have reached a lifeboat for every point in time simultaneously. This way – even if not all people can be rescued – it is made sure that the number of surviving passengers is at least maximized. This property is captured by **earliest arrival transshipments**, which are a special case of quickest transshipments. Sending the correction of the missile warning in Hawaii according to an earliest arrival transshipment would have been even more people-friendly. However, in case of earliest arrival transshipments the crux is that they *do not always exist*. For the special case of dynamic networks with a single source and a single sink **earliest arrival flows** have been introduced by Gale in 1959 [Gal59]. In this setting earliest arrival flows generalize maximum flows over time with respect to a time horizon by making sure that the flow that has arrived at the sink is maximal for each point in time until the time horizon (in this case no supply or demand is given). Gale also gave an existence proof for this special class of networks. Later, it was shown that also in dynamic networks with a single sink but multiple sources earliest arrival transshipment (with supplies and demands) do always exist [RT]. However, finding an efficient algorithm for computing earliest arrival transshipments turned out to be problematic.

In fact, it is not clear at all how to efficiently compute flows over time that are optimal with respect to any given objective. The problem is that flows over time are very flexible and hence there are many degrees of freedom: at every point in time we can choose a different flow value for each arc. We need (at least part of) this freedom in order to capture the needed variance that distinguishes flows over time from classical static network flows. However, it is also essential to bound this freedom in a smart way so that we can compute optimal solutions to flow over time problems efficiently. Ford and Fulkerson already introduced two solutions to this problem. The first and more straightforward approach is the use of **time expansion**. The main idea is that, instead of computing flows over time using the given dynamic network, we create one copy of the original network for each point in time and connect the different "layers" of this network by arcs according to the transit times of the arcs in the original network. This way the dynamic network is turned into a static network, which does not incorporate the temporal dimension anymore. A given flow over time problem can now be solved using algorithms from classical network flow theory in a static network. However, the size of the network gets blown up by the time horizon. This leads to a huge need of storage and to an at least pseudo-polynomial running time for every algorithm using the time-expanded network to solve flow over time problems.

The other approach developed by Ford and Fulkerson is to search for repetitive behavior in optimal solutions for flow over time problems that can be exploited. This reduces the degrees of freedom and allows us to find a solution more efficiently. In their algorithm for solving the maximum flow over time problem Ford and Fulkerson compute **temporally repeated flows** that send flow along paths from the source to the sink in the original dynamic network at a constant rate, and they show that the maximum flow over time problem can be solved by a flow over time with this structure. Hoppe and Tardos extend this notion to **generalized temporally repeated flows** in which flow

is also allowed to be sent along backwards arcs.
Coming back to algorithms for solving the earliest arrival transshipment problem with multiple sources, the first algorithm presented in [RT] used time expansion. Several years later, Baumann and Skutella [BS09] presented an algorithm that works without explicit time expansion but nevertheless requires pseudo-polynomial space due to the need to attach pseudo-polynomially many super-sinks to the network in the worst case in order to compute the earliest arrival transshipment. Moreover, the algorithm of Baumann and Skutella does *not* compute a (generalized) temporally repeated flow. Note, that it is unlikely that an algorithm with polynomial running time solving the earliest arrival transshipment problem in dynamic network with a single sink exist does exist as it was recently shown by Disser and Skutella [DS15] that it is $\mathcal{NP}$-hard to solve earliest arrival flow problems. However, it is a relevant task to come up with an algorithm for the earliest arrival transshipment problem in dynamic networks with only a single sink that only requires a polynomial amount of space. This is what part of this thesis focuses on.
In dynamic networks with multiple sinks it was first noted by Fleischer [FS07] that earliest arrival transshipments do not always exist. Due to this fact not much research has been put into the exact computation of earliest arrival transshipments in this class of networks. Except for the case of dynamic network with all zero transit times (see [SS14]) not much is known about such flows over time. The complexity of deciding whether an earliest arrival transshipment problem has a solution is unknown and also no algorithm (without using time expansion) is known to compute earliest arrival transshipments in multiple sink networks in case of existence. We can, however, answer a lot of the open questions regarding earliest arrival transshipment problems in dynamic networks with multiple sinks in this thesis, by settling the complexity – it is $\mathcal{NP}$-hard to determine whether a given earliest arrival transshipment problem has a solution – and presenting PSPACE algorithms for special cases.
In this thesis we focus on the three classical flow over time problems mentioned above: quickest transshipment problems (Chapter 4), earliest arrival transshipment problems in networks with a single sink (Chapter 5) and earliest arrival transshipment problems in networks with multiple sinks (Chapter 6). For each of these problems we develop new and more efficient algorithms that also compute structurally nice solutions.

Outline of this Thesis and Our Contributions

In this section we give a detailed overview of the structure of this thesis and its new contributions.

Chapter 2: Preliminaries. In this chapter we briefly introduce important notations and definitions that we use throughout this thesis. In particular, we give an introduction to parametric search, the foundations of submodular function minimization as well the basic definitions and results about static network flows that are relevant for our purposes. In particular, we review the successive shortest path algorithm and static lexicographically maximum flows. Finally, we introduce flows over time, formalize the concept of the time-expanded network and introduce the algorithm of Ford and Fulkerson [FF58; FF62] for the maximum flow over time problem.

Chapter 3: Flow Over Time Problems. In this chapter we give an introduction to the flow over time problems that we consider throughout 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. Regarding the quickest transshipment problem we concentrate on explaining the algorithm of Hoppe and Tardos for solving such problems. This in particular entails defining the lexicographically maximum (lex-max) flow over time problem and the strongly polynomial time algorithm of Hoppe and Tardos [HT00] for computing such flows over time.
Regarding earliest arrival flows in dynamic networks with a single source and a single sink we review Minieka's algorithm [Min73], which works in the time-expanded network, and the algorithm of Wilkinson [Wil71], which relies on the successive shortest path algorithm.
Considering earliest arrival transshipments in networks with a single sink we in particular introduce the algorithm of Baumann and Skutella [BS09] for computing the earliest arrival pattern and we review their results regarding the structure of the pattern. Finally, we shortly explain

how Baumann and Skutella solve earliest arrival transshipment problems by using a (pseudo-polynomial) expansion of the original network that relies on the structure of the earliest arrival pattern.

Chapter 4: Solving Quickest Transshipment Problems. This chapter focuses on our main contributions regarding the quickest transshipment problem. The algorithm by Hoppe and Tardos [HT00], which has been the best and only known strongly polynomial time algorithm for this problem for the past 20 years, works in two phases. First the minimal feasible time horizon T^* of a quickest transshipment problem is computed and afterwards the actual quickest transshipment is determined by reducing the problem to a lex-max flow over time problem. Due to this division of the algorithm into two phases this chapter is also divided into two parts.
At first we concentrate on determining the minimal feasible time horizon. Based on a feasibility criterion of Klinz [Kli] the so far most efficient approach to solve this problem relies on pairing submodular function minimization with the parametric search framework of Megiddo [Meg79], which results in a very expensive computation. For the special case of quickest transshipment problems in dynamic networks with only a single source *or* only a single sink we present a new approach to determine the minimal feasible time horizon that gets rid of using the parametric search framework and only needs to solve, in the worst case, number of terminals many (non-parametric) submodular function minimizations. The main fact that our algorithm exploits is a structural property of the parametric submodular functions that occur in the context of the quickest transshipment problem in this special class of networks: they are connected by a so-called **strong map**.
In the second part of this chapter we present two new algorithms for solving the quickest transshipment problem, both of which improve upon the worst case running time of the algorithm of Hoppe and Tardos. The faster of these algorithms needs essentially only one special parametric submodular function minimization, in contrast to Hoppe and Tardos' algorithm, whose bottleneck are two times number of terminals many calls to a black box for parametric submodular function minimization. The key idea of our approach is to not only use the actual result of the parametric submodular function minimization, but also to exploit the intermediate steps of this expensive computation and, in particular, the dual optimality certificate, which is a vector in the submodular function's base polytope. The vertices of this base polytope correspond to lex-max flows over time such that our solution to a quickest transshipment problem is simply a convex combination of these and therefore structurally somewhat simpler and certainly easier to analyze than the solution found by Hoppe and Tardos. More precisely, our quickest transshipment is a generalized temporally repeated flow. On the negative side, this implies that our quickest transshipment must be necessarily fractional, while the solution of Hoppe and Tardos is always integral.
The other algorithm we present also relies on the fact that the vertices of the base polytopes we consider correspond to lex-max flows over time while the supply/demand vector is – in case of a feasible transshipment problem – contained in this polytope. In order to find a suitable convex combination of lex-max flows over time solving our quickest transshipment problem, we present an implementation of Carathéodory's theorem that relies on a recent result about line search over the base polytope of a submodular function [GGJ17].
Some of the results from this section have been published in [SS17b].

Chapter 5: Earliest Arrival Transshipments in Networks with a Single Sink. In this chapter we focus on the earliest arrival transshipment problem in dynamic networks with a single sink – that is we only consider dynamic networks in which earliest arrival transshipments do always exist. In the first part of the chapter we present a faster algorithm for computing the parts of the **earliest arrival pattern** that we are interested in. The earliest arrival pattern is a function corresponding to a given earliest arrival transshipment problem that tells us for each point in time the value of an earliest arrival transshipment. This algorithm exploits the same facts as our algorithm for computing the minimal feasible time horizons for quickest transshipment problems in dynamic networks with a single source *or* a single sink.
Afterwards, we concentrate on developing a PSPACE algorithm for the earliest arrival transshipment problem in dynamic networks with only a single sink. Note that developing such an algorithm has been an open problem since Baumann and Skutella presented their algorithm for this flow over time problem more than a decade ago [BS09]. Overall, we present two PSPACE algorithms for solving the earliest arrival transshipment problem. The first algorithm produces a fractional solution but

does not require any parametric submodular function minimization, whereas the second approach produces an integral solution while relying on many parametric submodular function minimizations.
The first algorithm we derive is of a similar flavor as our new algorithm for the quickest transshipment problem, which, in particular, relies on a correspondence between the vertices of a submodular function's base polytope and lex-max flows over time. For the case of earliest arrival transshipments we do not only consider one base polytope but a product of several base polytopes of suitably chosen submodular functions. The precise definition of this polytope for a given earliest arrival transshipment problem strongly depends on the specific structure of its earliest arrival pattern. We will show that the vertices of this polytope correspond to generalizations of lex-max flows over time (called **generalized lex-max flows over time**) that we introduce in this chapter and for which we also derive a strongly polynomial time algorithm for computing them. Using the correspondence between the vertices of the suitably defined polytope and of the generalization of lex-max flows over time, we can deduce the structural result that earliest arrival transshipment problems in dynamic networks with *only a single sink* can always be solved by a convex combination of these flows. Furthermore, we will deduce that a suitable convex combination can essentially be computed while computing the required amount of information regarding the earliest arrival pattern. Despite the fact that the output size of the algorithm is necessarily pseudo-polynomial in the input size, it indeed runs in polynomial space and produces the output sequentially.
The second algorithm we present in this chapter is an adaptation of the algorithm of Hoppe and Tardos for the quickest transshipment problem that computes *integral* earliest arrival transshipments in polynomial space. In contrast to our other algorithm this algorithm needs number of sources many *parametric* submodular function minimizations and is thus a lot less efficient.
As a corollary of our results we are able to deduce the first FPTAS for earliest arrival transshipment problems *that does not rely on any form of time expansion.*

Chapter 6: Earliest Arrival Transshipments in Networks with Multiple Sinks. In the first part of this chapter we derive the earliest arrival pattern for earliest arrival transshipment problems in dynamic networks with only a single source but multiple sinks, and for the special case of **tight** problems in general dynamic networks. It turns out that the pattern construction is essentially symmetric to the patten construction of Baumann and Skutella [BS09].
In the remainder of this chapter we focus on devising a PSPACE algorithm that checks whether a given earliest arrival transshipment problem has a solution and that computes a suitable flow over time in case of existence. We distinguish between tight problems and non-tight problems. For tight problems we achieve such an algorithm for general networks. For non-tight problems our presented algorithm only works for earliest arrival transshipment problems in dynamic networks with multiple sources but only a single sink. It turns out that tight problem have a structural similarity to quickest transshipment problems. In order to achieve our results for the tight case, we at first concentrate on single source networks and in the end extend our results to general networks.
For single source networks we essentially define a suitable submodular function and show that a tight earliest arrival transshipment problem in such a network has a solution if and only if the demand vector is contained in the base polytope of this submodular function. Using our new PSPACE algorithm for evaluating the submodular function, we can thus check in PSPACE whether a given tight earliest arrival transshipment problem has a solution. To compute the solution we again show that the vertices of the considered base polytope correspond to a certain class of flows over time. Thus, we can deduce that – in case of existence – a tight earliest arrival transshipment problem can be solved by a convex combination of such flows. To obtain solutions of tight earliest arrival transshipment problems in PSPACE, we at first present a PSPACE algorithm for computing flows from the special class of flows over time that we defined, and we show that a suitable convex combination can essentially be computed during the submodular function minimization that is necessary to check whether the given problem has a solution.
Achieving a similar result for general networks requires to consider a more complicated class of flows over time, a more sophisticated polytope and also the feasibility criterion we achieve is more complicated. Overall, we show again that an earliest arrival transshipment problem can be achieved as a convex combination of special flows over time in case of existence.
At the end of the chapter we settle the complexity of the problem of deciding whether a given earliest arrival transshipment problem has a solution by showing that it is an $\mathcal{NP}$-hard problem.

Concluding Remarks. This thesis is supposed to be self-contained, however, we assume our reader to be familiar with the basic concepts of combinatorial optimization, complexity theory, approximation algorithms and linear programming.
For an introduction, see for example Schrijver [Sch03], Korte and Vygen [KV12] and Grötschel, Lovász and Schrijver [GLS88]. For classical network flow theory we specifically refer to the book by Ahuja, Magnanti, and Orlin [AMO93]. An introduction to complexity theory can be found in [WP05], and a thorough introduction to approximation algorithms is given in the book of Williamson and Shmoys [WS11].

2 Preliminaries

This chapter focuses on giving a formal introduction into the basic concepts that are required to understand the later chapters. In Section 2.1 we start by introducing graphs and our notations for them. Hereby, we solely focus on directed graphs as undirected graphs are not relevant throughout this thesis. In Section 2.2 we briefly introduce parametric search – a strongly polynomial time search framework developed by Megiddo [Meg79] in the 1970s, which is often used in the context of flow over time problems. Understanding the basics of submodular function minimization is essential throughout this thesis. This is why we give a short introduction into the theoretical foundation of submodular function minimization in Section 2.3. Before we introduce flows over time – the main topic of this thesis – in Section 2.5, we give a short survey of classical (static) network flows (Section 2.4) and present important static flow problems and techniques for solving them that are relevant throughout this thesis.

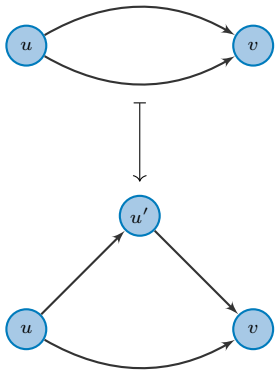
Contents

2.1 Graphs

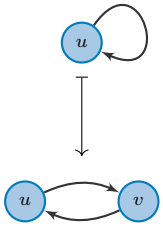
Usually, classical static flows (and also flows over time) are defined in directed graphs and this is why we only concentrate on these types of graphs in this section. In the literature directed graphs are sometimes called **networks** and vice versa. However, the distinction between these two terms is not always clear. Normally, a networks consists of a directed graph together with different attributes, like capacities, costs or transit times. In this thesis we will use the term *network* only when referring to a directed graph together with given attributes while always making sure that it is clear which attributes are used. The term *directed graph* will be used when we just refer to a directed graph without being interested in any potentially given attributes.

Directed Graphs. A **directed graph**, **digraph** or just **graph** is a pair $D = (V, A)$ where V is a finite set and A is a family of **ordered pairs** of elements from V. That is, A consists of elements in $V \times V$. The elements in V are called the **vertices** of D. Sometimes they are also denoted as **nodes**. The elements in A are called **arcs** or **directed edges**. Since each arc $a \in A$ is an element in $V \times V$ there are always nodes $u, v \in V$ with $a = (u, v)$. For an arc $a = (u, v) \in A$ with $u, v \in V$ we refer to u as the **start node** or **head** of a and to v as its **end node** or **tail** with the notation $\text{tail}(a) := u$

and head$(a) := v$.
If not stated otherwise, we usually denote the number of vertices of a directed graph $D = (V, A)$ by $n := |V|$ and the number of arcs by $m := |A|$. By definition, the arcs in A of a directed graph D form a family and hence it is possible that multiple copies of the same arc may occur. Hereby, two arcs with the same start and end node are called **parallel** arcs, i.e., $a = (u, v)$ and $a' = (u, v)$ for $u, v \in V$ are parallel arcs. Arcs of the form (v, v) for $v \in V$ are called **loops**. If not stated otherwise, we will assume throughout this thesis that the directed graphs we consider have neither parallel arcs nor loops. This restriction can usually be done without loss of generality because we can split parallel arcs and loops by introducing additional nodes (see Figure 2.1). Due to this



(a) Two parallel arcs and a construction to remove parallel arcs



(b) A loop and a construction showing how to remove it

Figure 2.1: How to remove parallel arcs and loops

assumption we can identify arcs by their start and end node. Given a directed graph $D = (V, A)$, we define $\delta_D^+(v)$ as the set of arcs of D leaving a node $v \in V$. We refer to this set as the set of **outgoing arcs**. Similarly, we define $\delta_D^-(v)$ to be the set of arcs of D entering a node $v \in V$. This set is denoted as the set of **incoming arcs** of the node v. The set $\delta_D(v)$ is defined to be the union of both sets:

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

This notations extends to subsets $U \subseteq V$ of nodes of D as follows,

$$\begin{aligned}
\delta_D^+(U) &:= \bigcup_{v \in U} \{a \in \delta_D^+(v) \mid \text{head}(a) \notin U\}\\
\delta_D^-(U) &:= \bigcup_{v \in U} \{a \in \delta_D^-(v) \mid \text{tail}(a) \notin U\}.
\end{aligned}$$

Thus, $\delta_D^+(U)$ is the set of arcs leaving U, while $\delta_D^-(U)$ is the set of arcs entering U. We call the arcs $a \in \delta_D(v)$ for $v \in V$ **incident** to v, and we refer to nodes $u, v \in V$ with $(u, v) \in A$ as **adjacent**. If the graph D can be inferred from the context, we just write $\delta^+(v)$, $\delta^-(v)$ and $\delta(v)$.
Given a directed graph $D = (V, A)$ and an arc $a \in A$ with $a = (u, v)$, the corresponding **backwards**

arc $\overleftarrow{a}$ is defined by $\overleftarrow{a} = (v, u)$. The **reverse directed graph** $\overleftarrow{D}$ corresponding to $D = (V, A)$ is defined by

$$\overleftarrow{D} := (V, \overleftarrow{A}) \text{ with } \overleftarrow{A} := \{\overleftarrow{a} \mid a \in A\}.$$

The **bidirected graph** $\overleftrightarrow{D}$ is the directed graph induced by a directed graph $D = (V, A)$ in which all arcs occur in both directions. That is $\overleftrightarrow{D} = (V, \overleftrightarrow{A})$ with $\overleftrightarrow{A} := A \cup \overleftarrow{A}$.

Cuts. Let $D = (V, A)$ be a directed graph. A subset $C \subseteq A$ is called a **cut** if $C = \delta^+(U)$ for some $U \subseteq V$. Thus, in particular the empty set $\varnothing$ is a cut. If $\varnothing \neq U \neq V$, then $\delta^+(U)$ is called a **nontrivial cut**. If $s \in U$ and $t \notin U$, then $\delta^+(U)$ is called an *s-t* **cut** or a cut **separating** s and t. If $S \subseteq U$ and $T \subseteq V \setminus U$, $\delta^+(U)$ is similarly called an *S-T* **cut** or a cut **separating** S and T. When the arcs have assigned costs c_a for all $a \in A$, the **cost** or **value** of a cut C is defined by $\sum_{a \in C} c_a$.

Paths and Cycles. Let $D = (V, A)$ be a directed graph. An *u-v* **sequence** $S = (a_1, a_2, \ldots, a_{|S|})$ in D is a sequence of arcs such that the arcs are connected, i.e., $\text{head}(a_i) = \text{tail}(a_{i+1})$ for $i \in \{1, 2, \ldots, |S| - 1\}$ with $u = \text{tail}(a_1)$ and $v = \text{head}(a_{|S|})$. The sequence S is called a **(directed)** *u-v* **path** if $a_i \neq a_j$ for all $i, j \in \{1, 2, \ldots, |S|\}$. If additionally each vertex on a path is visited exactly once, i.e. $\text{head}(a_i) \neq \text{head}(a_j)$ for all $i \neq j$ and $\text{head}(a_{|S|}) \neq \text{tail}(a_1)$, the path is called **simple**. A **cycle** is a path whose last arc ends at the first arc, i.e., $\text{head}(a_{|S|}) = \text{tail}(a_1)$.

2.2 Parametric Search

Often optimization problems occur depending on a linear parameter λ and the goal is to find the minimal or maximal value of λ such that the given optimization problem becomes **feasible**, i.e., the minimal or maximal value for λ such that a solution to the optimization problem exists. In the context of flows over time a common problem of this sort is the problem of determining the minimal feasible time horizon T such that the supplies and demands of a given dynamic network can be fulfilled within the time horizon T. If an upper bound λ^+ and a lower λ^- for the optimal value λ^* are known and λ^* is an integer, i.e., $\lambda^* \in [\lambda^-, \lambda^+]$, and if we have a decision algorithm to check feasibility for a fixed value of λ, then λ^* can be found using binary search by $\mathcal{O}(\log(\lambda^+ - \lambda^-))$ calls of the decision algorithm. However, this usually does not lead to an algorithm with strongly polynomial running time.
This is where the parametric search framework published by Megiddo in his seminal paper in 1979 comes into play [Meg79], see also [Meg83; VV02]. With this search framework the optimal λ^* for a parametric feasibility problem can be found in strongly polynomial time even if no upper and lower bounds on λ^* are known. That is, if the only thing known is that $\lambda^* \in (-\infty, \infty)$ (provided we have a strongly polynomial time decision algorithm that checks whether the optimization problem is feasible for a fixed value of λ).

Assume we are given some decision problem $\mathcal{D}(\lambda)$ that monotonically depends on a parameter λ. That is, if $\mathcal{D}(\lambda_0)$ is a YES-instance, then $\mathcal{D}(\lambda)$ is YES-instance for all $\lambda > \lambda_0$. Our goal is to compute the minimal λ^* such that $\mathcal{D}(\lambda^*)$ is a YES-instance. Additionally, suppose that we have an algorithm $\mathcal{A}_{\text{dec}}$ that solves the decision problem $\mathcal{D}(\lambda)$ and that can check whether $\lambda > \lambda^*$, $\lambda = \lambda^*$ or $\lambda < \lambda^*$ for any given λ. We also assume that the flow of control of the algorithm depends on comparisons each of which depends on the sign of a polynomial in λ.
The main idea of Megiddo's framework is to run $\mathcal{A}_{\text{dec}}$ generically on the unknown input λ^*. During the execution we maintain an interval I, initially defined to be $(-\infty, \infty)$, in which λ^* has to lie. Whenever a comparison depending on the parameter λ needs to be done, we need to know the sign of a polynomial p at λ^*. This sign can be determined without knowing the concrete value of λ^* as follows: to determine the position of λ^* among the roots of p we run the concrete version of $\mathcal{A}_{\text{dec}}$ on the roots of p. This either gives us two consecutive roots r_i and r_{i+1} such that $r_i < \lambda^* < r_{i+1}$ or we find out that λ^* is a root of p. In the latter case, we are done while in the first case we can find out the sign of $p(\lambda^*)$ by evaluating p at any $x \in (r_i, r_{i+1})$. This determines the outcome of a comparison and after updating I we proceed with the generic execution of the algorithm. During

the execution of the algorithm the interval I successively gets smaller and we either run $\mathcal{A}_{\text{dec}}$ to completion or we find λ^* prematurely. If $\mathcal{C}_\lambda(\mathcal{A}_{\text{dec}})$ is the number of comparisons of $\mathcal{A}_{\text{dec}}$ that depend on the given parameter λ, we thus have to run $\mathcal{A}_{\text{dec}}$ at most $\mathcal{O}(\mathcal{C}_\lambda(\mathcal{A}_{\text{dec}}))$ times during parametric search.

Theorem 2.1 (Parametric Search).

Let $\mathcal{D}(\lambda)$ be a decision problem that monotonously depends on the parameter λ. The goal is to find the minimal parameter λ^* such that $\mathcal{D}(\lambda^*)$ is a YES*-instance. Let $\mathcal{A}_{dec}$ be an algorithm to solve this decision problem for a given parameter λ and $\mathcal{C}_\lambda(\mathcal{A}_{dec})$ the number of comparisons of the algorithm that depend on λ. The optimal λ^* can be found in running time $\mathcal{O}(\mathcal{C}_\lambda(\mathcal{A}_{dec}) \cdot R)$ using parametric search. Here R is the running time of $\mathcal{A}_{dec}$.*

Given an algorithm $\mathcal{A}$ and a parameter λ, we will denote by $\mathcal{C}_\lambda(\mathcal{A})$ *the number of comparisons depending on λ during the execution of the algorithm $\mathcal{A}$.*

2.3 Submodular Functions

Submodular functions and their minimization play an important role in many problems from combinatorial optimization. Also in the context of the topic of this thesis – flows over time – different submodular functions occur, which have to be minimized. This is why we give a short introduction into the theory of submodular function minimization. For a thorough introduction into submodular function minimization see [McC05].

2.3.1 Basic Definitions

Given a finite ground set S, we say that a **set function** $g\colon 2^S \to \mathbb{R}$ is **submodular** if

$$g(A) + g(B) \geq g(A \cap B) + g(A \cup B) \text{ for all } A, B \subseteq S \tag{2.1}$$

or equivalently

$$g(A \cup \{e\}) - g(A) \geq g(B \cup \{e\}) - g(B) \text{ for all } A \subseteq B \subseteq S \text{ and } e \in S \setminus B. \tag{2.2}$$

By 2^S we denote the **powerset** of the set S. The set function g is **supermodular** if $-g$ is submodular and **modular** if it is both super- and submodular. Thus, a set function is supermodular if and only if it satisfies (2.1) (or (2.2)) with the reversed inequalities, and modular if and only if (2.1) is satisfied with equality.

A well known example of a submodular function is the **cut function** in a directed graph.

Example 2.2. *Let $D = (V, A)$ be a directed graph and $w\colon A \to \mathbb{R}$ a weight function on the arcs. Define $g\colon 2^V \to \mathbb{R}_{\geq 0}$ by $U \mapsto w(\delta^+(U))$. Then,*

$$g(U) + g(U') \geq g(U \cup U') + g(U \cap U') \text{ for all } U, U' \subseteq V,$$

and thus g is submodular.

2.3.2 Theoretical Foundations for Submodular Function Minimization

Let $g\colon 2^S \to \mathbb{R}$ be a submodular function. **Submodular function minimization (SFM)** is the problem of determining $X \subseteq S$ such that $g(X)$ is minimal over all subsets of S:

Submodular Function Minimization (SFM)

Instance: A submodular function $g\colon 2^S \to \mathbb{R}$ over a ground set S.

Task: $\min_{X \subseteq S} g(X)$ and the set $X^* \subseteq S$ at which the minimum is attained.

For some special cases of submodular functions (e.g., the cut function) strongly polynomial time algorithms for submodular function minimization are known. However, for general submodular functions it is not straightforward at all how to minimize them efficiently. The brute force approach of checking all 2^n values of a submodular function g over a ground set of size n is clearly not a practical idea.

Let $g\colon 2^S \to \mathbb{R}$ be a submodular function over a ground set S. In the following we always assume that $g(\varnothing) = 0$. We can ensure this without loss of generality by redefining $g(U)$ to be $g(U) - g(\varnothing)$ for all $U \subseteq S$. This change clearly does neither affect the submodularity of g nor its minimizer.
Each submodular function gives rise to a polyhedron, the **submodular polyhedron** defined as follows

$$\mathcal{P}(g) \coloneqq \{x \in \mathbb{R}^S \mid x(U) \leq g(U) \text{ for all } U \subseteq S\}. \tag{2.3}$$

In the context of submodular function minimization the face of $\mathcal{P}(g)$ defined by the equation $x(S) = g(S)$ turns out to useful. We define the **base polytope** $\mathcal{B}(g)$ of g by

$$\begin{aligned} \mathcal{B}(g) &\coloneqq \{x \in \mathcal{P}(g) \mid x(S) = g(U)\} \\ &= \{x \in \mathbb{R}^S \mid x(U) \leq g(U) \text{ for all } U \subseteq S \text{ and } x(S) = g(S)\}. \end{aligned} \tag{2.4}$$

By $x(U)$ for $x \in \mathbb{R}^S$ we denote the sum of the corresponding components of x, $x(U) = \sum_{u \in U} x(u)$ and by $\mathbf{0}$ the zero vector in the considered vector space. Given weights $w \in \mathbb{R}^S$, it is a natural question to wonder about maximizing the linear objective function $w^T x$ over the two polyhedra $\mathcal{P}(g)$ and $\mathcal{B}(g)$. It is a remarkable property of submodularity that a simple **Greedy Algorithm** (see Algorithm 1) can be used to solve this problem in strongly polynomial time. This fact was already shown by Edmonds [Edm70] in 1970.
Given a linear order $\prec$ on the elements of S, assume that $S = \{s_1, s_2, \ldots, s_n\}$ such that $s_1 \prec s_2 \prec \ldots \prec s_n$ and define $S_i \coloneqq \{s \in S \mid s \preceq s_i\} = \{s_1, s_2, \ldots, s_i\}$ for all $i \in \{0, 1, \ldots, n\}$

Algorithm 1: The Greedy Algorithm Greedy$(g, \prec)$

```
Input  : A submodular function g: 2^S → ℝ on the ground set S, a linear order ≺ on S
Output : A vector v^≺ ∈ ℝ^S
for i = 1, 2, ..., n do
    v_i ← g(S_i) − g(S_{i−1})
end
v^≺ ← (v_1, v_2, ..., v_n)
return v^≺
```

In order to maximize the linear objective $w^T x$ with $w \in \mathbb{R}^S$, note at first that we can assume $w \geq \mathbf{0}$ as otherwise the optimum value would be unbounded:
Assume we have $w_s < 0$ for some $s \in S$ and let $y \in \mathcal{P}(g)$. If $y' \leq y$, then – by definition of $\mathcal{P}(g)$ – it also holds that $y' \in \mathcal{P}(g)$. Thus, $\max\{w^T x \mid x \in \mathcal{P}(g)\}$ is unbounded, because we can let y'_e go to infinity, $y'_e \to -\infty$.
To use the Greedy Algorithm to maximize $w^T x$ over $\mathcal{P}(g)$ (or $\mathcal{B}(g)$) with $w \in \mathbb{R}^S$ and $w \geq \mathbf{0}$, we proceed as follows: index the elements in S as $s_1, s_2, \ldots, s_n$ such that $w_{s_1} \geq w_{s_2} \geq \ldots \geq w_{e_n}$, define a linear order $\prec_w$ on S by $s_1 \prec_w s_2 \prec_w \ldots \prec_w s_n$ and perform Greedy$(g, \prec_w)$ to obtain $v^{\prec_w}$ (see Algorithm 2). The running time of GreedyOpt(g, w) clearly is $\mathcal{O}(n \log n + n\mathrm{EO}(n))$, where $n = |S|$ and $\mathrm{EO}(n)$ is the running time of an **evaluation oracle** for the submodular function g. Note that

Algorithm 2: The Greedy Algorithm for Optimization GreedyOpt(g, w)

```
Input  : A submodular function g: 2^S → ℝ on the ground set S, a weight vector w ∈ ℝ^S
Output : A vector v^w ∈ ℝ^S maximizing w^T x over P(g) or Unbounded
if ∃s ∈ S with w_s < 0 then
    return Unbounded
end
index the elements in S as s_1, s_2, ..., s_n such that w_{s_1} ≥ w_{s_2} ≥ ... ≥ w_{s_n}
define a linear order ≺_w on S by s_1 ≺_w s_2 ≺_w ... ≺_w s_n
v^w ← Greedy(g, ≺_w)
return v^w
```

if $w \geq \mathbf{0}$, the vector v^w returned by GreedyOpt(g,w) lies in $\mathcal{B}(g)$, because $v^w(S) = \sum_{i=1}^{n} g(S_i) - g(S_{i-1}) = g(S_n) = g(S)$. It is due to Edmonds [Edm70] that GreedyOpt in fact optimizes $w^T x$ over $\mathcal{P}(g)$ for a given submodular function g.

Theorem 2.3 ([Edm70]).

Given a submodular function $g\colon 2^S \to \mathbb{R}^S$ on a finite ground set S, $w \in \mathbb{R}^S$ with $w \geq \mathbf{0}$, and $v^w :=$ GreedyOpt(g, w), it holds that

$$w^T v^w = \max\{w^T x \mid x \in \mathcal{P}(g)\} = \max\{w^T x \mid x \in \mathcal{B}(g)\}$$

and v^w is a vertex of $\mathcal{B}(g)$.

It is an immediate consequence of Theorem 2.3 that the vertices of $\mathcal{B}(g)$ correspond to linear orders on S – and thus $\mathcal{B}(g)$ can have as many has $n!$ vertices:
Given a linear order $\prec$ on S the corresponding vertex $v^{\prec}$ of $\mathcal{B}(g)$ can be computed using the Greedy Algorithm and on the other hand, for each vertex v of $\mathcal{B}(g)$ there is an corresponding linear order $\prec$ on S such that Greedy$(g, \prec) = v^{\prec} = v$. That we are able to compute vertices of $\mathcal{B}(g)$ using the Greedy Algorithm is an important fact used in many algorithms for submodular function minimization.
A second important building block for algorithms for submodular function minimization is the following theorem that gives a dual characterization of the minimum of a submodular function:

Theorem 2.4 ([Edm70]).

Let $g\colon 2^S \to \mathbb{R}$ be a submodular function on a finite ground set S. We have

$$\begin{aligned} \min_{U \subseteq S} g(U) &= \max\{z(V) \mid z \in \mathcal{P}(g), z \leq \mathbf{0}\} \\ &= \max\{x^-(S) \mid x \in \mathcal{B}(g)\}, \end{aligned} \tag{2.5}$$

with $x^-(s) = \min\{0, x(s)\}$.

2.3.3 Submodular Function Minimization in Strongly Polynomial Time

Since the 1970s, when Edmonds came up with the first results regarding submodular function minimization, this problem has been an object of research in the area of combinatorial optimization. However, no efficient algorithm was found for submodular function minimization until the ellipsoid method for solving linear programming problems arrived.

SFM and the Ellipsoid Method. In 1981 Grötschel, Lovász and Schrijver [GLS81] used the ellipsoid method to derive the equivalence of optimization and separation. This fact turned out to be immensely useful in the context of submodular function minimization. Using the Greedy Algorithm (Algorithm 2) we can optimize over the base polytope $\mathcal{B}(g)$ of a given submodular function g in strongly polynomial time. Thus, the equivalence of optimization and separation immediately implies

that the separation problem corresponding to $\mathcal{B}(g)$ can be solved in polynomial time using the ellipsoid method. Note that this is in particular nice as the description of $\mathcal{B}(g)$ as the intersection of hyperplanes (see (2.4)) contains exponentially many hyperplanes in the worst case.
An observation of Cunningham [Cun83] showing that the minimization of a submodular function g can be reduced to a separation problem corresponding to $\mathcal{B}(\tilde{g})$, where $\tilde{g}$ is another submodular function, finally implied that submodular function minimization can be solved in weakly polynomial time using the ellipsoid method. However, it remained an open question whether a *strongly* polynomial time algorithm for submodular function minimization exists. In 1988 Grötschel, Lovaász and Schrijver [GLS88] finally showed that the ellipsoid method can be adapted to submodular function minimization to achieve an algorithm with strongly polynomial running time $\tilde{\mathcal{O}}(n^5\mathrm{EO} + n^7)$ [McC05].

SFM with the Framework of Cunningham. Although the ellipsoid method can be used to solve submodular function minimization in strongly polynomial time, this result was not completely satisfactory since the ellipsoid method is not useful in practice and it does not give much insight into the structural properties of submodular function minimization problems. In 1985 Cunningham [Cun85] stated that "it is an outstanding open problem to find a combinatorial algorithm to minimize a general submodular function, which runs in polynomial time".
It was also Cunningham who laid the foundation for many combinatorial strongly polynomial time algorithms for submodular function minimization that were developed in the subsequent years. He used Theorem 2.4 to develop a framework for SFM and his application of it yielded a combinatorial pseudo-polynomial algorithm for submodular function minimization with running time $\mathcal{O}(Mn^6 \log nM \cdot \mathrm{EO})$ [BCT85; Cun84; Cun85], where M is the largest value of the submodular function.
The question whether a combinatorial (strongly) polynomial time algorithm for SFM existed, remained open until 1999, when nearly simultaneously two working papers appeared giving two different combinatorial strongly polynomial time algorithms for submodular function minimization. The algorithms were published by Schrijver [Sch00] ($\mathcal{O}(n^8\mathrm{EO} + n^9)$), and Iwata, Fleischer and Fujishige [IFF01] ($\mathcal{O}(n^7 \log n \cdot \mathrm{EO})$) and both were based on the framework of Cunningham. In the following years a sequence of combinatorial algorithms with strongly polynomial running time for SFM using Cunningham's framework were released. The current fastest combinatorial strongly polynomial time algorithm for SFM that *relies on the framework of Cunningham* is due to Orlin and has running time $\mathcal{O}(n^5 \cdot \mathrm{EO} + n^6)$ [Orl09].
Since the main ideas of Cunningham's framework are central throughout this thesis, we will shortly introduce them. For this purpose let $g\colon 2^S \to \mathbb{R}$ be a submodular function. The key element in Cunningham's framework is Theorem 2.4, which states that

$$\min_{X \subseteq S} g(X) = \max\{x^-(S) \mid x \in \mathcal{B}(g)\}.$$

Thus, instead of minimizing g, we can equivalently maximize the function $x^-(S)$ over all $x \in \mathcal{B}(g)$, i.e., we want to find the point x^* in $\mathcal{B}(g)$ such that the sum of all negative valued components is maximized. The main structure of all algorithms relying on the framework of Cunningham is as follows: Given a vector $x \in \mathcal{B}(g)$ inside the base polytope, the aim is to update the vector such that the value of the function $x^-(S)$ is increased *and* the updated vector x' still lies within the base polytope. To make sure that the vector remains inside the base polytope, the current vector $x \in \mathcal{B}(g)$ is maintained as a convex combination of vertices of $\mathcal{B}(g)$, i.e., we are given x as $x = \sum_{i \in I} \lambda_i v^i$ where the λ_i for all $i \in I$ are convex coefficients and the v_i for $i \in I$ are vertices of $\mathcal{B}(g)$. In each iteration the convex combination gets updated by some smart update procedure in which new vertices are added to the convex combination and others are removed. To avoid that the current convex combination gets too large a "Carathéodory" subroutine is necessary that gets a convex combination of vertices as the input and returns a minimal convex combination of vertices representing the same point in the polytope. A key factor here is that each vertex of $\mathcal{B}(g)$ is characterized by a total order $\prec$ on S and that, given this order, the corresponding vertex can be computed efficiently using the Greedy Algorithm 1 (provided we have an efficient evaluation oracle for the submodular function). Thus, in the first iteration of an algorithm for SFM just an arbitrary linear order $\prec$ on the ground set S is chosen and the corresponding vertex is computed using the Greedy Algorithm. In the subsequent iterations the vertices in the convex combination are usually updated by choosing new

more suitable linear orders on S.
One main feature of SFM algorithms relying on the framework of Cunningham is that during the computation of the minimum of a submodular function g, not only the minimum value is computed, but also a convex combination of vertices $\mathcal{B}(g)$ giving the vector $x^* = \operatorname{argmax}\{x^-(S) \mid x \in \mathcal{B}(g)\}$ (see Theorem 2.4).
The algorithm of Orlin is denoted by $\mathrm{SFM}_{\mathrm{Orlin}}$ throughout this thesis. It returns an inclusion-wise maximal subset $X^* \subseteq A$ minimizing a submodular function g, a vector $x^* \in \mathcal{B}(g)$ maximizing $x^-(S)$ over all $x \in \mathcal{B}(g)$ given as a minimal convex combination of vertices of $\mathcal{B}(g)$, and the minimal value $v_{\min}$ of the submodular function.

SFM without the Framework of Cunningham. Until recently, all combinatorial algorithms for submodular function minimization with strongly polynomial running time relied on the framework of Cunningham. However, the current fastest algorithm for submodular function minimization does *not* use this framework. This algorithm is due to Lee, Sidford and Wong [LSW15] and it improves upon the algorithm of Orlin by a factor of n^2 ($\mathcal{O}(n^3 \log n \cdot \mathrm{EO} + n^4 \log^{\mathcal{O}(1)} n)$). The authors state that the fact that they do not maintain a point inside the base polytope as a convex combination of vertices during the computation is what leads to the running time improvement. However, what turns out to be a downside in context of our flow over time computations is that we do not get the "optimal" point inside the base polytope as a convex combination of vertices by minimizing the submodular function.
Throughout this thesis we denote the algorithm of Lee, Sidford and Wong by $\mathrm{SFM}_{\mathrm{Lee}}$.

2.3.4 Parametric Submodular Function Minimization

Often a submodular function g occurring in the context of some optimization problem is also linearly dependent on an additional parameter λ. Thus, g^λ is a **parametric** or **parametrized** submodular function. In such a setting the main objective is often to determine the maximal or minimal value of the parameter λ such that the minimum value of the submodular function is below or above a certain threshold.
We say that a submodular function $g^\lambda \colon 2^S \to \mathbb{R}$ **monotonically** depends on the parameter λ if $g^\lambda(A)$ monotonously grows with λ for each $A \subseteq S$. By **Parametric Submodular Function Minimization** we denote the problem of determining the minimal value of $\lambda \geq 0$ such that $g^\lambda(A)$ lies *above* a certain threshold for all $A \subseteq S$. If for all $A \subseteq S$ the function $\lambda \mapsto g^\lambda(A)$ decreases monotonically with increasing λ we can similarly look at the problem of determining the maximal value $\lambda \geq 0$ such that $g^\lambda(A)$ lies *above* a certain threshold.

> PARAMETRIC SUBMODULAR FUNCTION MINIMIZATION (g^λ, v)
>
> *Instance:* A submodular function $g^\lambda \colon S \to \mathbb{R}$ that monotonously depends on a parameter $\lambda \in \mathbb{R}$, and a vector $v \in \mathbb{R}^S$
>
> *Task:* Determine the minimal parameter $\lambda^* \geq 0$ such that
>
> $$g^{\lambda^*}(A) - v(A) \geq 0 \text{ for all } A \subseteq S.$$
>
> and a maximal or minimal minimizing subset of $g^{\lambda^*} - v$.

The straightforward way to solve a parametric submodular function minimization problem efficiently is to couple a strongly polynomial time algorithm for submodular function minimization with the parametric search framework of Megiddo. We can do this, as by assumption the parametric submodular function g^λ monotonously depends on the parameter.
The fastest strongly polynomial running time that can be achieved this way is $\mathcal{O}(\mathcal{C}_\lambda(\mathrm{SFM}_{\mathrm{Lee}}) \cdot (k^3 \log k \cdot \mathrm{EO} + k^4 \log^{\mathcal{O}(1)} k))$, where k is the size of the ground set S of the submodular function, EO is the running time of an evaluation oracle for the given submodular function for a fixed parameter and $\mathcal{C}_\lambda(\mathrm{SFM}_{\mathrm{Lee}})$ is the number of comparisons that are done in the submodular function minimization algorithm of Lee, Sidford and Wong [LSW15] depending on λ. Clearly, the number of comparisons

depends on the implementation of the specific algorithm, but in the worst case this number is in the same order as the running time of the algorithm. Hence, using the parametric search framework together with the algorithm of Lee et al. could in the worst case square the running time of this algorithm. This is why it often makes sense to try to develop faster algorithms for parametric submodular function minimization when considering a specific problem.

The Strong Map Property. Especially interesting for us are submodular functions that are connected by a **strong map**.

Definition 2.5 (The Strong Map Property).
Let $g_1, g_2 \colon 2^S \to \mathbb{R}$ be two submodular functions defined over the same finite ground set S. We write $g_1 \leftarrow g_2$ or $g_2 \to g_1$ if $A \subseteq B \subseteq S$ implies

$$g_1(B) - g_1(A) \leq g_2(B) - g_2(A).$$

*The relation is called a **strong map**. Submodular functions $g_1, g_2, \ldots, g_k$ form a **strong map sequence** if $g_1 \leftarrow g_2 \leftarrow \ldots \leftarrow g_k$.*

In [Top78] was one of the first to consider submodular functions that form a strong map sequence and its properties. Given submodular functions $g_1, g_2, \ldots, g_k \colon 2^S \to \mathbb{R}$ that form a strong map sequence, one might want to minimize all of them simultaneously. We denote this problem by **Strong Map Submodular Function Minimization**. Note that in the literature, what we denote as **Strong Map Submodular Function Minimization**, is also often called **Parametric Submodular Function Minimization**.

Strong Map Submodular Function Minimization

Instance:	$g_1, g_2, \ldots, g_k \colon 2^S \to \mathbb{R}$ that form a strong map sequence, $g_1 \leftarrow g_2 \leftarrow \ldots \leftarrow g_k$.
Task:	A minimal or maximal minimizer X_i for the submodular g_i for all $i \in \{1, \ldots, k\}$ together with the minimal value of g_i for all $i \in \{1, \ldots, k\}$.

Fleischer and Iwata [FI00] give an algorithm for this problem with running time $\mathcal{O}((n^7 + kn^2)\,\mathrm{EO} + n^8)$ by extending their push/relabel algorithm for submodular function minimization. A faster algorithm for minimizing all submodular function from a strong map sequence is provided by Nagano who extended the submodular function minimization algorithm of Orlin [Orl09]. The running time Nagano achieves is $\mathcal{O}((n^5 + kn^3)\,\mathrm{EO} + n^6)$. Thus, this algorithm has the same asymptotic running time as a single execution of the algorithm of Orlin as long as $k \in \mathcal{O}(n^2)$. The algorithm of Nagano also works if the submodular functions from the strong map sequence are computed in an on-line fashion during the execution of another algorithm.

Fact 2.6. *Using the algorithm of Nagano all submodular functions from a strong map sequence $g_1 \leftarrow g_2 \leftarrow \ldots \leftarrow g_k$ can be minimized in running time $\mathcal{O}((n^5 + kn^3)\,\mathrm{EO} + n^6)$. During the minimization a minimal or maximal minimizer of each submodular function g_i for all $i \in \{1, \ldots, k\}$ is computed.*

2.4 Classical Network Flows

The main topic of this thesis are flow over time problems. However, many ideas and methods that are used in the context of flows over time origin from the theory of classical network flows, which we call **static** network flows throughout this thesis.
The first book covering network flows is the book “Flows in Networks” by Ford and Fulkerson

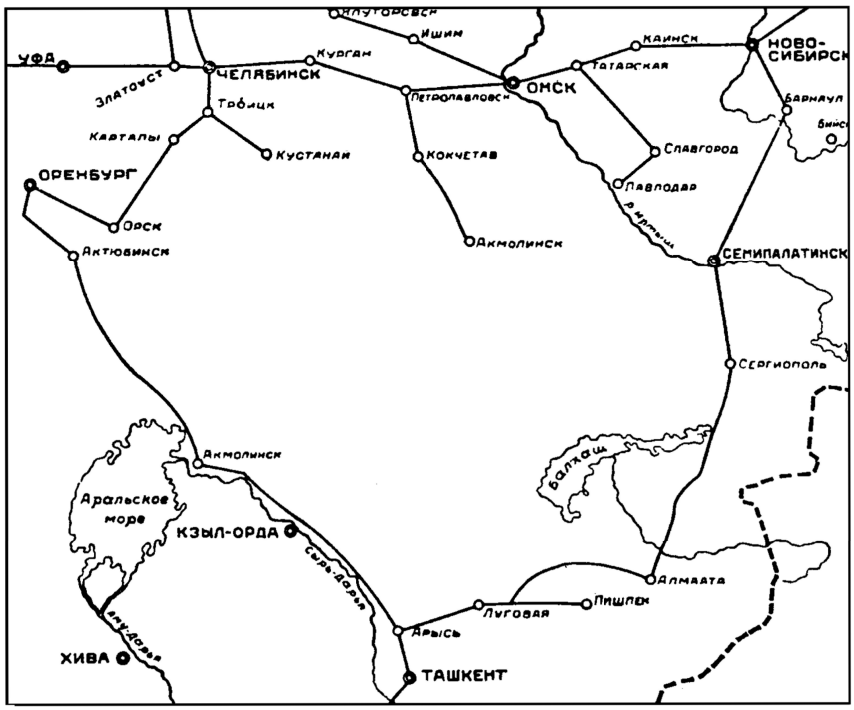


Figure 2.2: Figure from Tolstoĭ [Tol30] to illustrate a negative cycle

[FF62] published in 1962. In this book they also already introduced the concept of flows over time. For a thorough introduction to static network flows we refer the reader to Orlin et. al. [AMO93].

The History of Network Flows. As already stated, the first book on the topic of network flows was published in 1962 following a first peak of research regarding networks flows in the 1950s. However, the origins of network flow theory go back as far as 1930 when the Russian mathematician A.N. Tolstoĭ [Tol30] published a study on a transportation problem for which he developed a negative cycle criterion that he used to solve the problem to optimality (see Figure 2.2). Tolstoĭ was probably the first to notice that the negative cycle criterion is necessary for the optimality in a minimum-cost flow problem.
In the 1950s also the Americans became interested in the Soviet rail system. Instead of being actually interested in computing a maximum flow in this network, their objective was indeed to "cut off" the Soviet railway system from the western part of Europe in the cheapest way. This task was first posed in a RAND report by Harris and Ross [HR55] in 1955. Since the Soviet railway system was too large to handle, they at first describe a way to aggregate parts of the network to reduce the network size. The instance they achieve is shown in Figure 2.3. To find a maximum flow (or minimum cut) in this network they used a heuristic due to Boldyreff [Bol54], the so-called "flooding technique". In a their basic paper "Maximal Flow through a Network", which was first published as a RAND report in 1954 [FF54], Ford and Fulkerson suggest the simplex method for solving the maximum flow problem. However, Harris and Ross remark that the calculations required in the simplex method would be too cumbersome. In the same paper Ford and Fulkerson also show that the value of a minimum cut and the value of a maximum flow are equal. The same result was independently observed by Elias, Feinstein and Shannon [EFS56]. The well known augmenting path algorithm of Ford and Fulkerson was published a year later [FF55].

2.4.1 Basic Definitions and Results

The main object in the study of static network flows is the so-called **static network** $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ that consists of a directed graph $D = (V, A)$, an integral **capacity function**

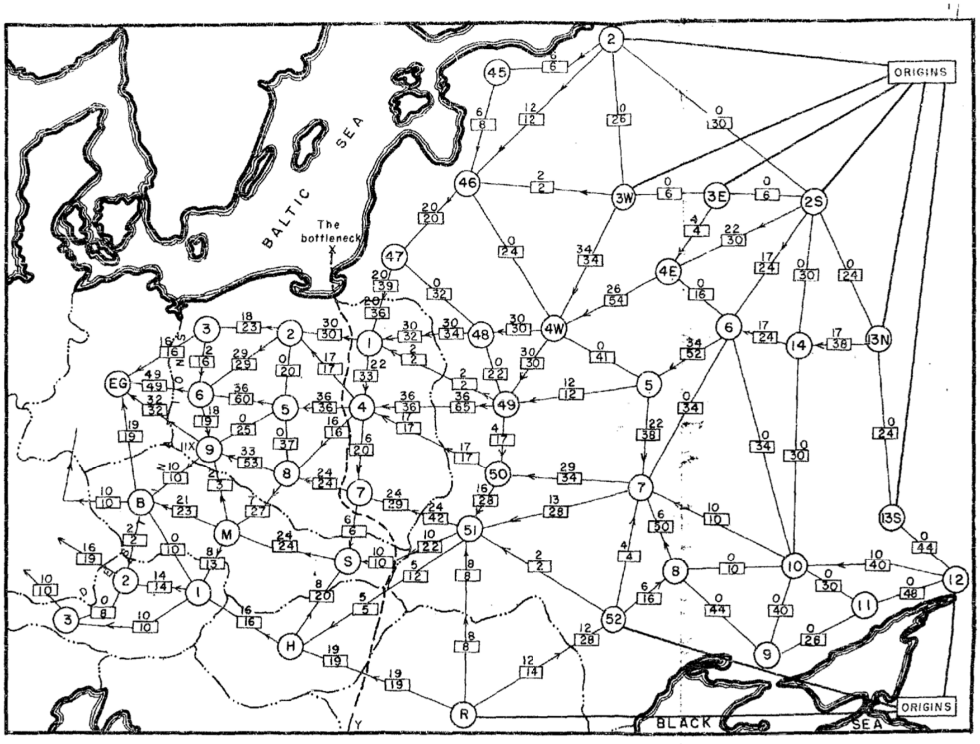


Figure 2.3: The network that Harris and Ross considered. The dashed line is the minimum cut.

$u\colon A \to \mathbb{Z}_{\geq 0}$ and subsets $S^+, S^- \subseteq V$ of vertices with $S^+ \cap S^- = \varnothing$ denoted as **sources** and **sinks**, respectively. We denote the union of sources and sinks, $S^+ \cup S^-$, as the set of **terminals** while the nodes v in $V \setminus (S^+ \cup S^-)$ are called **intermediate** nodes. In the case of networks with a single source $s \in V$ and a single sink $t \in V \setminus \{s\}$ we also write $\mathcal{N} = (D, u, s, t)$ instead of $\mathcal{N} = (D, u, \{s\}, \{t\})$. To simplify the notation we set $u_a \coloneqq u(a)$ for all $a \in A$.

One straightforward application of such networks is to model the transportation of goods from suppliers to customers. In such a setting the suppliers (sitting at the sources) usually can only deliver a limited number of goods, whereas the customers (located a the sinks) also have a limited demand. Such applications motivate to consider networks in which each source has some supply and each sink some demand. This is also the problem that Tolstoĭ considered in the 1930s.

The supplies and demands on the terminals are given by an integral **supply/demand-function** $b\colon S^+ \cup S^- \to \mathbb{Z}$ with

$$b(s) > 0 \text{ for all } s \in S^+,\ b(t) < 0 \text{ for all } t \in S^- \text{ and } \sum_{u \in S^+ \cup S^-} b(u) = 0.$$

To simplify the definition of static flows in networks we will throughout this thesis assume that the sources have no ingoing and the sinks no outgoing arcs. The following construction shows that we can make this assumption without loss of generality.

For a given static network $\mathcal{N} = (D, u, S^+, S^-)$ we create the modified static network $\mathcal{N}' \coloneqq (D' = (V', A'), u', S^{+\prime}, S^{-\prime})$ which is constructed as follows: To each source $s \in S^+$ we attach a super-source s' by an arc (s, s') with infinite capacity and to each sink $t \in S^-$ we attach a super-sink t' by an arc with infinite capacity. The sources and sinks of $\mathcal{N}'$ consist of these newly added super-sources and super-sinks, respectively. Overall we define,

$$\begin{aligned} S'^+ &\coloneqq \{s' \mid s \in S^+\} \text{ and } S'^- \coloneqq \{t' \mid t \in S^-\} \\ V' &\coloneqq S'^+ \cup S'^- \\ A' &\coloneqq A \cup \{(s', s) \mid s \in S^+\} \cup \{(t, t') \mid t \in S^-\}. \end{aligned}$$

The new capacity function $u' \colon A' \to \mathbb{Z}_+$ is defined as follows,

$$u'_a := \begin{cases} u_a & \text{if } a \in A \\ \infty & \text{if } a \notin A. \end{cases}$$

If for the original network $\mathcal{N}$ a supply/demand-function b was given, we shift these supplies and demands to the newly created sources and sinks, i.e. we define a new supply/demand-function $b' \colon S'^+ \cup S'^- \to \mathbb{Z}$ by setting $b'(s') := b(s)$ for all $s \in S^+$ and $b'(t') := b(t)$ for all $t \in S^-$. Clearly, the sources in $\mathcal{N}'$ do not have ingoing arcs and the sinks in $\mathcal{N}'$ do not have outgoing arcs. During the construction of $\mathcal{N}'$ at most $n = |V|$ new vertices are added to the original network $\mathcal{N}$. Thus, this construction can be done in running time $\mathcal{O}(n)$. Relying on the assumption that the sources do not have incoming arcs, whereas the sinks do not have outgoing arcs, static flows in networks are defined in the next paragraph.

Arc Flow. A **(static) flow** f in a static network $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ is a function $f \colon A \to \mathbb{R}_{\geq 0}$ such that on each arc the capacity is respected and for each intermediate node $v \in V \setminus (S^+ \cup S^-)$ it holds that exactly as much flow as flows into the node v also flows out of this node. Thus, f has to fulfill the **capacity constraint**

$$f(a) \leq u(a) \text{ for each } a \in A,$$

and **flow conservation**

$$\sum_{e \in \delta^-(v)} f(a) = \sum_{e \in \delta^+(v)} f(a) \text{ for each } v \in V \setminus (S^+ \cup S^-).$$

A network flow that also fulfills flow conservation on the terminals is called a **circulation**. The **value** $|f|$ of a flow f is the net amount of flow that flows out of the sources (and thus into the sinks),

$$|f| := \sum_{a \in \delta^+(S^+)} f(a).$$

Note, here we use the fact that the sources do not have incoming arcs.
For each $v \in V$ we denote by $\text{net}_f(v)$ the net amount of flow that **leaves** the node $v \in V$, i.e.,

$$\text{net}_f(v) := \sum_{a \in \delta^+(v)} f(a) - \sum_{a \in \delta^-(v)} f(a).$$

Clearly, in a static flow f we have that $\text{net}_f(v) = 0$ for all $v \in V \setminus (S^+ \cup S^-)$ because of the flow conservation. Thus, a static flow f in a static network $\mathcal{N}$ flows from the sources in S^+ to the sinks in S^-. This is why f is said to be a **S^+-S^- flow**. If the network $\mathcal{N}$ only has a single source s and a single sink t, then f is said to be an **s-t flow**.
If for each $a \in A$ the flow value $f(a)$ is integral, we say that f is an **integral** flow. To simplify notation, we set $f_a := f(a)$ for all $a \in A$ and a given static flow f in $\mathcal{N}$.

Path Flow. The previous definition of a static flow in a static network $\mathcal{N}$ is an edge based definition: A flow f in $\mathcal{N}$ fixes a flow value on each arc of $\mathcal{N}$ such that flow conservation and capacities are satisfied. However, intuitively in an S^+-S^- flow, f flow that contributes to the value of f travels along s-t paths for $s \in S^+$ and $t \in S^-$ (some flow also might travel along cycles). This motivates the following path based definition of a static flow in a static network $\mathcal{N}$.

Definition 2.7 (Path Flow).
*Let $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ be a network and $\mathcal{P}$ the set of all s-t paths for any pair of a source $s \in S^+$ and a sink $t \in S^-$. By $\mathcal{P}_a$ we denote the subset of paths using the arc $a \in A$, i.e., $\mathcal{P}_a \coloneqq \{P \in \mathcal{P} \mid a \in P\}$. A **path flow** x in $\mathcal{N}$ is an function $x \colon \mathcal{P} \to \mathbb{R}_{\geq 0}$ that respects the capacity constraints,*

$$\sum_{P \in \mathcal{P}_a} x(P) \leq u_a \text{ for all } a \in A.$$

*The **value** of x is defined to be $|x| \coloneqq \sum_{P \in \mathcal{P}} x(P)$. The set $\{P \in \mathcal{P} \mid x(P) > 0\}$ is called the set of **flow carrying paths**.*

Note that in the above definition the flow conservation constraint is not explicitly stated. A path flow x fulfills this constraint by definition as flow is only sent along s-t paths. Clearly, each path flow x in a static network $\mathcal{N}$ induces an edge flow f by defining $f(a) \coloneqq \sum_{P \in \mathcal{P}_a} x(P)$ for all $a \in A$. The other direction of this statement is not as straightforward but a consequence of the fact that each edge flow f in $\mathcal{N}$ can be decomposed into flow on S^+-S^- paths and flow along cycles:

Theorem 2.8 (Flow Decomposition, [Gal58; FF62]).
Let $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ be a static network and f a static S^+-S^- flow in $\mathcal{N}$. Then there exists a family $\overline{\mathcal{P}}$ of S^+-S^- paths in $\mathcal{N}$, a family $\overline{\mathcal{C}}$ of cycles in $\mathcal{N}$ and weights $w \colon \overline{\mathcal{P}} \cup \overline{\mathcal{C}} \to \mathbb{R}_+$ such that

$$\begin{aligned} f(a) &= \sum_{P \in \overline{\mathcal{P}} \cup \overline{\mathcal{C}} \colon a \in P} w(P) \text{ for all } a \in A, \\ |f| &= \sum_{P \in \overline{\mathcal{P}}} w(P), \\ |\overline{\mathcal{P}}| + |\overline{\mathcal{C}}| &\leq |A|. \end{aligned}$$

Additionally, it holds that if f is integral, then all weights can be chosen integral.

The proof of Theorem 2.8 can for example be found in Schrijver [Sch03]. It also yields an algorithm with running time in $\mathcal{O}(mn)$ to compute a flow decomposition of a given static flow f in a static network $\mathcal{N}$.
Given an edge flow f in $\mathcal{N}$ we can thus construct a corresponding path flow x with the same value as follows: compute the flow decomposition of f, given by paths $\overline{\mathcal{P}}$ and cycles $\overline{\mathcal{C}}$ and afterwards define for all $P \in \mathcal{P}$

$$x(P) \coloneqq \begin{cases} w(P) & \text{if } P \in \overline{\mathcal{P}} \\ 0 & \text{if } P \in \mathcal{P} \setminus \overline{\mathcal{P}} \end{cases}.$$

Note that we do not have $x(a) = f(a)$ for all $a \in A$. Overall, the following lemma holds:

Lemma 2.9. *For each static arc flow f in a static network $\mathcal{N}$ there exists a path flow x with the same value and vice versa.*

2.4.2 The Maximum Flow Problem

The by far most famous network flow problem is the **Maximum Flow Problem** that has already been introduced in the 1950s by Harris and Ross [HR55]. The aim of this problem, which is usually defined in networks with a single source s and a single sink t, is to compute an s-t flow with the maximum possible value:

MAXIMUM FLOW PROBLEM

Instance: A static network $\mathcal{N} = (D = (V, A), u, s, t)$

Task: Find a network flow f in $\mathcal{N}$ with maximum value $|f|$

Max-Flow Min-Cut Theorem. The max-flow min-cut theorem (a generalization of Menger's Theorem [Men27]), which states that the value of a maximum flow in $\mathcal{N}$ equals the minimal capacity of a cut in $\mathcal{N}$ separating the sources from the sinks, was first established in 1956 independently by Ford and Fulkerson [FF56], and Elias, Feinstein and Shannon [EFS56]. Both proofs are combinatorial proofs. Moreover, the max-flow min-cut theorem is also a special case of the strong duality theorem of linear programming because the LPs for the max-flow problem and the min-cut problem are in fact dual LPs. As the name suggests the minimum cut problem is the problem of finding a cut with minimal capacity that separates the source s from the sink t of a given network $\mathcal{N}$.

MINIMUM CUT PROBLEM

Instance: A network $\mathcal{N} = (D = (V, A), u, s, t)$

Task: Find the cut C with minimal capacity separating s and t

The max-flow min-cut theorem states that the capacity of a minimal cut is equal to the value of a maximum flow in a given static network $\mathcal{N}$.

Theorem 2.10 (Max-Flow Min-Cut Theorem, [FF56; EFS56]).

Let $\mathcal{N} = (D = (V, A), u, s, t)$ be a network. The maximum value of an s-t flow equals the minimum capacity of an s - t cut.

Usually, **maximum flow problem** is defined in static networks with only a single source s and a single sink t. This, however, is no restriction, as was already noted by Ford and Fulkerson [FF62]. Let $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ be a static network with multiple sources and sinks and assume that we want to compute a maximum S^+-S^- flow in $\mathcal{N}$. Using an **extended** network $\mathcal{N}^*$ that arises from the original network $\mathcal{N}$ by attaching a **super-source** s^* and a **super-sink** t^* we can define an equivalent maximum flow problem in the network $\mathcal{N}^*$ (see Figure 2.4 for a visualization of $\mathcal{N}^*$). More precisely, the **extended network** $\mathcal{N}^* = (D^* = (V^*, A^*), u^*, s^*, t^*)$ is obtained by

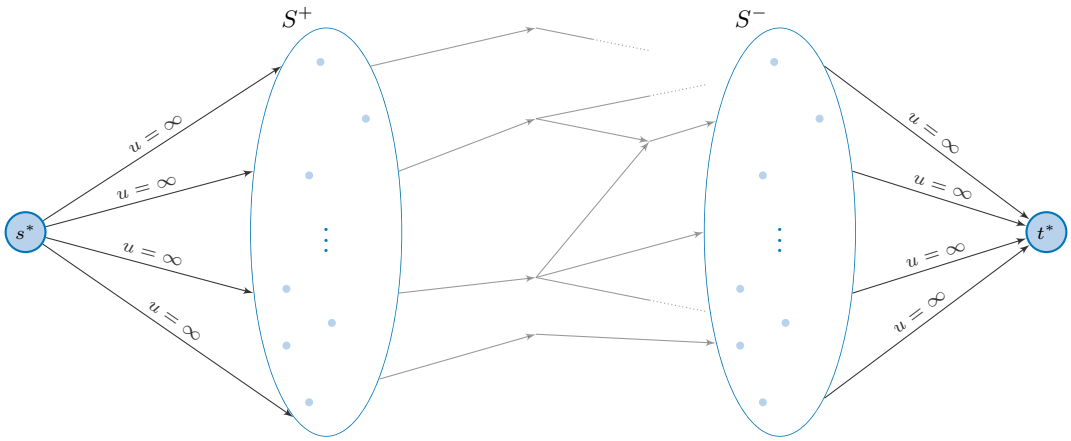


Figure 2.4: The extended network $\mathcal{N}^*$

adding two new nodes s^* and t^* to V, i.e., $V^* \coloneqq V \cup \{s^*, t^*\}$. The **super-source** s^* is connected to every node in S^+ by an arc with infinite capacity and t^* to every sink in S^- by an arc with infinite capacity. Thus,

$$A^* \coloneqq A \cup \bigcup_{s \in S^+} \{(s^*, s)\} \cup \bigcup_{t \in S^-} \{(t, t^*)\},$$

while the new capacity function u^* is defined as

$$\begin{aligned} u^*_{(s^*,s)} &\coloneqq \infty \text{ for all } s \in S^+, \\ u^*_{(t,t^*)} &\coloneqq \infty \text{ for all } t \in S^-, \\ u^*_a &\coloneqq u_a \text{ for all } a \in A. \end{aligned}$$

We conclude with the following observation that yields that a maximum flow in $\mathcal{N}$ can be obtained from a maximum flow in the extended network $\mathcal{N}^*$.

Observation 2.11 ([FF62]). *Let $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ be a static network. The values of a maximum flow in $\mathcal{N}$ and in the extended network $\mathcal{N}^*$ are the same and a maximum flow in $\mathcal{N}$ can be obtained from a maximum flow on $\mathcal{N}^*$ and vice versa.*

During this thesis we denote by $\max_{\mathcal{N}}(S, T)$ the **value of a maximum flow from S to T** in a static network $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ for $S \subseteq S^+$ and $T \subseteq S^-$.

The Transshipment Problem. A problem related to the maximum flow problem is the so-called **Transshipment Problem**. Assume we are given a static network $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ together with a supply/demand function $b\colon S^+ \cup S^- \to \mathbb{Z}$ on the terminals. Now, the goal is not to find a maximum S^+-S^- flow in $\mathcal{N}$ but a flow satisfying all supplies and demands. Such a flow is called a **transshipment**, a **b-flow** or **b-transshipment**. For the transshipment problem given by $\mathcal{N}$ and b we shortly write $(\mathcal{N}, b)$. An application that can be modeled by transshipment problems is the transportation of goods from suppliers with bounded supplies to customers with bounded demands.

Transshipment Problem $(\mathcal{N}, b)$

Instance:	A static network $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ and a supply/demand function b on the terminals
Task:	Find a static network flow f in $\mathcal{N}$ that satisfies the given supplies and demands, i.e., $\sum_{a \in \delta^+(s)} f(a) = b(s)$ and $\sum_{a \in \delta^-(t)} f(t) = -b(t)$ for all sources $s \in S^+$ and sinks $t \in S^-$, respectively

Using an extended network, we can transform a given transshipment problem into a maximum flow problem. We again construct the extended network $\mathcal{N}^*$ as above but now we set the capacities on the arcs connected to the super-source s^* according to the capacities of the specific original source. The same is done for the arcs connected to the super-sink t^*. More precisely, we define

$$\begin{aligned} u^*_{(s^*,s)} &\coloneqq b(s) \text{ for all } s \in S^+, \\ u^*_{(t,t^*)} &\coloneqq -b(t) \text{ for all } t \in S^-, \\ u^*_a &\coloneqq u_a \text{ for all } a \in A. \end{aligned}$$

The new supply on s^* is defined to be $b^*(s^*) \coloneqq \sum_{s \in S^+} b(s)$ and the new demand of t^* is $b^*(t^*) \coloneqq \sum_{t \in S^-} b(t^*)$.

Observation 2.12. *Let $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ be a static network, b a supply/demand function on the terminals $S^+ \cup S^-$, and $\mathcal{N}^*$ the corresponding extended network with the new supply/demand function b^*. A transshipment solving $(\mathcal{N}, b)$ does exist in $\mathcal{N}$ if and only if a transshipment solving $(\mathcal{N}^*, b^*)$ does exist in $\mathcal{N}^*$. A transshipment solving $(\mathcal{N}^*, b^*)$ does exist in $\mathcal{N}^*$ if and only if the value of a maximum s^*-t^* flow in $\mathcal{N}^*$ is equal to $\sum_{s \in S^+} b(s)$.*

Computing Maximum Flows. Clearly, one way to compute maximum flows is to solve the corresponding linear program using for example the simplex method. In this section we are giving a brief introduction into the main ideas behind *combinatorial* algorithms for the maximum flow problem.

A concept central to many network flow algorithms is the **residual network** of a given static network flow f. Using the residual network allows algorithms to essentially take back flow that has been sent in earlier iterations of an algorithm. The main idea of the construction is to consider the bidirected digraph and to let the reverse arcs account for the flow that has already been sent.

Definition 2.13.
*Let $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ be a static network and f an edge based static flow in $\mathcal{N}$. The **residual network** $\mathcal{N}_f := (\overleftrightarrow{D}, u_f, S^+, S^-)$ consists of the bidirected graph $\overleftrightarrow{D}$, the original sources S^+ and sinks S^- and the **residual capacity** u_f with respect to f defined as*

$$u_f(a) := \begin{cases} u_a - f_a & \text{if } a \in A \\ f_a & \text{if } a \in \overleftrightarrow{A}. \end{cases}$$

Most algorithms for the maximum flow problem are executed in the residual network. There are essentially two classes of maximum flow algorithms: the algorithms that rely on augmenting flow along so-called **augmenting paths** and the so-called **push/relabel** algorithms. The first algorithm for the maximum flow problem, which was developed by Ford and Fulkerson [FF62], belongs to the first class.

Definition 2.14.
*Let $\mathcal{N}$ be a static network and f a static low in $\mathcal{N}$, P a path or a cycle in $\mathcal{N}_f$ and $\gamma > 0$. To **augment f by γ along P** means to increase the flow by γ on each arc $a \in P$ with $a \in A$ and to decrease the flow value by γ on each $a \in P$ with $\overleftarrow{a} \in A$.*
*An s-t path P for $s \in S^+$ and $t \in S^-$ in the residual network $\mathcal{N}_f$ with $\gamma := \min_{a \in P} u_f(a) > 0$ is called an s-t **augmenting path**, because augmenting f along P by γ yields a network flow with a strictly larger value.*

If for a given flow static f there exists an s-t augmenting path with $s \in S^+$ and $t \in S^-$ in $\mathcal{N}_f$, then f cannot be a maximum flow. The converse direction of this statement also holds as was shown by Ford and Fulkerson in [FF55].

Theorem 2.15 ([FF55]).
A static S^+-S^- flow f is a maximum flow in a given static network $\mathcal{N}$ if and only if there is no augmenting path in the residual network $\mathcal{N}_f$.

The first maximum flow algorithm incorporating Theorem 2.15 was the algorithm of Ford and Fulkerson [FF55; FF62]. It just consists of a loop that augments flow along an arbitrary augmenting path in the residual network until no such path can be found anymore. When the algorithm terminates, the current flow in the network is maximal according to Theorem 2.15. Ford and Fulkerson already noticed that their algorithms does not necessarily terminate if the arc capacities are irrational numbers and even if all arc capacities are integers, a bad way of choosing the augmenting path in each iteration might result in an exponential worst case running time. In general the running time of the algorithm of the maximum flow algorithm of Ford and Fulkerson is in $\mathcal{O}(U \cdot m \cdot n)$ with

n and m being the number of nodes and arcs, respectively and U being the highest arc capacity, and thus the running time may be pseudo-polynomial.
Edmonds and Karp (1972) [EK72], and independently Dinic (1970) [Din70], observed that the running time of the algorithm of Ford and Fulkerson can be considerably improved if the augmenting paths are not chosen arbitrarily but in a smart way. Edmonds and Karp obtain a maximum flow algorithm with a strongly polynomial running time in $\mathcal{O}(m^2 \cdot n)$ by choosing the shortest augmenting path in each iteration. Dinic does not only use augmenting paths in his algorithm but augments along so-called *blocking flows* achieving an even better running time in $\mathcal{O}(n^2 \cdot m)$. Two years later Dinic' algorithms was improved by Karzanov yielding a running time in $\mathcal{O}(n^3)$ [Kar74; MKM78]. In the subsequent years faster algorithm using blocking flows have been developed by Cherkasskĭ [Che77] ($\mathcal{O}(n^2 \cdot \sqrt{n})$), Galil [Gal78] ($\mathcal{O}(n \cdot (nm)^{2/3})$), Shiloach [Shi78] , Galil and Naamad [GN80] ($\mathcal{O}(nm \log^2 n)$), Sleator [Sle80], and Sleator and Tarjan [ST83] ($\mathcal{O}(nm \log n)$). More running time improvements were obtained with a new technique introduced by Goldberg and Tarjan [GT88]. They developed the first algorithm relying on the push-relabel (or preflow-push) technique and obtained a running time in $\mathcal{O}(nm \log(n^2/m))$. By constructing better selection rules and using more sophisticated data structures eventually the algorithm of King, Rao and Tarjan [KRT94] was obtained with a running time of $\mathcal{O}(nm \cdot \log_{m/n \log(n)} n)$. Note that a decomposition of flows into paths can have a size of $\Omega(nm)$. Thus $\mathcal{O}(nm)$ is natural target bound for algorithms solving the maximum flow problem. It was a long-standing open question whether an algorithm with this running time exists which was answered by Orlin in 2013 who developed a maximum flow algorithm with a running time in $\mathcal{O}(nm)$. However, the flow decomposition size is *not* a lower bound for computing maximum flows as a flow can be represented in $\mathcal{O}(m)$ space and dynamic trees can be used to augment flow on a path in logarithmic time. The algorithm of Orlin is the fastest known algorithm for the maximum flow problem with strongly polynomial time running time. However, in the recent years also much improvement has been achieved regarding weakly polynomial time algorithms:
The unit capacity problem on a graph with no parallel arcs can be solved in $\mathcal{O}(\min\{n^{2/3}, \sqrt{m}\} \cdot m)$ (which is much faster than $\mathcal{O}(mn)$), which was shown independently by Karzanov [Kar74] and Even and Tarjan [ET75]. For 25 years there was a big gap between the unit capacity case and the general case which was narrowed in 1998 by Goldberg and Rao [GR98], who obtained an $\mathcal{O}(\min\{n^{2/3}, \sqrt{m}\} \cdot m \log(n^2/m) \log(U))$ algorithm for the problem with integral capacities. In the context of undirected graph a recent series of papers, including Christiano et al. [Chr+11], Kelner et al. [Kel+14], Lee et al. [LRS13] and Sherman [She13] studied the problem of finding an approximately maximum flow (within a factor of $(1 + \varepsilon)$ of maximum) in undirected graphs. This line of work was culminated by Sherman [She13] and Kelner et al. [Kel+14] who independently achieve an approximation algorithm for maximum flows with nearly linear running time. All these papers used linear algebraic techniques and electrical flows. Building on this work, Mądry [Mąd13] in 2013 obtained an exact algorithm for unit capacity flows in directed graphs with a running time in $\widetilde{\mathcal{O}}(m^{10/7})$. The first improvement over the algorithm by Goldberg and Rao was achieved in 2014 by Lee and Sidford [LS14] who developed a new interior point method which led to an algorithm for the maximum flow problem with a running time in $\widetilde{\mathcal{O}}(m\sqrt{n} \log^{\mathcal{O}(1)} U)$. For the special case of solving the maximum flow problem in dense directed graphs with unit capacities their algorithm also improves upon the algorithm of Mądry. Two years later Mądry [Mąd16] published a new algorithm for the maximum flow problem in directed graphs with integer capacities with a running time in $\widetilde{\mathcal{O}}(m^{10/7} U^{1/7})$. This algorithms improves over the algorithm of Lee and Sidford whenever U is moderately large and the graph is sufficiently sparse.

2.4.3 The Minimum-Cost Flow Problem

Another classical network flow problem is the **Minimum-Cost Flow Problem**, which appears in the literature in many different variants. It is sometimes (especially in old papers from the beginnings of network flow theorem) referred to as **transportation problem**. The minimum-cost flow problem is as old as the theory of network flows. One of the first to study this problem was the Russian mathematician Tolstoĭ[Tol30].

Let $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ be a static and $c\colon A \to \mathbb{R}$ a **cost function** on the arcs. For any static flow f in $\mathcal{N}$ the **cost** of f is defined to be

$$c(f) \coloneqq \sum_{a \in A} c(a) \cdot f(a).$$

Using this notion of cost, we define two variants of the minimum-cost flow problem. Given a static network $\mathcal{N} = (D = (V, A), u, S^+, S^-)$, a supply/demand function b and a cost function $c\colon A \to \mathbb{R}$ on the arcs, a minimum-cost flow in this setting is a static flow f that satisfies all supplies and demands while being of minimum cost.

MINIMUM-COST FLOW PROBLEM $(\mathcal{N}, b, c)$

Instance: A static network $\mathcal{N} = (D = (V, A), u, S^+, S^-)$, a supply/demand function b on the terminals and a cost function c on the arcs, or shortly $(\mathcal{N}, b, c)$.

Task: Find a static network flow f in $\mathcal{N}$ that satisfies the given supplies and demands with minimum cost $c(f)$

In other words the minimum-cost flow problem given by $(\mathcal{N}, b, c)$ is the problem of finding a b-flow of minimum cost. A special case of the minimum-cost flow problem is the **Minimum-Cost Circulation Problem**:

MINIMUM-COST CIRCULATION PROBLEM

Instance: A static network $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ and a cost function c on the arcs A

Task: Find a circulation f in $\mathcal{N}$ with minimum cost $c(f)$

Clearly, an instance of the minimum-cost circulation problem can be transformed to an instance of the minimum-cost flow problem by defining an all zero supply/demand function, $b \equiv 0$.

Another straightforward observation is that, given an instance of the minimum-cost circulation problem in a static network $\mathcal{N}$, it is only possible to obtain a circulation f that is different from the zero flow if $\mathcal{N}$ contains a cycle of negative cost. In the general variant of the minimum-cost flow problem we might, however, also have to send flow along expensive paths with positive costs in order to be able to fulfill the supplies and demands.
For solving the minimum-cost flow problem and the minimum cost circulation problem, again the residual network $\mathcal{N}_f$ corresponding to a given b-flow or circulation f is central.

The **residual network** $\mathcal{N}_f$ corresponding to some static flow f in a static network $\mathcal{N}$ with a cost function c on the arcs is defined exactly as in Definition 2.13 with the addition that we also extend the cost function to $\mathcal{N}_f$ by setting

$$c(\overleftarrow{a}) \coloneqq -c(a) \text{ for all } a \in A.$$

If f is a circulation or a b-flow in $\mathcal{N}$ and $\mathcal{N}_f$ contains a cycle C of negative cost with $0 < \gamma \coloneqq \min_{a \in C} u_f(a)$, we can augment f along C by $\gamma = \min_{a \in P} u_a$ and obtain a new circulation or b-flow with lower cost. Thus, if $\mathcal{N}_f$ contains a cycle of negative cost, then f is not a minimum cost circulation or a b-flow of minimal cost, respectively. Again it turns out that the other direction of this statement also holds:

Theorem 2.16 ([Kle67]).

Let $(\mathcal{N}, b, c)$ be an instance of the minimum-cost flow problem and f a b-flow with respect to that. Then f is of minimal cost if and only if $\mathcal{N}_f$ contains no cycle of negative cost.

The above negative cycle criteria was for example derived by Klein in 1967 [Kle67]. The idea however goes back to Tolstoĭ [Tol30] and was rediscovered several times in different forms for example by Robinson [Rob50], Gallai [Gal58], Busacker and Gowen [BG60] and Fulkerson [Ful61].

Solving the minimum-cost flow Problem. The minimum-cost flow problem can be formulated as a linear program and can thus be solved in strongly polynomial time by using the ellipsoid method. Historically, before combinatorial algorithms for the minimum-cost flow were developed, specializations of the simplex algorithm for this specific problem were found.
In 1941 Hitchcock studied a variant of the minimum-cost flow problem – the **Hitchcock Problem** – and also developed a solving procedure inspired by Dantzig's simplex algorithm[Hit41]. Ten years later Dantzig showed how his simplex algorithm specializes to yield an algorithm for the minimum-cost flow problem [Dan51] and thus developed the first version of the **Network Simplex Algorithm**. Although network simplex algorithms perform fast in practice, the existence of a provable efficient network simplex algorithm was a long-standing open problem. The first network simplex with a polynomial running time in $\mathcal{O}(n^2 m \log(nC))$, where C is the maximal cost on the arcs, was provided in 1995 by Orlin [Orl97]. His algorithm was improved 2 years later by Tarjan [Tar97] who achieved a running time of $\mathcal{O}(nm \log(n \log(nC)))$ using dynamic trees. It is still an open problem whether there is a strongly polynomial time network simplex algorithm.
In contrast to that, a strongly polynomial time dual network simplex algorithm has already been developed in 1985 by Orlin [Orl85].
For a combinatorial algorithm to solve the minimum-cost flow problem it is a straightforward idea to use Theorem 2.16. In their book about network flows Ford and Fulkerson published the first minimum-cost flow algorithm relying on the negative cycle criterion. However, in the worst case their algorithm is of exponential running time.
The first combinatorial weakly polynomial time algorithm for the minimum-cost flow problem is due to Edmonds and Karp [EK72], who also raised the question whether a strongly polynomial time algorithm for the minimum-cost flow problem does exist.
This question was answered in 1985 by Tardos who gave the first strongly polynomial time algorithm for this problem. The currently fastest known strongly polynomial time algorithm for the minimum-cost flow problem with a running time of $\mathcal{O}((m \log m) \cdot (m + n \log n))$ is due to Orlin [Orl93].
The algorithm which is of most interest throughout this thesis – the **successive shortest path algorithm (SSPA)** – is not of polynomial running time. However, it plays a central role in the context of many flow over time problems and this is why we describe it in more detail. The following theorem, which has been shown independently by Jewell [Jew58], Iri [Iri60] and Busacker, and Gowen [BG60], immediately leads to the successive shortest path algorithm.

Theorem 2.17 ([Jew58; Iri60; BG60]).

Let $(\mathcal{N}, b, c)$ be an instance of the minimum-cost flow problem and f a b-flow of minimal cost. Let P be augmenting s-t path in $\mathcal{N}_f$ for $s \in S^+$ and $t \in S^-$ that is of minimum cost with respect to c. Let f' be the flow in $\mathcal{N}$ that is the result of augmenting f along P by at most the minimal residual capacity of P. Then f' is a b'-flow of minimal cost for a suitable choice of b'.

The above theorem leads towards the successive shortest path algorithm shown in Algorithm 3. For the algorithm we assume that the cost function is nonnegative. This assumption can be made without loss of generality (see [AMO93]).

Clearly, the algorithm terminates after at most $B := \sum_{sS^+ \cup S^-} |b(s)|$ augmentations resulting in a pseudo-polynomial running time in $\mathcal{O}(nm + B(m + n \log n))$ [Tom71; EK72]. An example instance in which the successive shortest path algorithm needs pseudo-polynomially many iterations was first given by Zadeh [Zad73] in 1973. A more compact construction is given in [DS15]. However, if $B \in \mathcal{O}(n)$ the successive shortest path algorithm is the fastest known algorithm for the minimum-cost flow problem.

Algorithm 3: Algorithm for solving a given minimum-cost flow problem, SSPA$(\mathcal{N}, b, c)$

```
Input  : An instance (N, b, c) of the minimum-cost flow problem
Output : A b-flow with minimal costs if a b-flow exists, FALSE otherwise
b' ← b and f(a) ← 0 for all a ∈ A
while b' ≠ 0 do
    choose a node s with b'(s) > 0
    choose a node t with b'(t) < 0 such that t can be reached from s in N_f
    if such a t does not exist then
        return FALSE
    else
        choose an augmenting s-t path P in N_f of minimal cost with respect to c
        γ ← min{min_{a∈P} u_f, b'(s), −b'(t)}
        b'(s) ← b'(s) − γ and b'(t) ← b'(t) + γ and augment f along P by γ
    end
end
return f
```

2.4.4 Lexicographically Maximum Flows

Let $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ be a static network. For some applications it is important that not just a maximum flow from the sources to the sinks is computed, but that the terminals also have an internal priority order with respect to which the net flow out of the terminals should be maximized. Hence, we now additionally assume that we are given a total order $\prec$ on the terminals $S^+ \cup S^-$. For simplicity we also assume that $S^+ \cup S^- = \{s_1, s_2, \dots, s_k\}$ with $s_i \prec s_j$ for all $i, j \in \{1, \dots, k\}$ with $i < j$. If $s \prec s'$ for $s, s' \in S^+ \cup S^-$ we say that s has a **higher priority** than s'. A flow f^2 in $\mathcal{N}$ is **lexicographically bigger** than a flow f^1, or $f^1 \leq_L f^2$, with respect to $\prec$ if

$$\exists l \in \{0, 1, \dots, k-1\}\colon |\operatorname{net}_{f^1}(s_i)| = |\operatorname{net}_{f^2}(s_i)| \text{ for all } i \in \{1, 2, \dots, l\} \text{ and } |\operatorname{net}_{f^1}(s_{l+1})| < |\operatorname{net}_{f^2}(s_{l+1})|,$$

or if

$$|\operatorname{net}_{f^1}(s_i)| = |\operatorname{net}_{f^2}(s_i)| \text{ for all } i \in \{1, 2 \dots, k\}.$$

A static flow f in $\mathcal{N}$ is a **lexicographically maximum (lex-max) flow** with respect to $\prec$ if it is lexicographically bigger than any other static flow f' in $\mathcal{N}$. The **lexicographically maximum flow problem** is the problem of finding a lexicographically maximal flow with respect to some total order on the terminals.

> LEXICOGRAPHICALLY MAXIMUM FLOW PROBLEM
>
> *Instance:* A static network $\mathcal{N} = (D = (V, A), u, S^+, S^-)$ and a total order $\prec$ on the terminals $S^+ \cup S^-$
>
> *Task:* Compute a static flow f in $\mathcal{N}$ that is a lexicographically maximum flow with respect to $\prec$

A lex-max flow with respect to a given order $\prec$ on the terminals is thus a flow that maximizes the amount of flow sent out of the sources within the given order but *minimizes* the amount of flow sent into the sinks within the order $\prec$.
The existence of a lex-max um flow with respect to any total order $\prec$ on the terminals has been shown by Minieka [Min73] and Megiddo [Meg74]. Minieka also gives an algorithm to compute lex-max flows that relies on at most $|S^+ \cup S^-|$ many maximum flow computation. Thus, such flows can be obtained in strongly polynomial time. One theorem that the existence proof of Minieka relies

on is the following results which basically states that the arrival and departure patterns of maximum flows are independent.

Theorem 2.18 ([Min73]).
Let f^1 and f^2 be two maximum flows in $\mathcal{N}$. There exists a third maximum flow f^3 in $\mathcal{N}$ with

$$\text{net}_{f^3}(s) = \text{net}_{f^1}(s) \text{ for all } s \in S^+ \text{ and } \text{net}_{f^3}(t) = \text{net}_{f^2}(t) \text{ for all } t \in S^-.$$

That is, f^3 has the departure pattern *of f^1 and the* arrival pattern *of f^2.*

For the proof of this theorem, the whole existence proof of lex-max flows and the algorithm of Minieka [Min73] for computing lex-max flow we refer to [Min73]. As already stated the algorithm of Minieka [Min73] needs $|S^+ \cup S^-|$ many maximum flow computations in order to compute a lex-max flow with respect to a given order. In static networks with a single sink t (or a single source s) there is a faster way to compute static lex-max flows that uses an algorithm by Gallo, Grigoriadis and Tarjan for computing parametric maximum flows [GGT89]. Gallo et al. address the problem of computing maximum flows for each member of an increasing sequence of parameter values $\lambda_1 < \lambda_2 < \ldots < \lambda_l$ in a static network $\mathcal{N}_\lambda = (D, u_\lambda, \{s\}, \{t\})$ in which the arc capacities depend on a parameter λ. Their main result is to extend the maximum flow algorithm of Goldberg and Tarjan to find maximum flows in a special class of parametrized networks for $\mathcal{O}(n)$ ordered values of the parameter in $\mathcal{O}(nm \log(n^2/m))$ with $n = |V|$. That is, Gallo et al. show how to compute $\mathcal{O}(n)$ maximum flows in the same running time that *only one* maximum flow computation using the algorithm of Goldberg and Tarjan requires. The class of parametrized static networks that Gallo et al. consider are those in which the capacities of arcs leaving the source are non-decreasing functions of the parameter, and those of arcs entering the sink are non-increasing functions of the parameter, and those of all other arcs are constant. Thus, they regard parametrized static networks $\mathcal{N}_\lambda = (D, u_\lambda, \{s\}, \{t\})$ with

- $u_\lambda(a)$ is a non-decreasing function of λ for all $a \in A$ with $a = (s, u)$ for some $u \in V$,
- $u_\lambda(a)$ is a non-increasing function of λ for all $a \in A$ with $a = (v, t)$ for some $v \in V$,
- $u_\lambda(a)$ is constant for all $a \in A$ with $a = (u, v)$ for $u \in V \setminus \{s\}$ and $v \in V \setminus \{t\}$.

A parametrized static network $\mathcal{N}_\lambda$ with these properties is called a **GGT network**. Denote by $C_\lambda \subseteq V \setminus \{t\}$ a subset of vertices that induces a minimum s-t cut in $\mathcal{N}_\lambda$ for the parameter value λ. The main structural property of such parametrized networks that Gallo et al. exploit is that the sets C_λ are nested with growing λ, that is $C_{\lambda_1} \subseteq C_{\lambda_2}$ with $\lambda_1 \leq \lambda_2$. Summarizing, given a parametrized static GGT network $\mathcal{N}_\lambda = (D, u_\lambda, \{s\}, \{t\})$ and $\lambda_1 < \lambda_2 < \ldots < \lambda_l$ with $l \in \mathcal{O}(|V|)$, the algorithm of Gallo et al. returns maximum flows $f_1, f_2, \ldots, f_k$ and minimum cuts $C_{\lambda_1}, C_{\lambda_2}, \ldots, C_{\lambda_l}$ for all parameter values $\lambda_1, \ldots, \lambda_l$, respectively, in the same running time as required by the maximum flow algorithm of Goldberg and Tarjan. We will now argue how to solve a lex-max flow problem in a static network with only a single sink t using the algorithm of Gallo et al. (the case with only a single source works symmetrically). For this purpose let $\mathcal{N}$ be a static network with only a single sink t and $\prec$ a total order on $S^+ \cup \{t\}$. The goal is to compute a static lex-max flow in $\mathcal{N}$ with respect to $\prec$. In order to obtain a GGT network we at first attach a super-source s^* to the sources of $\mathcal{N}$. The capacities of the arcs by which the super-source is connected to the sources of $\mathcal{N}$ are parametrized by a parameter λ. Without loss of generality let $S^+ = \{s_1, s_2, \ldots, s_k\}$ with $s_1 \prec s_2 \prec \ldots \prec s_k \prec t$. We define for all $i \in \{1, \ldots, k\}$,

$$u_\lambda(s^*, s_i) := \begin{cases} 0 \text{ for all } \lambda \in [0, i) \\ \infty \text{ for all } \lambda \in [i, \infty). \end{cases}$$

The capacities of all other arcs remain constant in λ. We denote the resulting parametric network by $\mathcal{N}_\lambda$. By definition this network is a GGT network. We now define a sequence of parameters $\lambda_1 < \lambda_2 < \ldots \lambda_k$ by $\lambda_i := i$. When looking into the way the algorithm of Gallo et al. works in more detail, it turns out that when applying this algorithm to $\mathcal{N}_\lambda$ and $\lambda_1 < \lambda_2 < \ldots \lambda_k$ defined as above, this algorithm in fact computes a static lex-max flow with respect to $\prec$ in $\mathcal{N}$. More precisely, the

flow f_k returned by this algorithm is a suitable lex-max flow when restricted to $\mathcal{N}$. We conclude this section with the following lemma about static lex-max flows.

Lemma 2.19. *Let $\mathcal{N}$ be a network, $\prec$ an order on the terminals and $f_\prec$ a lex-max flow with respect to $\prec$. The flow $f_\prec$ fulfills for all $s \in S^+ \cup S^-$,*

$$\mathrm{net}_f(\{s' \in S^+ \cup S^- \mid s' \preceq s\}) = \max_{\mathcal{N}}(S^+ \cap \{s' \in S^+ \mid s' \preceq s\}, S^- \setminus \{s' \in S^- \mid s' \preceq s\}).$$

For the proof we again refer to Minieka [Min73]

2.5 Flows Over Time

With classical network flows it is possible to obtain excellent results in static situations. However, once an application depends on a temporal dimension, static networks flows are not able to capture this dependence on time.
In order to be able to model applications that depend on time. Ford and Fulkerson introduced **flows over time** [FF58; FF62]. In contrast to the setting in static network flows, in the dynamic setting each arc a additionally has a **transit time** $\tau(a)$. Flow entering an arc at time $\theta \geq 0$ will leave the arc after time $\theta + \tau(a)$. In this time-dependent setting the capacity of an arc is not an upper bound on the overall amount flow that is allowed to be on an arc, but it represents the flow that can enter an arc *per time*. A **flow over time** f now specifies a flow value (that is, the rate at which flow enters the arc) for each arc a for each point in time. In Ford and Fulkerson's setting time is *discrete* and also the flow is discrete: a discrete flow unit enters an arc and travels along this arc as a whole arriving at the end of the arc according to the transit time. Later, a **continuous** model was introduced, in which the time and also the flow "flows continuously".
In this section we present the most basic definitions and techniques in the context of flows over time. In the next chapter we will give a more thorough presentation of the more advanced problems that concern us throughout this thesis. For an introduction to flows over time see [Sku09].

2.5.1 Basic Definitions

As in the study of static flows, the main object when considering flows over time is a network, which we call the **dynamic network** denoted by $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$. Here, $D = (V, A)$ is a directed graph with vertices V and arcs A and S^- and S^+ are disjoint subsets of vertices denoted as the **sources** or **sinks**, respectively. We again call the union of sources and sinks, $S^+ \cup S^-$, the set of **terminals** while the nodes in $V \setminus (S^+ \cup S^-)$ are called **intermediate nodes**.
We associate with each arc an integral capacity $u(a)$, or u_a, that is $u\colon A \to \mathbb{Z}_{\geq 0}$. The only thing in which a dynamic network differs from a static network is that in the dynamic setting each arc also has an integral **transit time** given by the integral function $\tau\colon A \to \mathbb{Z}_{\geq 0}$. We write $\tau(a)$ or τ_a for the transit time of an arc a. The transit time of an arc a captures the dynamic effect of the passing of time while flow travels along an arc. Flow that enters a a time $\theta \geq 0$ leaves the arc at time $\theta + \tau_a$. The dynamic aspect of the networks is additionally covered by the given capacities on the arcs, because – in contrast to static networks – the capacity in the dynamic setting is *not* a bound on the overall amount of flow that is allowed on an arc, but it is an upper bound on the *rate* at which flow can travel into an arc. If only a single source s or a single sink t is given, we write $\mathcal{N} = (D, u, \tau, s, t)$ instead of $\mathcal{N} = (D, u, \tau, \{s\}, \{t\})$.
Thus, a dynamic network $\mathcal{N}$ can be imagined as a system of pipes, where the transit time of an arc is the analogue to the length of a pipe, while the capacity of an arc can be imagined as the width of a pipe. A flow over time (in the continuous setting) is then similar to a liquid flowing from the sources in S^+ to the sinks in S^- through the pipe system given by a dynamic network $\mathcal{N}$. All definitions needed in the context of flows over time are given in the following paragraphs.
Clearly, we can also consider static flows in a given dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ by just ignoring the transit times on the arcs. However, usually when we compute static flows in a dynamic network the transit times are considered as *costs* on the arcs. Let x be a static flow in $\mathcal{N}$, then we can again consider the corresponding **residual network** $\mathcal{N}_x := (D_f, u_f, \tau, S^+, S^-)$.

The dynamic network $\mathcal{N}_x$ is defined completely similar to the static case, with the sole difference that in this case in $\mathcal{N}_x$ the transit times are extended to $\overleftrightarrow{A}$ by setting $\tau(\overleftarrow{a}) := -\tau_a$ for all $a \in A$.

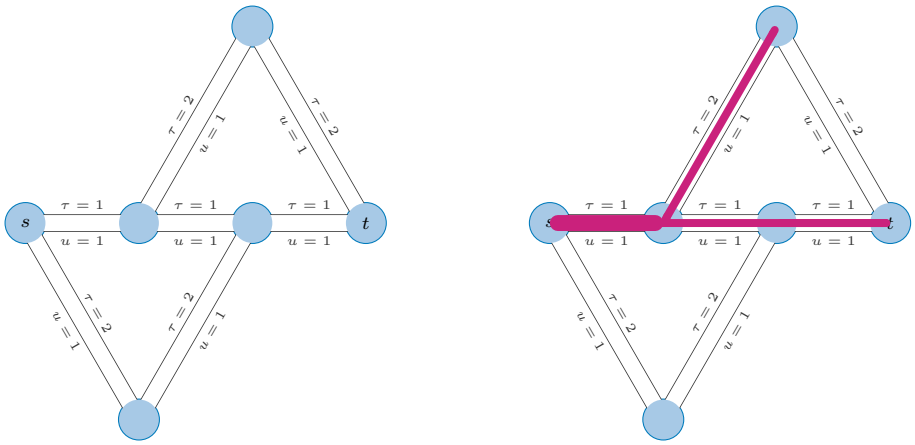
Discrete and Continuous Flows Over Time. Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network. A flow over time in the dynamic network is a function that assigns a flow value to each arc in A **for each point in time**. In the literature two settings of flows over time are considered. When Ford and Fulkerson introduced flows over time in the 1950s [FF58; FF62] they considered a **discrete** setting. In this setting only discrete points in time are considered, i.e., $\{1, 2, \dots, T\}$ are the first T points in time. In the continuous setting time is continuous and thus the whole real interval $[0, T)$ has to be considered until time T. Clearly, the intervals $[0, 1)$, $[1, 2)$, $\dots$, $[T-1, T)$ in the continuous setting can be identified with the first T points in time in the discrete setting. Overall, we obtain the following two definitions of a **flow over time** with **time horizon** T.
A **discrete flow over time** in $\mathcal{N}$ is a function

$$f\colon A \times \{1, 2, 3, \dots, T\} \to \mathbb{R}_{\geq 0},$$

while a **continuous flow over time** in $\mathcal{N}$ is a *Lebesque integrable* function

$$f\colon A \times [0, T) \to \mathbb{R}_{\geq 0}.$$

From now on we will only consider the *continuous setting* and we assume that T is rational if not stated otherwise. See Figure 2.5 for an illustration of a dynamic network and a flow over time. In



(a) A dynamic network. The transit times and capacities are indicated.

(b) A dynamic flow at time $\theta = 3$

Figure 2.5: A dynamic network and visualization of a flow over time that is obtained by sending flow at rate 1/2 into the upper path and the horizontal path for three time units visualized at time $\theta = 3$. In all the flow pictures throughout this thesis, we assume that the arcs are directed from left to right.

contrast to static flows, a flow value $f(a, \theta)$ of a flow over time in a dynamic network $\mathcal{N}$ does not give the overall amount of flow on arc a at time $\theta > 0$ but the **rate** at which flow travels into the arc a at time θ. Note that no flow is allowed in the network before time 0 *and* after time T. We also say that a flow over time f in a dynamic network $\mathcal{N}$ has **time horizon** T if no flow remains in the network after time T, i.e., it has to hold $f(a, \theta) = 0$ for all $a \in A$ and $\theta \geq T - \tau_a$.
A flow over time f in $\mathcal{N}$ is **feasible** if it respects the capacities, i.e., $f(a, \theta) \leq u_a$ for all $a \in A$ at all times $\theta \in [0, T)$, and **flow conservation**. Similar to flow conservation in the case of static flows, flow conservation in the dynamic setting means that flow that travels out of an intermediate node at some point in time θ has to have traveled into the node at some point before time θ. Formally, we define the **inflow** $\text{in}_f(v, \theta)$ of a continuous flow over time f into a node $v \in V$ at time

$\theta \geq 0$ to be the overall amount of flow that has traveled into node v until time θ in the flow over time f,

$$\text{in}_f(v, \theta) \coloneqq \sum_{a \in \delta^-(v)} \int_0^{\theta - \tau_a} f(a, \xi) d\xi.$$

Similarly, the **outflow** $\text{out}_f(v, \theta)$ from a node $v \in V$ at time $\theta \geq 0$ is defined by,

$$\text{out}_f(v, \theta) \coloneqq \sum_{a \in \delta^+(v)} \int_0^{\theta} f(a, \xi) d\xi.$$

For $v \in V$ the **excess** at a node $v \in V$ and time $\theta \geq 0$ is the net amount of flow that has entered the node v up to time θ, i.e.,

$$\text{ex}_f(v, \theta) \coloneqq \text{in}_f(v, \theta) - \text{out}_f(v, \theta).$$

Similarly, we define for all $v \in V$ the net amount of flow that has *left* the node v up to time θ by,

$$\text{net}_f(v, \theta) \coloneqq -\text{ex}_f(v, \theta).$$

The flow over time f fulfills **weak flow conservation** if $\text{ex}_f(v, \theta) \geq 0$ for each $v \in V \setminus (S^+ \cup S^-)$ and all $\theta \in [0, T)$ and $\text{ex}_f(v, T) = 0$ for all intermediate nodes v. The flow satisfies **strict flow conservation** if the stronger requirement

$$\text{ex}_f(v, \theta) = 0$$

holds for all intermediate nodes $v \in V \setminus (S^+ \cup S^-)$ and all $\theta \geq 0$. Intuitively, if weak flow conservation is allowed it means that we are allowed to store flow for some time at an intermediate node. The **value** of a flow over time f at time θ is the overall amount of flow that has reached the sinks until time θ and is defined as,

$$|f|_\theta \coloneqq \sum_{t \in S^-} \text{ex}_f(t, \theta).$$

The **total value** of f is the value of f at time T.

Throughout this thesis, whenever we speak of a **flow over time** f in a network $\mathcal{N}$, we mean that f is a continuous feasible flow over time satisfying *weak* flow conservation if not stated otherwise.

Path Flows Over Time. The above definition of a flow over time is an arc based definition: A flow over time f in a given dynamic network $\mathcal{N}$ is a function specifying a flow value for each arc for each point in time $\theta \in [0, T)$ or $\theta \in \{0, 1, 2, ..., T\}$. Similar to static flows we can, however, again consider a path based definition. Given a dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^-, S^+)$ recall that the set of all s-t paths for every pair of $s \in S^+$ and $t \in S^-$ is denoted by $\mathcal{P}$.
A **path flow over time** f assigns a flow value to each path in $\mathcal{P}$ for each point in time $\theta \geq 0$. The flow value at time $\theta \geq 0$ at some path $P \in \mathcal{P}$ specifies the inflow rate into this path at time θ. Additionally, in a path flow over time flow travels along a path without waiting in between. To ensure that the resulting flow is feasible, we have to make sure that not too much flow enters an arc at the same time.
Before we get to the formal definition of path flows over time, we need some preliminary definitions. For a path $P \in \mathcal{P}$ in a dynamic network $\mathcal{N}$ we define the **length** of P to be the overall transit time $\tau(P)$ of this path,

$$\tau(P) \coloneqq \sum_{a \in P} \tau_a.$$

If P is a path from $s \in S^+$ to $t \in S^-$ and $v \in P$ is a node on that path, we denote the corresponding sub-path from s to v by $P_{[s,v]}$ and the sub-path from v to t by $P_{[v,t]}$. For an arc $a \in A$ and a given point in time $\theta \geq 0$, we denote the set of paths using arc $a = (u, v)$ at time θ by

$$\mathcal{P}_a^\theta := \{P \in \mathcal{P} \mid a \in P \text{ and } \tau(P_{[s,u]}) \leq \theta \text{ and } \tau(P_{[u,t]}) < T - \theta\}.$$

A **path flow over time** with time horizon T is a function $x \colon \mathcal{P} \times [0, T) \to \mathbb{R}_{\geq 0}$. Again, the flow over time x has to satisfy the capacity constraint, i.e.,

$$\sum_{P \in \mathcal{P}_a^\theta} x(P, \theta - \tau(P_{[s,u]})) \leq u_a$$

for all arcs $a = (u, v)$ and times $\theta \in [0, T)$, and no flow arrives too late, i.e. $x(P, \theta) = 0$ for all $\theta \in [T - \tau(P), T)$. The **total value** of a path flow over time is given by

$$|x|_T := \sum_{P \in \mathcal{P}} \int_0^T x(P, \xi) d\xi.$$

Note that sending flow along a path automatically enforces strict flow conservation. Hence, we do not have to worry about flow conservation here. However, path flows thus cannot be used to model situations in which waiting at intermediate nodes is required without changing the underlying network structure. It is clear that each path based flow over time x induces an arc based flow over time f of the same value. On the other hand it is not immediate, how an equivalent path based flow over time can be constructed from an arc based flow over time.

The Encoding Size Of Flows Over Time. One downside of flows over time is that the dependence on time yields a super-polynomial natural encoding size. For an arc based flow over time f in a network $\mathcal{N}$ with time horizon T we have to store one flow value per arc for *each point in time*, i.e. for each $\theta \in [0, T)$. The same clearly holds for a path based flow over time x. That there are in the worst case exponentially many paths from S^+ to S^- in a dynamic network $\mathcal{N}$ is an additional problem for a path based flow over time. The super-polynomial encoding size of a flow over time seems unavoidable. However, nicer descriptions of flows over time are possible. In the next paragraph we describe an especially useful class of flows over time.

(Generalized) Temporally Repeated Flows. Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and x a *static* flow with flow composition $(x_P)_{P \in \mathcal{P} \cup \mathcal{C}}$ according to Theorem 2.8. The corresponding **temporally repeated flow** f with time horizon T is obtained by sending flow at a constant rate into the (polynomially many) flow carrying paths from the flow decomposition of x as long as possible, that is

$$f(a, \theta) := \sum_{P \in \mathcal{P}_a^\theta} x(P) \text{ for all } a \in A \text{ and } \theta \in [0, T).$$

Unfortunately, not many flow over time problem can be solved by temporally repeated flows as the restriction that we have to start sending flow into a path at time zero is too limiting. However, all flow over time problems considered in this thesis can be solved by **generalized temporally repeated flows** that we define in the following. Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and x a static flow in the underlying static network. Recall that by $\overleftrightarrow{\mathcal{P}}$, we denote the set of s-t paths in the bidirected graph $\overleftrightarrow{D}$ for all pairs of $s \in S^+$ and $t \in S^-$. The collection $(x_P)_{P \in \overleftrightarrow{\mathcal{P}}}$ is a **generalized path decomposition** of x if $x_P \geq 0$ for each $P \in \overleftrightarrow{\mathcal{P}}$ and

$$x_a = \sum_{P \in \overleftrightarrow{\mathcal{P}} : a \in P} x_P - \sum_{P \in \overleftrightarrow{\mathcal{P}} : \overleftarrow{a} \in P} x_P \text{ for all } a \in A.$$

The **generalized temporally repeated flow** f corresponding to such a generalized path decomposition is defined as

$$f(a,\theta) \coloneqq \sum_{P \in \overleftrightarrow{\mathcal{P}}_a^\theta} x(P) - \sum_{P \in \overleftrightarrow{P}_{\overleftarrow{a}}^\theta} x(P), \text{ for all } a \in A \text{ and } \theta \in [0,T).$$

Thus, again we obtain f by sending flow at a constant rate x_P along all flow carrying paths in the generalized path decomposition x_P. Note that the paths from the generalized path decomposition are paths in $\mathcal{N}_x$ and hence also contain backwards arcs with negative transit time.
A temporally repeated flow f corresponding to a static flow x is a feasible flow over time by definition as flow conservation is clearly fulfilled and the flow value on an arc a is bounded by the static flow value $x(a)$ at each point in time. For a generalized temporally repeated flow the same observation is not necessarily true. It is possible that the flow value of an arc might be negative at some point in time. This happens whenever flow along a backwards arc is sent when there is no flow on the corresponding forward arc.
Temporally repeated flows were already introduced by Ford and Fulkerson [FF58] while generalized temporally repeated flow were first considered by Hoppe and Tardos [HT00].

2.5.2 The Time-Expanded Network

One straightforward way to solve flow over time problems is using the so-called **time-expanded network**. Time expansion is a concept to transform a dynamic problem with transit times and a time horizon into a static problem without transit times and a time horizon that can be solved by methods of static network flow theory. The downside of such a reduction is however that it usually causes an exponential blow-up of the network size such that the resulting algorithms for flow over time problems using time expansion are usually not of polynomial running time. The concept of time expansion was introduced by Ford and Fulkerson in the same paper in which they introduced the concept of flows over time [FF58].
The general idea behind time expansion is to remove the dependency on time from a given dynamic network by introducing one copy of the underlying static network for each point in time. Thus, we create one copy of each node in the dynamic network for each point in time, starting from time $\theta = 1$ on. These nodes are then connected by arcs according to the transit times of the arcs from the original dynamic network. The resulting network is the **time-expanded network**. When we want to allow storage of flow at intermediate nodes, **holdover arcs** are added to the network. The holdover arcs connect two copies of the same node in adjacent time layers. Thus, flow traveling through a holdover arc can be interpreted as flow waiting at a node for one time step.

Definition of the Time-Expanded Network. Let $\mathcal{N} = (D = (V,A), u, \tau, u, S^+, S^-)$ be a dynamic network and $T \geq 0$ an integral time horizon. For the construction of the time-expanded network we additionally assume that we are given an integral **supply/demand-function** on the terminals, i.e., $b\colon S^+ \cup S^- \to \mathbb{Z}$ with $b(s) > 0$ for all $s \in S^+$, $b(t) < 0$ for all $t \in S^-$ and $b(S^+ \cup S^-) = \sum_{s \in S^+ \cup S^-} b(s) = 0$. The corresponding **time-expanded network** is denoted by

$$\mathcal{N}^T \coloneqq (D^T = (V^T, A^T), u^T, s^*, t^*)$$

while the **time-expanded network with holdover arcs** is denoted by

$$\mathcal{N}_{\mathrm{h}}^T \coloneqq (D^T = (V^T, A_{\mathrm{h}}^T), u^T, s^*, t^*).$$

We will introduce all components of both versions of the time-expanded network in the following paragraph. The set of nodes V^T consists of T copies of the nodes from the original dynamic network $\mathcal{N}^T$, the original terminals $S^+ \cup S^-$, a super-source s^* and a super-sink t^*, that is,

$$V^T \coloneqq \{v^\theta \mid v \in V, \theta \in \{1,2,\dots,T\}\} \cup S^+ \cup S^- \cup \{s^*, t^*\}.$$

The super-source s^* forms the sole source of the time-expanded network while t^* is its only sink. For each arc $a \in A$ from the dynamic network we create $T - \tau_a$ copies connecting a node copy on layer θ with another node copy on layer $\theta + \tau_a$ if the arrival time lies withing the given time horizon. I.e., an arc $a = (u, v)$ with transit time τ_a from the dynamic network results in $T - \tau_a$ arcs in the time-expanded network originating from the nodes u^1, u^2, ..., $u^{T-\tau_a}$. Overall we define

$$A' \coloneqq \{a^\theta = (u^\theta, v^{\theta+\tau_a}) \mid a = (u, v) \in A \text{ and } \theta \in \{1, 2, \dots, T - \tau_a\}\}.$$

The original terminals $S^+ \cup S^-$ of $\mathcal{N}$ are connected to their copies in each layer and the super-source s^* and super-sink t^* are linked with the nodes in S^+ and S^-, respectively. The arcs connected to S^+ and s^* are defined as follows,

$$A^+ \coloneqq \{(s^*, s) \mid s \in S^+\} \cup \{(s, s^\theta) \mid s \in S^+ \text{ and } \theta \in \{1, 2, \dots, T\}\}.$$

The arcs connected to S^- and t^* are analogously defined by

$$A^- \coloneqq \{(t, t^*) \mid t \in S^-\} \cup \{(t^\theta, t) \mid t \in S^- \text{ and } \theta \in \{1, 2, \dots, T\}\}.$$

The whole set of arcs A^T is then defined by

$$A^T \coloneqq A' \cup A^+ \cup A^-.$$

If we want to allow storage at intermediate nodes, we additionally add holdover arcs between copies of intermediate nodes to the network,

$$H \coloneqq \{(v^\theta, v^{\theta+1} \mid v \in V \setminus S^+ \cup S^- \text{ and } \theta \in \{1, 2, \dots, T - 1\})\}.$$

In this case we define $A^T_{\text{h}} \coloneqq A' \cup A^+ \cup A^- \cup H$. It remains to define the capacity function u^T on the arcs in A^T. All arcs a^θ corresponding to an arc $a \in A$ from the dynamic network $\mathcal{N}$ get the capacity u_a of the arc a and the holdover arcs and the arcs connecting S^+ and S^- with their respective copies get infinite capacity. The capacities of the arcs connecting s^* and t^* to S^+ and S^- depend on whether we were given supplies/demands b for the terminals in the dynamic network, or not. If we are not given a supply/demand function, we define for all $a' \in A^T$

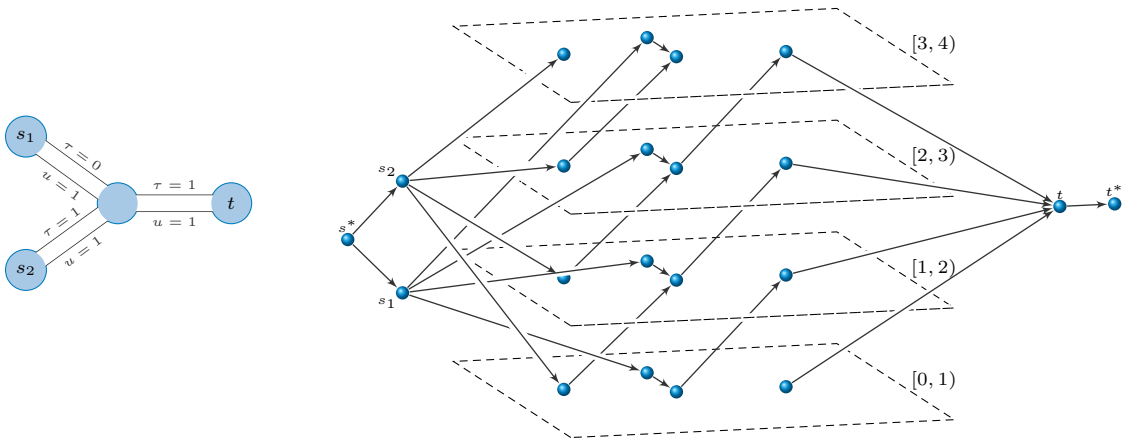
$$u^T(a') \coloneqq u^T_{a'} \coloneqq \begin{cases} u_a & \text{if } a' \in A' \text{ with } a' = a^\theta \text{ for } a \in A \text{ and } \theta \in \{1, 2, \dots, T - \tau_a\}, \\ \infty & \text{else .} \end{cases}$$

If we are given a supply/demand function b, the arc linking s^* to a source $s \in S^+$ gets capacity $b(s)$, while an arc linking a sink $t \in S^-$ to t^* gets capacity $-b(t)$, i.e.,

$$u^T_{a'} \coloneqq \begin{cases} u_a & \text{if } a' \in A' \text{ with } a' = a^\theta \text{ for } a \in A \text{ and } \theta \in \{1, 2, \dots, T - \tau_a\}, \\ b_s & \text{if } a' = (s^*, s) \text{ with } s \in S^+, \\ -b_t & \text{if } a' = (t, t^*) \text{ with } t \in S^-, \\ \infty & \text{else.} \end{cases}$$

See Figure 2.6 for an illustration of the time-expanded network.

The Time-Expanded Network for Rational Time Horizons. The construction of the time-expanded network we described above only works if the given time horizon is integral. We will now briefly describe how we can construct a time-expanded network for rational time horizons. For this purpose assume that $\mathcal{N}$ is a dynamic network and $T = p/q$ with $p, q \in \mathbb{Z}$ such that $p \geq 0$ and $q \geq 1$ is a rational time horizon. The construction of the time-expanded network $\mathcal{N}^T$ (or $\mathcal{N}^T_h$) now works mostly similar as above, but with two differences. First, we now construct a time layer for each $\theta \in \{1/q, 2/q, \dots, p/q\}$. The second difference are the capacities of the arcs connecting the time layers. Let $a \in A$ be an arc in $\mathcal{N}$. In the rational version of the time-expanded network this arc get



(a) A dynamic network $\mathcal{N}$

(b) The time-expanded network $\mathcal{N}^4$

Figure 2.6: A dynamic network $\mathcal{N}$ with the corresponding time-expanded network $\mathcal{N}^4$

$q \cdot (T - \tau_a)$ copies. One copy, for each $\theta \in \{1/q, 2/q, \dots T - \tau_a\}$. Instead of giving each copy a^θ of this arc capacity u_a in $\mathcal{N}^T$ or $(\mathcal{N}_h^T)$, we define

$$u^T(a^\theta) \coloneqq \frac{1}{q} \cdot u_a.$$

All other arc capacities are defined similar to the integral case.

Flows Over Time and Static Flows in the Time-Expanded Network. Our next goal is to connect flows over time in a given dynamic network $\mathcal{N}$ with static flows in the time-expanded network $\mathcal{N}^T$ for a given rational time horizon $T = p/q$. Essentially, we can find an equivalent flow over time without storage in $\mathcal{N}$ for a given static flow in $\mathcal{N}^T$ and vice versa. Analogously, we can find an equivalent flow over time with storage for a static flow in $\mathcal{N}_{\mathrm{h}}^T$ and vice versa. Ford and Fulkerson [FF58] already showed that given a static flow x^T in $\mathcal{N}^T$ (or $\mathcal{N}_{\mathrm{h}}^T$) we can find a flow over time f without storage (or with storage) in $\mathcal{N}$ of the same value, i.e., $|x^T| = |f|_T$, and vice versa. We need slightly stronger result.

Lemma 2.20. *Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and $T = p/q$ with $p, q \in \mathbb{Z}$, $p \geq 0$ and $q \geq 1$ a rational time horizon. Each feasible static flow x^T in $\mathcal{N}^T$ (or $\mathcal{N}_h^T$) yields a flow over time f in $\mathcal{N}$ without storage (or with storage) with time horizon T such that*

$$\mathrm{net}_{x^T}(\{u^{1/q}, u^{1/q}, \dots, u^\theta\}) = \mathrm{net}_f(u, \theta) \text{ for all } u \in S^+ \cup S^- \text{ and } \theta \in \{1/q, 2/q, \dots, T\}, \tag{2.6}$$

i.e., in particular

$$|f|_T = |x^T|,$$

and vice versa. The same statement also holds if supplies and demands are given.

We do not give the proof of this lemma but refer to [Kap14]. We will, however, how to obtain the corresponding flows.

Given a flow over time f with time horizon T in a dynamic network $\mathcal{N}$, the static flow x^T in $\mathcal{N}^T$ (or $\mathcal{N}_{\mathrm{h}}$) is defined as follows: We define the flow value on a copy a^θ of an arc $a \in A$ to be

the the overall amount of flow that enters arc a in the flow over time f during $[\theta - 1/q, \theta)$ for all $\theta \in \{1/q, 2/q, \dots, \theta - \tau_a\}$,

$$x^T(a^\theta) \coloneqq \int_{\theta-1/q}^{\theta} f(a,\xi)d\xi, \text{ for all } a \in A \text{ and } \theta \in \{1/q, 2/q, \dots, T - \tau_a\}.$$

For any source $s \in S^+$ and every $\theta \in \{1/q, 2, /q \dots, \theta/q\}$ we define

$$x^T((s, s^\theta)) \coloneqq \text{out}_f(s,\theta) - \text{out}_f(s, \theta - 1/q),$$

Similarly, we define for each $t \in S^-$ and $\theta \in \{1/q, 2/q, \dots, \theta\}$

$$x^T((t^\theta, t)) \coloneqq \text{in}_f(t,\theta) - \text{in}_f(t, \theta - 1/q).$$

Accordingly, we define $x^T((s^*, s))$ to be the amount of flow that leaves the source $s \in S^+$ in f,

$$x^T((s^*, s)) \coloneqq \text{out}_f(s, T) \text{ for all } s \in S^+,$$

and

$$x^T((t, t^*)) \coloneqq \text{in}_f(t, T) \text{ for all } t \in S^-.$$

If holdover arcs are present, we define

$$x^T((v^\theta, v^{\theta+1/q})) \coloneqq \text{ex}_f(v,\theta) \text{ for } v \in V \setminus (S^+ \cup S^-) \text{ and } \theta \in \{1/q, 2, /q \dots, T - 1/q\}.$$

On the other hand, let x^T be a static flow in the time-expanded network $\mathcal{N}^T$ (or $\mathcal{N}_h^T$). Denote by r the function that rounds down $\theta \in [0, T)$ to the element in $\{1/q, 2/q, \dots, T\}$ that is nearest to θ. We define the flow over time f in $\mathcal{N}$ as follows,

$$f(a,\theta) \coloneqq x^T(a^{r(\theta)+1/q}) \cdot q,$$

for all $a \in A$ and $\theta \in [0, T)$.
A lot of flow over time problems can be solved by using methods from static flow theory in the time-expanded network. However, algorithms for flow over time problems obtained in this way are usually not of polynomial running time, because the size of the time-expanded network depends linearly on T and T is generally not polynomially bounded.

2.5.3 The Maximum Flow Over Time Problem

We will illustrate the techniques defined in the previous section on the **Maximum Flow over Time Problem**, the most basic extension of a static network flow problem to the temporal setting. The maximum flow over time problem was already introduced by Ford and Fulkerson in the 1950s. Given a dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ and a time horizon $T \geq 0$, the goal of the maximum flow over time problem is to compute a flow over time f in $\mathcal{N}$ from S^+ to S^- with time horizon T with maximal value.

Maximum Flow over Time Problem

Instance:	A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ and a time horizon T
Task:	A flow over time f with time horizon T maximizing the total flow value $\lvert f \rvert_T$

By attaching a super-source and a super-sink we can again reduce the general maximum flow over time problem to the maximum flow over time problem in networks with only a single source and a single sink. The new arcs added to create the extended network all get zero transit time.

Solving the Maximum Flow Over Time Problem in the Time-Expanded Network. It is easy to see that a maximum flow over time can be computed in the time-expanded network using Lemma 2.20. The lemma states that any flow in the time-expanded network yields a flow over time of the same value. Thus, a maximum flow over time corresponds to a maximum flow in the time-expanded network. Hence, we obtain the following algorithm for computing a maximum flow over time using the time-expanded network. The size of the time-expanded network is not necessarily bounded in the

Algorithm 4: Algorithm for the maximum flow over time problem using the time-expanded network

```
Input  : A dynamic network N = (D = (V, A), u, τ, S⁻, S⁻) and a (rational) time horizon T
Output : A maximum flow over time f with time horizon T
1 N^T ← time-expanded network corresponding to N and T
2 x^T ← maximum s*-t* flow in N^T
3 f ← flow over time with time horizon T corresponding to f^T
4 return f
```

original input size because T is not bounded. Thus, the algorithm above only has pseudo-polynomial running time.

The Algorithm of Ford and Fulkerson. Because of the natural exponential encoding size and the dependence on T in a maximum flow over time problem, it might be surprising that this problem can be solved fairly simple in strongly polynomial time using temporally repeated flows. It is a result by Ford and Fulkerson [FF58] that a maximum flow over time problem can always be solved by a temporally repeated flow and hence in order to solve such a problem it suffices to compute a temporally repeated flow with time horizon T of maximal value. Note that a temporally repeated flow with time horizon T can only send flow along paths of length at most T. For such a temporally repeated flow, we make the following observation: Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and $T \geq 0$ a time horizon. Let x be a static flow from S^+ to S^- in the static network underlying $\mathcal{N}$ with flow decomposition $(x_P)_{P \in \mathcal{P} \cup \mathcal{C}}$ such that $x_P = 0$ for all $P \in \mathcal{P}$ with $\tau(P) < T$. We get for the value of the corresponding temporally repeated flow f

$$\begin{aligned}
|f| &= \sum_{P \in \mathcal{P}} (T - \tau(P)) \cdot x_P \\
&= T \cdot \sum_{P \in \mathcal{P}} x_p - \sum_{P \in \mathcal{P}} \sum_{a \in P} \tau_a \cdot x_P \\
&= T \cdot |x| - \sum_{a \in A} \tau_a \cdot \sum_{P \in \mathcal{P} : a \in P} x_P \\
&= T \cdot |x| - \sum_{a \in A} \tau_a \cdot x_a.
\end{aligned}$$

In particular, note that the value of such a temporally repeated flow does not depend on the specific path decomposition.

In fact, the main idea of the algorithm of Ford and Fulkerson for the maximum flow over time problem is to compute a temporally repeated flow f with time horizon T corresponding to a static flow x that maximizes $T \cdot |x| - \sum_{a \in A} \tau_a \cdot x_a$. It turns out that a static flow x maximizing this value can be computed by just one minimum-cost flow computation in a slightly extended network $\tilde{\mathcal{N}}$. The extended network $\tilde{\mathcal{N}}$ is obtained by attaching a super-source s^* and a super-sink t^* to $\mathcal{N}$ by arcs with infinite capacity and zero transit time. Additionally, we add an arc (t^*, s^*) with infinite capacity and transit time $-T$. Let x be a minimum-cost circulation in $\tilde{\mathcal{N}}$ where the transit times are regarded as the cost function. The flow value on the arc (t^*, s^*) equals the

Algorithm 5: Ford-Fulkerson algorithm for the maximum flow over time problem, $\mathrm{FF}(\mathcal{N}, T)$

```
Input  : A dynamic network N = (D = (V, A), u, τ, S⁻, S⁻) and a time horizon T ∈ ℤ₊
Output : A maximum flow over time f with time horizon T given as a temporally repeated flow
1  Ṽ ← V ∪ {s*, t*}
2  Ã ← A ∪ {(s*, s) | s ∈ S⁺} ∪ {(t, t*) | t ∈ S⁻} ∪ {(t*, s*)}
3  define τ((s, s*)) := 0 for all s ∈ S⁺, τ((t, t*)) = 0 for all t ∈ S⁻ and τ((t*, s*)) = −T
4  define u((s, s*)) := ∞ for all s ∈ S⁺, u((t, t*)) = ∞ for all t ∈ S⁻ and u((t*, s*)) = ∞
5  Ñ ← (D = (Ṽ, Ã), u, τ, s*, s⁻)
6  x' ← static minimum-cost circulation in Ñ with τ as costs
7  x ← static s*-t* flow corresponding to x
8  (x_P)_{P∈P∪C} ← flow decomposition of x
9  f ← temporally repeated flow corresponding to (x_P)_{P∈P∪C}
10 return f
```

overall flow value of the corresponding s-t flow in $\mathcal{N}$, that is the overall cost of the static flow x is

$$\sum_{a \in A} \tau_a x_a - T \cdot |x|.$$

Thus, since x is a minimum cost circulation, it maximizes $T|x| - \sum_{a \in A} \tau_a x_a$ as required. To deduce that a corresponding temporally repeated flow also maximizes this value, we need to argue that a path decomposition of x cannot contain a path of length at least T. But this is straightforward as such a path in the flow decomposition would induce a negative cycle in the residual graph and hence would contradict the fact that x is a minimum cost circulation.
Our arguing above directly implies that Algorithm 5 returns a temporally repeated flow with time horizon T of maximal value. It is not directly clear that the returned flow over time f is also a flow over time with an overall maximal value. However, this was shown by Ford and Fulkerson [FF58] in 1958 by using a generalization of the max flow - min cut theorem for flows over time. We can thus conclude with the following fact:

Fact 2.21. *Let $\mathcal{N}$ a dynamic network and $T \geq 0$ a time horizon. The corresponding maximum flow over time can be solved by one minimum-cost flow computation in the underlying static network.*

Flow Over Time Problems

3

In this chapter we give a thorough overview over the state of the art regarding the two classical flow over time problems that we mainly focus on throughout this thesis: The **Quickest Transshipment Problem** and the **Earliest Arrival Transshipment Problem**. For both problems we precisely describe the currently best known algorithms for exactly solving these problems and in case of the earliest arrival transshipment problem we also shortly review the state of the art regarding approximation algorithms. Moreover, we give a short overview over our main contributions in the context of both problems.

Contents

3.1 The Quickest Transshipment Problem

In the previous section we introduced flows over time as an extension of classical (static) network flows that also incorporates a time component. In many applications connected with network flows time plays a vital role, for example in routing problems or problems from logistics. Flows over time are much better suited to model real life problems from such areas than static network flows, which simply ignore the time dependence of the problems they are used to model. One application of flows over time, where their advantage over classical network flows becomes most apparent, are evacuation scenarios. Taking time into account is obviously essential when trying to rescue people from life threatening situations. Imagine, for example, a cruise ship on the ocean that gets into distress such that all the passengers have to be evacuated into the lifeboats. Of course, the goal in such a situation is to lead all passengers to the safety of the lifeboats as quickly as possible. In order to create a suitable evacuation strategy, one could model the cruise ship as a dynamic network in which the lifeboats are located at the sinks and the cabins of the passengers at the sources of the network. Clearly, the number of people that fit into a lifeboat is bounded while also only a certain number of passengers can stay in a cabin. An evacuation strategy needs to respect these bounds, i.e., it doesn't make sense to try to evacuate more people from a cabin than are accommodated there, whereas it is even life-threatening to put more passengers into a lifeboat than it is safe to carry. This leads to the **quickest transshipment problem** in which we are given a dynamic network $\mathcal{N}$ and a supply/demand function on the terminals which gives each source a positive supply and each sink a negative demand. The goal is to find a flow over time that satisfies all supplies and demands as quickly as possible.

In contrast to the static transshipment problem with multiple sources and sinks the quickest transshipment problem with multiple sources and sinks *cannot* be reduced to the case with only a single source and a single sink. Adding super terminals is of no use in the dynamic setting as shifting the supplies and demands to them leaves us with no guarantee that the computed flow over time respects the original supplies and demands. Adding suitable capacities to the arcs connected to the super-terminals also does not help as these capacities only bound the rate at which flow might

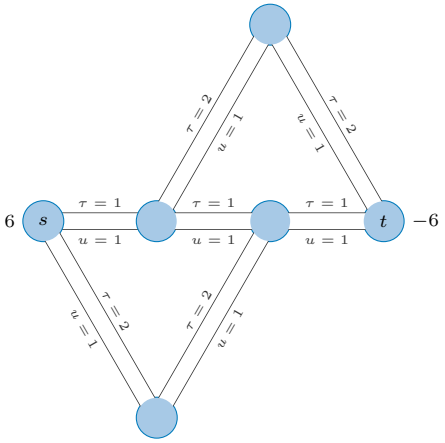
travel into them.
Thus, it makes sense to distinguish between the general quickest transshipment problem in networks with multiple sources and sinks (we will also separately consider networks with only one source *or* one sink) and the quickest transshipment problem in networks with a single source and a single sink, which is also called the **quickest flow problem**.

The Quickest Flow Problem. As already stated above, in the quickest flow problem we are given a dynamic network $\mathcal{N} = (D = (V, A), u, \tau, s, t)$ with a single source s and a single sink t with a supply $b(s)$ and a demand $b(t)$ such that $b(s) = -b(t)$. The goal is to find the minimal time horizon T such that a flow over time that fulfills the supply and demand exists and to compute the corresponding flow. In other words, we want to find the minimal time horizon T such that the value of a maximum flow over time with time horizon T in $\mathcal{N}$ is $b(s)$. One application of such a flow are evacuation scenarios in which people need to be brought from one isolated endangered area at the source to a safe location at the single sink [Ber78; CFS82; CHT88; HT02]. An example for a quickest flow problem can be seen in Figure 3.1.

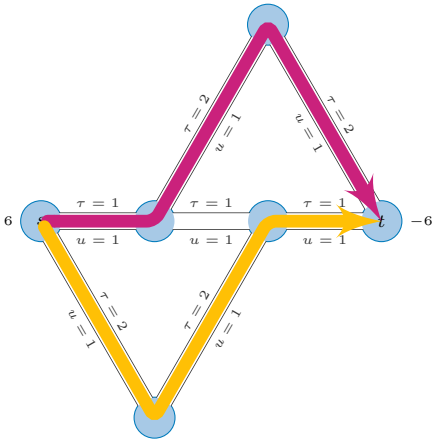
THE QUICKEST FLOW PROBLEM, $(\mathcal{N}, s)$

Instance: A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, s, t)$ with only a single source s and a single sink t together with a supply $b(s)$ and a demand $b(t)$ such that $b(s) = -b(t)$

Task: Compute a flow over time f with minimal time horizon T that fulfills the supply and demand.



(a) A dynamic network with a single source and a single sinks, supply 6 and demand −6



(b) A quickest flow with respect to the given dynamic network is obtained by sending flow at rate 1 into both of the indicated paths during $[0, 3)$. This results in a minimal time horizon of 8.

Figure 3.1: An example for a quickest flow problem

An algorithm with pseudo-polynomial running time for the quickest flow problem can clearly be achieved by doing (static) maximum flow computations in the time-expanded network paired with binary search to determine the minimal time horizon. For this purpose denote by $d(\mathcal{N}, s, t)$ the length of a shortest path from s to t with respect to τ in a dynamic network $\mathcal{N}$ and let $P \in \mathcal{P}$ be a corresponding shortest s-t path. The maximal rate at which we can send flow into P is $\min_{a \in P} u_a$ and we can send flow into P for at most $b(s)/\min_{a \in} u_a$ time units until the supply is exhausted. This implies that $d(\mathcal{N}, s, t) + b(s)/\min_{a \in} u_a$ gives an upper bound on the minimal time necessary to be able to fulfill the supply and demand.

Observation 3.1. *Let $(\mathcal{N}, b)$ with $\mathcal{N} = (D = (V, A), u, \tau, s, t)$ be an instance of the quickest flow problem. The minimal time horizon necessary to be able to fulfill all supplies and demands is at most $d(\mathcal{N}, s, t) + b(s)/\min_{a\in} u_a$.*

This upper bound can be incorporated into the binary search framework. A weakly polynomial time algorithm can be achieved by coupling the maximum flow over time algorithm by Ford and Fulkerson (see Algorithm 5) with binary search using the upper bound on the minimal time horizon from Observation 3.1. For a strongly polynomial time algorithm the algorithm of Ford and Fulkerson can be paired with the parametric search framework of Megiddo (see Section 2.2). Applying the parametric search framework without any adaption leads to an algorithm for the quickest flow problem with running time $\mathcal{O}(\mathcal{C}_T(\mathrm{FF}(\mathcal{N}, T)) \cdot \mathrm{FF}(\mathcal{N}, T))$. Here, $\mathcal{C}_T(\mathrm{FF}(\mathcal{N}, T))$ is the number of comparisons depending on the parameter T needed when applying the algorithm of Ford and Fulkerson to a dynamic network $\mathcal{N}$ with time horizon T. We denote this algorithm by by $\mathrm{FF}(\mathcal{N}, T)$ (see also Section 2.5.3). Burkhard, Dlaska and Klinz [BDK93] specifically tailor the parametric search framework of Medgiddo [Meg79; Meg83] to the quickest flow problem and achieve an algorithm with a strongly polynomial running time in $\mathcal{O}(m^2(\log n)^3(m+n\log n))$. Their algorithm needs to repeatedly call subroutines that solve static minimum-cost flow problems. In 2015, Lin and Jaillet [LJ15] devised a polynomial time algorithm for the quickest flow problem whose worst case running time is the same as the one of the cost scaling algorithm of Goldberg and Tarjan [GT87; GT90] for solving static minimum-cost flow problems. Their result showed that quickest flow problems can be solved in the same time bound as one of the fastest algorithm for the minimum-cost flow problem. The algorithm of Lin and Jaillet gives a weakly polynomial running time in $\mathcal{O}(nm\log(n^2/m)\log(nC))$ where C is the maximum transit time of an arc. Recently, Saho and Shigeno [SS17a] achieved an algorithm with strongly polynomial running time in $\mathcal{O}(nm^2(\log n)^2)$ by replacing the scaling phases in the algorithm of Lin and Jaillet with procedures of the cancel-and-tighten algorithm for minimum-cost flows. The algorithm of Saho and Shigeno is currently the fastest known algorithm for the quickest flow problem with a strongly polynomial running time.

Fact 3.2. *An instance $(\mathcal{N}, b)$ of the quickest flow problem with $\mathcal{N} = (D = (V, A), u, \tau, s, t)$ can be solved in strongly polynomial running time $\mathcal{O}(nm^2(\log n)^2)$ using the algorithm of Saho and Shigeno [SS17a].*

When considering evacuation scenarios being limited to networks with only a single source and a single sink is clearly too restrictive. Usually, the evacuees are not located in a single area. For example, in order to devise an evacuation strategy for a huge building or a stadium filled with people, it clearly does not suffice to assume that all people are located on a single floor or in a certain area of the stadium. However, it might be sufficient to assume that there is only one safe area to which all endangered people are evacuated, i.e., the outside of the building. The version of the quickest transshipment problem with multiple sources with supplies but only a single sink is sometimes referred to as **evacuation problem**. There has been some research towards algorithms for special cases of the evacuation problem. Mamada et al. [Mam+06] gave an $\mathcal{O}(n\log^2 n)$ algorithm for tree networks. Hall et al. [HHS03] studied the case called *uniform path-lengths* where there exists a single sink t and for any vertex v the sum of transit times of arcs on any path from v to t takes the same value. Kamiyama et al. [KKT06] devised an $\mathcal{O}(n\log n)$ time algorithm for grid networks of size $\sqrt{n}\times\sqrt{n}$ with uniform path lengths. In a later paper Kamiyama [KKT09] generalized the class for which they achieve efficient algorithms to the class of dynamic networks with a single sink t with uniform path-lengths and with the property that for each vertex v the minimum v-t cut is determined by the arcs incident to t whose tails are reachable from v. However, the best known algorithm for the evacuation problem in general dynamic networks is still also the best possible algorithm for the general quickest transshipment problem for which we device a new algorithm throughout this thesis.

The Quickest Transshipment Problem. An instance $(\mathcal{N}, b)$ of the **quickest transshipment problem** consists of a dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ and a supply/demand function $b\colon S^+ \cup S^- \to \mathbb{Z}$ on the terminals such that $b(s) > 0$ for all $s \in S^+$ and $b(t) < 0$ for all $t \in S^-$ and $\sum_{s\in S^+\cup S^-} b(s) = 0$. The goal is to compute a flow over time f in $\mathcal{N}$ with minimal time horizon T that fulfills all supplies and demands.

The Quickest Transshipment Problem $(\mathcal{N}, b)$

Instance: A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ with a supply/demand function $b\colon S^+ \cup S^- \to \mathbb{Z}$ on the terminals

Task: Compute a flow over time f in $\mathcal{N}$ that fulfills all supplies and demands with minimal time horizon.

Of course, quickest transshipment problems can be solved using time expansion and a search framework in pseudo-polynomial running. However, coming up with (strongly) polynomial running time algorithms for this problem is much more complicated. In 1995 Hoppe and Tardos published the first algorithm for this problem with a strongly polynomial running time. Afterwards no improvements regarding algorithms for the general version of the quickest transshipment problem have been made for more than 20 years. The only results published in the meantime all considered special settings (see the results about the quickest and the evacuation problem that we mentioned above and [Fle01]). In Chapter 4 of this thesis we present the first improvement upon the algorithm of Hoppe and Tardos for the quickest transshipment problem.
Understanding the foundations of Hoppe and Tardos' algorithm is vital to understanding the contributions of this thesis. Thus, in the following we give an overview over the results that the algorithm of Hoppe and Tardos is based on.

The algorithm of Hoppe and Tardos works in two phases. In the first phase the **minimal feasible time horizon** of a given quickest transshipment problem $(\mathcal{N}, b)$ is determined and in the second phase the actual flow over time solving the quickest transshipment problem is computed. The minimal feasible time horizon is the minimal time needed to be able to fulfill all supplies and demands. In the second phase a so-called **transshipment over time problem** is solved:

The Transshipment over Time Problem $(\mathcal{N}, b, T)$

Instance: A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ with a supply/demand function $b\colon S^+ \cup S^- \to \mathbb{Z}$ on the terminals and a time horizon $T \geq 0$

Task: Compute a flow over time f in $\mathcal{N}$ with time horizon T that satisfies all supplies and demands if such a flow exists.

It is not straightforward how to solve such a **transshipment over time problem** $(\mathcal{N}, b, T)$ in strongly polynomial time. As we already mentioned before, introducing super terminals and then using the algorithm of Ford and Fulkerson for the maximum flow over time problem with time horizon T in the extended network does not help in this case. Hoppe and Tardos manage to reduce transshipment over time problems $(\mathcal{N}, b, T)$ to lexicographically maximum (lex-max) flow over time problems, for which they present a strongly polynomial time algorithm. Before we describe this reduction, which relies on $2|S^+ \cup S^-| + 1$ many parametrized submodular function minimizations and their algorithm for the lex-max flow over time problem in more detail, we explain how the minimal feasible time horizon of a quickest transshipment problem can be found by using a feasibility criterion of Klinz [Kli]. This is also how Hoppe and Tardos find the minimal feasible time horizon in their algorithm.

3.1.1 Checking Feasibility of a Transshipment Over Time Problem

Let $(\mathcal{N}, b, T)$ be a transshipment over time problem corresponding to a dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$, $b\colon S^+ \cup S^- \to \mathbb{Z}$ a supply/demand function on the terminals, and T a time horizon. The **transshipment over time feasibility problem** is the decision problem of checking whether there exists a flow over time f in $\mathcal{N}$ with time horizon T that satisfies all supplies and demands.

THE TRANSSHIPMENT OVER TIME FEASIBILITY PROBLEM

Instance: A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ with a supply/demand function $b\colon S^+ \cup S^- \to \mathbb{Z}$ on the terminals and a time horizon T

Task: Return YES if a flow over time solving the transshipment problem $(\mathcal{N}, b, T)$ exists and NO, otherwise

A criterion for the feasibility of a given transshipment problem $(\mathcal{N}, b, T)$ is due to Klinz [Kli]. It relies on the parametrized set function o^T defined in the following.

Definition 3.3.
Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and $T \geq 0$ a time horizon. The set function $o^T\colon 2^{S^+ \cup S^-} \to \mathbb{R}$ is defined as follows,

$$o^T(X) \coloneqq \begin{array}{l}\text{the maximal amount of flow that can be sent} \\ \text{from the sources in } S^+ \cap X \text{ to the sinks in } S^- \setminus X \text{ until time } T,\end{array}$$

for all $X \subseteq S^+ \cup S^-$.

Before we present some deeper results about the set function o^T, we start with the following observation:

Observation 3.4. *Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network, $T \geq 0$ a time horizon, and $\mathcal{N}^T = (D^T = (V^T, A^T), u^T, s^*, t^*)$ the corresponding time-expanded network as defined in Section 2.5.2. We have*

$$o^T(X) = \min\{u^T(\delta^+(U)) \mid U \subseteq V^T,\ X \cap S^+ \subseteq U \text{ and } (S^- \setminus X) \cap U = \varnothing\}.$$

Proof. By definition $o^T(X)$ is the maximum value of flow that can be sent from $S^+ \cap X$ towards $S^- \setminus X$ until time T. Let f be a flow over time with time horizon T in $\mathcal{N}$ that sends a flow of value $o^T(X)$ from the sources in $S^+ \cap X$ to the sinks in $S^- \setminus X$ until time T and assume that no flow is sent out of or towards the other terminals.
By Lemma 2.20 there exists a corresponding static flow x_f in the time-expanded network $\mathcal{N}^T$, in which the same amount of flow is sent from $S^+ \cap X$ towards $S^- \setminus X$. This is also the maximum amount of flow that can be sent between these sets of terminals in the time-expanded network.
Applying the max-flow min-cut theorem (Theorem 2.10) thus yields

$$o^T(X) = |f|_T = |x_f| = \min\{u^T(\delta^+(U)) \mid U \subseteq V^T,\ X \cap S^+ \subseteq U \text{ and } (S^- \setminus X) \cap U = \varnothing\}.$$

□

Assume we are given a transshipment over time problem $(\mathcal{N}, b, T)$. If there exist subsets $U \subseteq S^+$ and $X \subseteq S^-$ such that $o^T(U \cup X) < b(U \cup X)$, then it is obvious that a flow over time solving $(\mathcal{N}, b, T)$ does not exist: The maximum amount of flow that the sources in U can send towards the sinks in S^- is clearly upper bounded by $o^T(U \cup X) - b(X)$. Thus, if $o^T(U \cup X) < b(U \cup X)$, then the sources in U in a flow over time respecting the supplies and demands can never send more than $o^T(U \cup X) - b(X) < b(U)$. Thus, the supply of U can never be satisfied. It is due to Klinz [Kli] that the other direction also holds.

Theorem 3.5 (Feasibility Criterion, [Kli]).
A transshipment problem $(\mathcal{N}, b, T)$ is feasible if and only if

$$o^T(X) \geq b(X) \text{ for all } X \subseteq S^+ \cup S^-.$$

Proof. This theorem is a consequence of Gale's (see e.g., [Sch03]) feasibility theorem for transshipments in static networks and Observation 3.4. We denote by $\mathcal{N}^T$ the time-expanded network corresponding to $\mathcal{N}$ and the time horizon T. In contrast to the definition of $\mathcal{N}^T$ in Section 2.5.2 we now assume that $S^+ \cup S^-$ are also the terminals of $\mathcal{N}^T$. Additionally, we assume that there is a supply/demand function b^T on V^T which is equal to b on $S^+ \cup S^-$ and zero, otherwise.
By Lemma 2.20 the dynamic transshipment problem $(\mathcal{N}, b, T)$ is feasible if and only if the corresponding static transshipment problem $(\mathcal{N}^T, b^T)$ is feasible. Gale's theorem [Sch03] now states that $(\mathcal{N}^T, b^T)$ is feasible if and only if

$$u^T(\delta^+(U)) \geq b^T(U) \text{ for all } U \subseteq V^T. \tag{3.1}$$

Assume at first that $(\mathcal{N}, b, T)$ is **not** feasible, i.e., $(\mathcal{N}^T, b^T)$ is not feasible and by Gale's theorem there exists a subset $U^* \subseteq V^T$ with $u^T(\delta^+(U^*)) < b^T(U^*)$. This immediately implies that some terminals have to be contained in U^* as otherwise we would have $b^T(U^*) = 0$ and thus $u^T(\delta^+(U^*)) \geq b^T(U)$, because $u^T(\delta^+(U^*)) \geq 0$ by the definition of u^T.
Let $X \subseteq S^+ \cup S^-$ be the set of terminals contained in U^*, i.e., $\delta^+(U^*)$ is a cut in $\mathcal{N}^T$ separating $S^+ \cap X$ and $S^- \setminus X$. Observation 3.4 thus yields $o^T(X) < b(X)$.
If $(\mathcal{N}^T, b^T)$ is feasible, then the fact that all inequalities in (3.1) are fulfilled implies together with Observation 3.4 that $o^T(X) \geq b(X)$ for all $X \subseteq S^+ \cup S^-$. □

It was first observed by Megiddo [Meg74] that the function o^T is in fact a submodular function for every time horizon $T \geq 0$.

Theorem 3.6 ([Meg74; HT00]).
Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and $T \geq 0$ a time horizon. The function $o^T \colon 2^{S^+ \cup S^-} \to \mathbb{R}$ is a submodular function.

Proof. This theorem is a consequence of Observation 3.4 and the fact that cut functions in directed graphs are submodular. Let $X \subseteq S^+ \cup S^-$ and $Y \subseteq S^+ \cup S^-$. Observation 3.4 states that $o^T(X)$ equals the value of a minimum cut separating $X \cap S^+$ and $X \setminus S^-$ in the time-expanded network $\mathcal{N}^T$ (again with terminals $S^+ \cup S^-$), the similar statement holds for $o^T(Y)$. Let $U, U' \subseteq S^+$ such that $\delta^+_{\mathcal{N}^T}(U)$ is a minimum cut corresponding to $o^T(X)$ and $\delta^+_{\mathcal{N}^T}(U)$ a minimum cut corresponding to $o^T(Y)$. Using the fact that cut functions in directed graphs are submodular we achieve

$$\begin{aligned} o^T(X) + o^T(Y) &\overset{\text{Obs. } 3.4}{=} u^T(\delta^+_{\mathcal{N}^T}(U)) + u^T(\delta^+_{\mathcal{N}^T}(U')) \\ &\overset{\text{Ex. } 2.2}{\geq} u^T(\delta^+_{\mathcal{N}^T}(U \cup U')) + u^T(\delta^+_{\mathcal{N}^T}(U \cap U')). \end{aligned}$$

In order to show the submodularity of o^T it thus remains to be shown that $o^T(X \cup X) \leq u^T(\delta^+_{\mathcal{N}^T}(U \cup U'))$ and $o^T(X \cap X) \leq u^T(\delta^+_{\mathcal{N}^T}(U \cap U'))$, i.e., it suffices to deduce that $\delta^+_{\mathcal{N}^T}(U \cup U')$ is a cut separating $S^+ \cap (X \cup Y)$ and $S^- \setminus (X \cup Y)$ while $\delta^+_{\mathcal{N}^T}(U \cap U')$ is a cut in $\mathcal{N}^T$ separating $S^+ \cap (X \cap Y)$ and $S^- \setminus (X \cap Y)$.
For this purpose note that we have by assumption $S^+ \cap X \subseteq U$, $(S^- \setminus X) \cap U = \varnothing$, $S^+ \cap Y \subseteq U'$ and $(S^- \setminus Y) \cap U = \varnothing$. Thus, we get

$$\begin{aligned} S^+ \cap (X \cup Y) &= S^+ \cap X \cup S^- \cap Y \subseteq U \cup U', \\ (S^- \setminus (X \cup Y)) \cap (U \cup U') &= (S^- \setminus X) \cap (S^- \setminus Y) \cap (U \cap U') = \varnothing \text{ and} \\ S^+ \cap (X \cap Y) &\subseteq U \cap U', \\ (S^- \setminus (X \cap Y)) \cap (U \cap U') &= ((S^- \setminus X) \cup (S^- \setminus Y)) \cap (U \cap U') = \varnothing. \end{aligned}$$

□

In order to check whether a given transshipment over time problem $(\mathcal{N}, b, T)$ is feasible, we can thus just minimize the submodular function $o^T - b$. If the minimum is at least zero, then according to Theorem 3.5 the problem is feasible. If the minimum is smaller than zero, Theorem 3.5 implies the infeasibility of the problem. Fortunately, submodular functions can be minimized in strongly

polynomial time if we have a strongly polynomial time evaluation oracle for the submodular function. Thus, in order to be able to efficiently minimize the submodular function $o^T - b$, we need an efficient algorithm to evaluate this function at arbitrary subsets of $S^+ \cup S^-$. The following observation shows that we can compute the value of o^T at every subset of $S^+ \cup S^-$ by just one static minimum-cost flow computation in the static network corresponding to $\mathcal{N}$.

Observation 3.7. *Let $\mathcal{N} = (D, u, \tau, S^+, S^-)$ be a dynamic network, $T \geq 0$ a time horizon, $U \subseteq S^+$ and $X \subseteq S^-$. The value of a maximum s-t flow over time with time horizon T in the network $\tilde{\mathcal{N}}$ in which we attach a super-source s to the sources in U and a super-sink t to the sinks in $S^- \setminus X$ by arcs with zero transit time and infinite capacity is equal to $o^T(U \cup X)$. Thus, we can compute the value of $o^T(U \cup X)$ by using the algorithm of Ford and Fulkerson (see Algorithm 5) in the dynamic network $\tilde{\mathcal{N}}$.*

Putting together Theorems 3.5, and 3.6 and Observation 3.7 thus yields that the feasibility of a given transshipment problem can be checked in strongly polynomial running time by doing submodular function minimization of $o^T - b$.

Fact 3.8. *Given a transshipment over time problem $(\mathcal{N}, b, T)$, its feasibility can be checked in running time $\mathcal{O}(k^3 \log k \cdot \mathrm{MCF}(n, m) + k^4 \log^{\mathcal{O}(1)} k)$ by using the submodular function minimization algorithm of Lee, Sidford and Wong [LSW15] with $k := |S^+ \cup S^-|$.*
Given an instance $(\mathcal{N}, b)$ of the quickest transshipment problem, the minimal feasible time horizon T^* can be determined in strongly polynomial time by pairing a submodular function minimization algorithm with strongly polynomial running time with Megiddo's parametric search framework.
The best overall running time that can be achieved this way is $\mathcal{O}(\mathcal{C}_T(\mathrm{SFM}_{Lee}) \cdot (k^3 \log^3 k \cdot \mathrm{MCF}(n, m) + k^4 \log^{\mathcal{O}(1)} k))$ where $\mathcal{C}_T(\mathrm{SFM}_{Lee})$ is the number of comparisons depending on the parameter T done in the algorithm of Lee, Sidford and Wong.

In order to better understand the worst case running time of parametric search paired with the algorithm of Lee et al. for determining T^*, we need to look at the number of comparisons $\mathcal{C}_T(\mathrm{SFM}_{\mathrm{Lee}})$ in this algorithm that depend on the parameter T. Recall, that each such comparison leads to a new execution of the algorithm of Lee at al. during the parametric search framework (See Section 2.2). In the worst case the algorithm $\mathrm{SFM}_{\mathrm{Lee}}$ does $\mathcal{O}(k^3 \cdot \log k)$ calls to an evaluation oracle for the submodular function at hand, i.e., in our case it does in the worst case $\mathcal{O}(k^3 \cdot \log k)$ minimum-cost flow computations in a network in which the cost of one arc is linearly parametrized by our parameter T (see Observation 3.7 and Algorithm 5). That is, each of these minimum-cost flow computations will contain at least one comparison depending on our parameter T (and probably more). We can thus conclude with the following observation:

Observation 3.9. *Let $(\mathcal{N}, b)$ be an instance of the quickest transshipment problem. The best known strongly polynomial time algorithm to determine T^* has a running time in $\mathcal{O}(\mathcal{C}_T(\mathrm{SFM}_{Lee}) \cdot (k^3 \log k \cdot \mathrm{MCF}(n, m) + k^4 \log^{\mathcal{O}(1)} k))$ with $k := |S^+ \cup S^-|$, $n = |V|$ and $m = |A|$. Additionally we have*

$$\mathcal{C}_T(\mathrm{SFM}_{Lee}) \notin \mathcal{O}(k^{3-\varepsilon} \log k \cdot \mathcal{C}_T(\mathrm{MCF}(n, m)))$$

for every $\varepsilon > 0$ and $\mathcal{C}_T(\mathrm{MCF}(n, m)) \geq 1$.
Overall, for this specific example, pairing the algorithm of Lee et al. with parametric search increases the worst case running time of this algorithm by at least a factor of $k^3 \log k$.

3.1.2 Lexicographically Maximum Flows Over Time

In the previous section we described the first part of the algorithm of Hoppe and Tardos for determining the minimal feasible time horizon T^* of a quickest transshipment problem $(\mathcal{N}, b)$. The second part of their algorithm is the computation of the actual flow over time f solving the transshipment over time problem $(\mathcal{N}, b, T^*)$. The main idea of the flow computation is to efficiently reduce it to a **lexicographically maximum flow over time problem** and to solve this problem efficiently. In this section we define the lexicographically maximum flow over time problem and shortly present the algorithm that Hoppe and Tardos developed in [HT00] to solve such problems.

Definition and Existence of Lexicographically Maximum Flows Over Time. Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network. Lexicographically maximal flows over time have, similar to their static equivalent (see Section 2.4.4), the property that they maximize the net amount of flow that leaves the terminals during a given time horizon with respect to a linear ordering on the terminals. For the formal definition of lexicographically maximum flow over time we assume that we are given a total order $\prec$ on the set of terminals $S^+ \cup S^-$. For simplicity, say $S^+ \cup S^- = \{s_1, s_2, \ldots, s_k\}$ with $s_i \prec s_j$ for all $i, j \in \{1, \ldots, k\}$ with $i < j$. We say that $s \in S^+ \cup S^-$ has a **higher priority** than $s' \in S^+ \cup S^-$ with respect to $\prec$ if $s \prec s'$.
A flow over time f with time horizon T is a **lexicographically maximum (lex-max) flow over time** with respect to the ordering $\prec$ if it maximizes the net amount of flow leaving the terminals within the given order $\prec$. For the sinks this means that f *minimizes the flow entering the sinks* within the given order. A flow over time f^1 in $\mathcal{N}$ with time horizon T is **lexicographically bigger** than a flow over time $f2$ with time horizon T with respect to $\prec$, or $f^1 \geq_{\text{lex}} f^2$, if

$$\exists l \in \{0, 1, \ldots, k-1\} \text{ with } |\operatorname{net}_{f^1}(s_i, T)| = |\operatorname{net}_{f^2}(s_i, T)| \text{ for all } i \in \{1, 2, \ldots, l\} \text{ and}$$
$$|\operatorname{net}_{f^1}(s_{l+1}, T)| > |\operatorname{net}_{f^2}(s_{l+1}, T)|,$$

or if

$$|\operatorname{net}_{f^1}(s_i, T)| = |\operatorname{net}_{f^2}(s_i, T)| \text{ for all } i \in \{1, 2 \ldots, k\}.$$

A flow over time f in the dynamic network $\mathcal{N}$ with time horizon T is a **lexicographically maximum flow over time** with respect to $\prec$ if it is lexicographically bigger than any other flow over time f' in $\mathcal{N}$ with time horizon T. The **lexicographically maximum flow over time problem** is the problem of finding a lexicographically maximal flow over time with respect to some linear order on the terminals and some time horizon T.

LEXICOGRAPHICALLY MAXIMUM FLOW OVER TIME PROBLEM $(\mathcal{N}, \prec, T)$

Instance: A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$, a linear order $\prec$ on $S^+ \cup S^-$, and a time horizon $T \geq 0$

Task: Compute a flow over time f in $\mathcal{N}$ with time horizon T that is lexicographically maximal with respect to the linear order $\prec$

Minieka [Min73] and Megiddo [Meg74] derived the existence of static lexicographically maximal flows (see also Section 2.4.4). Their result also implies the existence of lexicographically maximal flows over in using the time-expanded network. As the time-expanded network is exponentially larger than the underlying original dynamic network, the algorithms of Minieka or Megiddo, and also the method using the algorithm of Gallo et al. [GGT89] (see Section 2.4.4) cannot be used directly to solve the lexicographically maximum flow over time problem efficiently. In [HT00] Hoppe and Tardos present a strongly polynomial time algorithm for this problems that only works on the original dynamic network without using any significant forms of expansion.

The Algorithm of Hoppe and Tardos for the Lexicographically Flow Over Time Problem. As the lexicographically maximal flow over time algorithm of Hoppe and Tardos will be important throughout this thesis, we will give a verbal description of the workings of this algorithm and also its formal formulation (see Algorithm 6).

Let $(\mathcal{N}, \prec, T)$ be a lex-max flow over time problem and assume that $S^+ \cup S^- = \{s_1, \ldots, s_k\}$ with $s_1 \prec s_2 \prec \ldots \prec s_k$. Note that a lex-max flow over time (as a consequence from our knowledge regarding static lex-max flows) fulfills the property that in such a flow over time a source $s \in S^+$ only sends flow towards a sink $t \in S^-$ if we have $s \prec t$. The first step of the algorithm of Hoppe and Tardos is to attach a super-source s to all the sources in S^+. TThus, the algorithm of Hoppe and Tardos iterates over all terminals starting with the terminals with *lower* priority. Basically nothings happen before the first sink s_i is found in iteration i of the algorithm. The sources that appeared in the previous iterations are ordered behind all the sinks in the ordering $\prec$ and thus

do not need to send any flow according to our remark above. Thus, all that is done in these iterations is to disconnect these sources from the super-source s such that they are ignored in the following iterations. The first sink s_i intuitively needs to receive as much flow as possible until time T from all the sources that have a higher priority than s_i. Thus, this is what is computed in iteration i: a max-flow over time with time horizon T from s to s_i using the algorithm of Ford and Fulkerson. In the subsequent iterations the flow towards s_i is not changed. Assume that in iteration j the next sink is found. Thus, the sources $s_{j+1}, \dots, s_{i-1}$, only need to send flow towards s_i. However, the amount of flow sent out of these source needs to respect the given order $\prec$. In each iteration $l \in \{i-1, \dots, j+1\}$ it is made sure that the flow out of s_l respects the given order by rerouting as much flow as possible towards the sources with a higher priority than s_l. Afterwards s_l is disconnected from the super-source s in order to ensure that in the subsequent iterations the flow out of s_l is not changed. For the next sink s_j it again holds that it is only supposed to receive flow from the sources that have a higher priority than itself. So, in iteration j a maximum flow over time from the super-source s (to which only the sources with a higher priority than s_j remain connected) to s_j with time horizon T is computed in the residual network of all the static flow computations from the previous iterations. This way we make sure that s_j receives the correct amount of flow. In the following iterations the flow originating from the sources that only send flow towards s_j and s_i is again rerouted accordingly, and so on. Algorithm 6 shows a formal description of this algorithm.

Algorithm 6: Algorithm for the lex-max flow over time problem, LexMax$(\mathcal{N}, \prec, T)$

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^-, S^-)$, a linear order $\prec$ on $S^+ \cup S^- = \{s_1, s_2, \dots, s_k\}$ with $s_i \prec s_j$ for $i < j$, and a time horizon $T \geq 0$
Output : A flow over time f solving the lex-max flow over time problem $(\mathcal{N}, \prec, T)$

$k \leftarrow |S^+ \cup S^-|$
$V \leftarrow V \cup \{s\}$
$A^{k+1} \leftarrow A \cup \{(s, s') \mid s' \in S^+\}$
extent u to A^{k+1} by defining $u_{(s,s')} = \infty$ for all $s' \in S^+$
extent τ to A^{k+1} by defining $\tau_{(s,s')} = 0$ for all $s' \in S^+$
$x^{k+1} \leftarrow 0$
$X^{k+1} \leftarrow \varnothing$
for $i \in \{k, k-1, \dots, 1\}$ **do**
    $A^i \leftarrow A^{i+1}$
    if $s_i \in S^-$ **then**
        $A^i \leftarrow A^i \cup \{(s_i, s)\}$ with $u_{(s_i,s)} = \infty$ and $\tau_{(s_i,s)} = -T$
        $\mathcal{N}^i \leftarrow (V, A^i)$
        $y^i \leftarrow$ minimum-cost circulation in $\mathcal{N}^i_{x^{i+1}}$ with τ as costs
    else
        $A^i \leftarrow A^i \setminus \{(s, s_i)\}$
        $\mathcal{N}^i \leftarrow (V, A^i)$
        $y^i \leftarrow$ minimum-cost s-s_i flow in $\mathcal{N}^i_{x^{i+1}}$ with τ as costs
    end
    $x^i \leftarrow x^{i+1} + y^i$
    compute a path decomposition of the static flow y^i in $\mathcal{N}^i_{x^{i+1}}$ given by $(\mathcal{P}^i, w^i)$
    $X^{i+1} \leftarrow X^i \cup (\mathcal{P}^i, w^i)$
end
return *generalized temporally repeated flow f corresponding to X^1*

Theorem 3.10 (Hoppe and Tardos, [HT00]).
Let $\mathcal{N}$ be a dynamic network with k terminals, $\prec$ a total order on the terminals of $\mathcal{N}$, and $T \geq 0$ a time horizon. The lex-max flow over time problem $(\mathcal{N}, \prec, T)$ can be computed via k minimum-cost flow computations using Algorithm 6.

In order to understand the second part of the algorithm of Hoppe and Tardos for solving quickest transshipment problems, we need the following lemma about the characteristic vectors of lex-max flows over time.

Lemma 3.11. *Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network, $T \geq 0$ a time horizon, $\prec$ a total order on $S^+ \cup S^-$, and $f_\prec$ a lex-max flow over time in $\mathcal{N}$ with time horizon T with respect to $\prec$. For each $v \in S^+ \cup S^-$ we have*

$$\text{net}_{f_\prec}(\{v' \in S^+ \cup S^- \mid v' \preceq v\}, T) = o^T(\{v' \in S^+ \cup S^- \mid v' \preceq v\}),$$

i.e.,

$$\text{net}_{f_\prec}(v, T) = o^T(\{v' \in S^+ \cup S^- \mid v' \preceq v\}) - o^T(\{v' \in S^+ \cup S^- \mid v' \prec v\}).$$

On the other hand, a flow over time with time horizon T fulfilling these equation is a lex-max flow over time with respect to $\prec$.

Proof. Using Lemma 2.20, the flow over time $f_\prec$ induces a static flow $x_\prec$ in the time-expanded network $\mathcal{N}^T$. For the purpose of the proof we assume the set of terminals of $\mathcal{N}^T$ is given by $S^+ \cup S^-$. Because of Lemma 2.20, since $f_\prec$ is a lax-max flow over time in $\mathcal{N}$, the static flow $x_\prec$ is a static lex-max flow in $\mathcal{N}^T$ with respect to the same order on the terminals $S^+ \cup S^-$. By Lemma 2.19 we know that

$$\text{net}_{x_\prec}(\{v' \in S^+ \cup S^- \mid v' \preceq v\}) \overset{\text{Lem. 2.19}}{=} \max_{\mathcal{N}^T}(\{v' \in S^+ \mid v' \preceq v\}, S^- \setminus \{v \in S^- \mid v' \prec v\})$$

for all $v \in S^+ \cup S^-$.
Using the correspondence of $x_\prec$ and $f_\prec$ and the definition of o^T, this implies

$$\begin{aligned}\text{net}_{f_\prec}(\{v' \in S^+ \cup S^- \mid v' \preceq v\}, T) &\overset{\text{Lem. 2.20}}{=} \text{net}_{x_\prec}(\{v' \in S^+ \cup S^- \mid v' \preceq v\}) \\ &\overset{\text{Lem. 2.19}}{=} \max_{\mathcal{N}^T}(\{v' \in S^+ \mid v' \preceq v\}, S^- \setminus \{v \in S^- \mid v' \prec v\}) \\ &= \quad o^T(\{v' \in S^+ \cup S^- \mid v' \preceq v\}),\end{aligned}$$

for all $v \in S^+ \cup S^-$. The last statement of the lemma is immediate. □

3.1.3 The Algorithm of Hoppe and Tardos

The next main step of the algorithm of Hoppe and Tardos for solving quickest transshipment problems is the reduction of a given feasible transshipment over time problem $(\mathcal{N}, b, T)$ to a lex-max flow over time problem, which is then solved using Algorithm 6.
Let $(\mathcal{N}, b)$ be a quickest transshipment problem with minimal feasible time horizon T^*. We call a subset $X \subseteq S^+ \cup S^-$ a **tight** subset if

$$o^T(X) = b(X),$$

i.e., in this case the maximal amount of flow that can be sent from the sources in $X \cap S^+$ towards the sinks in $S^- \setminus X$ until time T exactly equals the sum of the supplies and demands of the terminals in X. The empty set trivially is a tight subset, because $b(\varnothing) = 0 = o^T(\varnothing)$. The second trivial tight subset is the whole set of terminals $S^+ \cup S^-$ because by assumption we have $b(S^+ \cup S^-) = 0$ and clearly also $o^T(S^+ \cup S^-) = 0$.
The central observation for the reduction of Hoppe and Tardos is that when we are given a complete chain of subsets of terminals, i.e.,

$$\varnothing = S_0 \subsetneq S_1 \subsetneq S_2 \subsetneq \ldots \subsetneq S_k = S^+ \cup S^-$$

with $k = |S^+ \cup S^-|$ and all of these subsets are tight, then the corresponding transshipment over time problem can be solved by a single lex-max flow over time computation. This observation is summarized in the following lemma.

Lemma 3.12. *Let $(\mathcal{N}, b)$ be a quickest transshipment problem with minimal feasible time horizon T^*. Assume that $S^+ \cup S^- = \{s_1, s_2, \ldots, s_k\}$, $S_i = \{s_1, \ldots, s_i\}$ for all $i \in \{1, \ldots, i\}$, and that the chain $\varnothing = \subsetneq S_1 \subsetneq S_2 \subsetneq \ldots \subsetneq S_k = S^+ \cup S^-$ is a chain of tight subsets. Then a flow over time solving $(\mathcal{N}, b, T^*)$ can be obtained by computing a lex-max flow over time solving $(\mathcal{N}, \prec, T^*)$, where $\prec$ is defined by $s_i \prec s_j$ for all $i, j \in \{1, \ldots, k\}$ with $i < j$.*

Proof. Let f be a flow over time with time horizon T^* solving $(\mathcal{N}, b, T^*)$. The flow over time f clearly has to fulfill

$$\text{net}_f(\{s_1, \dots, s_i\}, T^*) = b(\{s_1, \dots, s_i\}) \overset{\text{tight set}}{=} o^{T^*}(\{s_1, \dots, s_i\}),$$

for all $i \in \{1, \dots, r\}$ and thus the flow over time f is a lex-max flow over time with respect to the order $\prec$ by Lemma 3.11. On the other hand, for a lex-max flow over time $f_\prec$ with respect to $\prec$ and T^* the following holds for all $i \in \{1, \dots, k\}$,

$$\begin{aligned}\text{net}_f(s_i, T^*) &\overset{\text{Lem. 3.11}}{=} o^{T^*}(\{s_1, \dots s_i\}) - o^{T^*}(\{s_1, \dots s_{i-1}\}) \\ &\overset{\text{tight sets}}{=} b(\{s_1, \dots s_i\}) - b(\{s_1, \dots s_{i-1}) = b(s_i),\end{aligned}$$

which implies that $f_\prec$ is a transshipment solving $(\mathcal{N}, b, T^*)$. □

The algorithm of Hoppe and Tardos for solving a quickest transshipment problem $(\mathcal{N}, b)$ with feasible time horizon T start with the chain $\varnothing \subseteq S^+ \cup S^-$ (which is a trivial chain of tight sets) and by successively attaching new sources and sinks by arcs with suitably chosen capacities and transit times to $\mathcal{N}$, they manage to obtain a new extended dynamic network $\mathcal{N}'$ with a new set of terminals $S'^+ \cup S'^-$, new supplies/demands b' on this set of terminals together with a complete chain of tight subsets of $S'^+ \cup S'^-$ with respect to b'. The dynamic network $\mathcal{N}'$ also has the property that a flow over time with time horizon T in $\mathcal{N}'$ can be turned into an equivalent flow over time with time horizon T in $\mathcal{N}$.
The algorithm of Hoppe and Tardos terminates after at most $|S^+ \cup S^-|$ iterations and in each iteration at most two new sources or sinks are attached to the network. In order to compute the suitable capacities or transit times of the arcs by which these new terminals are attached to the network, a parametric submodular function minimization problem has to be solved.

Theorem 3.13 (Hoppe and Tardos, [HT00]).

The algorithm of Hoppe and Tardos solves a feasible transshipment problem $(\mathcal{N}, b, T)$ in strongly polynomial time with the help of at most $2|S^+ \cup S^-|$ parametric submodular function minimization and a single lex-max flow over time computation.
An instance of a quickest transshipment problem $(\mathcal{N}, b)$ can thus be solved in strongly polynomial time with the help of at most $1 + 2|S^+ \cup S^-|$ parametric submodular function minimizations and a single lex-max flow over time computation.

3.1.4 Summary and Our Results

We conclude with a summary of the problems presented in this section. All of these problems are listed in Table 3.1. After that we compare the results we achieve in this thesis with the previously best known results.

Table 3.1: Overview over all flow over time problems introduced in this section

Problem	Setting	Objective
Quickest Flow	single source, single sink with supply/demand	flow over time with minimum time horizon fulfilling supply/demand
Minimum Feasible Time Horizon	multiple sources and sinks with supplies and demands	minimum time horizon such that supplies/demands can be fulfilled
Transshipment Over Time	multiple sources and sinks with supplies and demands, a time horizon T	flow over time with time horizon T that fulfills supplies/demands
Quickest Transshipment	multiple sources and sinks with supplies and demands	flow over time with minimal time horizon that fulfills supplies/demands

Minimal Feasible Time Horizon Problem.

Previously best known algorithm: The algorithm of Lee et al. for SFM coupled with the parametric search framework with an overall running time of

$$\mathcal{O}\big(\mathcal{C}_T(\mathrm{SFM}_{\mathrm{Lee}}) \cdot (k^3 \log k \cdot \mathrm{MCF}(n,m) + k^4 \log^{\mathcal{O}(1)} k)\big),$$

where $k = |S^+ \cup S^-|$ and $\mathcal{C}_T(\mathrm{SFM}_{\mathrm{Lee}})$ is the number of comparison done in the algorithm $\mathrm{SFM}_{\mathrm{Lee}}$ that depend on the parameter T.

Our Results:

- *Dynamic networks with only a single source s:* $|S^-|$ many submodular function minimizations are needed, resulting in an running time of

  $$\mathcal{O}(|S^-|^4 \log |S^-|) \cdot \mathrm{MCF}(n,m) + |S^-|^5 \log^{\mathcal{O}(1)} |S^-|).$$

  This yields a running time improvement of at least a factor of $|S^-|^2 \cdot \log |S^-|$ (see Observation 3.9).

- *Dynamic networks with only a single source t:* $|S^+|$ many submodular function minimizations are needed, resulting in an running time of

  $$\mathcal{O}(|S^+|^4 \log |S^+|) \cdot \mathrm{MCF}(n,m) + |S^+|^5 \log^{\mathcal{O}(1)} |S^+|).$$

  This yields a running time improvement of at least a factor of $|S^+|^2 \cdot \log |S^+|$ (see Observation 3.9).

The Transshipment Over Time Problem.

Previously best known algorithm: One submodular function minimization to check the feasibility and afterwards one execution of the algorithm of Hoppe and Tardos (requiring $2|S^+ \cup S^-|$ many parametrized submodular function minimizations) to compute the transshipment. This results in an overall running time of

$$\mathcal{O}\big((1 + 2k\mathcal{C}_\lambda(\mathrm{SFM}_{\mathrm{Lee}})) \cdot (k^3 \log k \cdot \mathrm{MCF}(n,m) + k^4 \log^{\mathcal{O}(1)} k)\big),$$

where $k = |S^+ \cup S^-|$ and $\mathcal{C}_\lambda(\mathrm{SFM}_{\mathrm{Lee}})$ is the number of comparisons done in the algorithm $\mathrm{SFM}_{\mathrm{Lee}}$ depending on the parameter λ.

Our Results: One execution of a submodular function minimization algorithm using the framework of Cunningham, for example the Algorithm of Orlin $\mathrm{SFM}_{\mathrm{Orlin}}$, resulting in an overall running time of

$$\mathcal{O}(k^5 \cdot \mathrm{MCF}(n,m) + n^6).$$

This yields a running time improvement of at least a factor of $k^2 \cdot \log k$ (see Observation 3.9).

The Quickest Transshipment Problem.

Previously best known algorithm: One parametrized submodular function minimization to determine the minimal feasible time horizon T^* and afterwards one execution of the algorithm of Hoppe and Tardos (requiring in the worst case $2|S^+ \cup S^-|$ many parametrized submodular function minimizations) to compute the transshipment. This results in an overall running time of

$$\mathcal{O}\big((1 + 2k) \cdot \mathcal{C}_\lambda(\mathrm{SFM}_{\mathrm{Lee}}) \cdot (k^3 \log k \cdot \mathrm{MCF}(n,m) + k^4 \log^{\mathcal{O}(1)} k)\big),$$

where $k = |S^+ \cup S^-|$ and $\mathcal{C}_\lambda(\mathrm{SFM}_{\mathrm{Lee}})$ is the number of comparison done in the algorithm $\mathrm{SFM}_{\mathrm{Lee}}$ that depend on the parameter λ.

Our Results:

- *Dynamic networks with multiple sources and multiple sinks:* One parametrized submodular function minimization to determine the minimal feasible time horizon T^* and afterwards one execution of a submodular function minimization algorithm relying on the framework of Cunningham, for example the algorithm of Orlin, $\text{SFM}_{\text{Orlin}}$. This results in an overall running time of

$$\mathcal{O}\big(\mathcal{C}_T(\text{SFM}_{\text{Lee}}) \cdot (k^3 \log k \cdot \text{MCF}(n,m) + k^4 \log^{\mathcal{O}(1)} k) + k^6 + k^5 \cdot \text{MCF}(n,m)\big),$$

  with $k = |S^+ \cup S^-|$, yielding an overall running time improvement of at least a factor of $k \cdot \log k$ (see Observation 3.9).

- *Dynamic networks with only a single source s:* Using our results for the computation of the minimum feasible time horizon, we obtain an overall running time of

$$\mathcal{O}\big((k^4 \cdot \log k + k^5) \cdot \text{MCF}(n,m) + k^4 \log^{\mathcal{O}(1)} k + k^6\big),$$

  with $k = |S^-|$, resulting in a running time improvement of at least a factor of a factor of k^2.

- *Dynamic networks with only a single source t:* Using our results for the computation of the minimum feasible time horizon, we obtain an overall running time of

$$\mathcal{O}\big((k^4 \cdot \log k + k^5) \cdot \text{MCF}(n,m) + k^4 \log^{\mathcal{O}(1)} k + k^6\big),$$

  with $k = |S^+|$, resulting in a running time improvement of at least k^2.

3.2 Earliest Arrival Flows and Earliest Arrival Transshipments

We already introduced the maximum flow over time problem and the quickest transshipment problem. In both of these flow over time problems *exactly one* objective is optimized: a maximum flow over time in a dynamic network $\mathcal{N}$ maximizes the amount of flow that has reached the (single) sink t until a given time horizon T, while a quickest transshipment f' fulfills all supplies and demands as fast as possible, i.e., with the minimal possible time horizon. Both problems have in common that they are only interested in the result at the time horizon. However, one can easily imagine many applications for which it is also important how much flow has reached the sinks at points in time before the time horizon. We already described how quickest transshipments can be used for evacuation planning. An evacuation strategy obtained from a quickest transshipment has indeed the property that all endangered people are rescued as quickly as possible. However, there are emergencies for which this is not sufficient. Assume, for example, that an earthquake occurred and severely damaged a large building. It might be foreseeable that the damaged building will collapse in the near future, but the precise point in time when this disaster will happen is of course not known. The goal of an efficient evacuation strategy is to lead as many people as possible to safety. However, what might happen in such a situation is that the time until the building collapses (which is not known) is shorter than the minimal time needed to bring all people out of the building. Thus, in such a situation it is advisable to resort to a different strategy. Since it is not known whether all people can be saved, it should at least be made sure that as much people as possible are rescued. This can be achieved by ensuring that for each point in time as much people as possible have been lead out of the building (of course while still making sure that supplies and demands are not violated). This property is modeled by a flow over time, which is a special case of a quickest transshipment, called **earliest arrival transshipment**.
A similar special case can be defined for the maximum flow over time problem: an **earliest arrival flow** is a flow over time with the property that for each point in time until the given time horizon T the amount of flow that has reached the sink is maximized.

Definition 3.14.
Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and $T \geq 0$ a time horizon. We say that a flow over time f in $\mathcal{N}$ fulfills the ***earliest arrival property*** *if as much flow as possible has arrived at the sinks for each point in time $\theta \in [0, T)$ simultaneously, with the additional property that potentially given supplies and demands b on the terminals are not violated.*

The **earliest arrival pattern** corresponding to a quickest transshipment problem $(\mathcal{N}, b)$ is the next important concept in the context of earliest arrival flows. Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network, b a supply/demand function on the terminals, and T the minimal time horizon required to fulfill all supplies and demands. Assume f is a flow over time with time horizon T in $\mathcal{N}$ that also fulfills the given supplies and demands b on the terminals. A **pattern** is a function $p\colon [0, T) \to \mathbb{R}_{\geq 0}$. The **arrival pattern** p_f of the flow over time f is defined by

$$p_f(\theta) \coloneqq |f|_\theta \text{ for all } \theta \in [0, T).$$

Let f^*_θ be a flow over time in $\mathcal{N}$ with time horizon θ with maximal flow value subject to the constraint that the supplies and the demands are not violated. The **earliest arrival pattern** for $(\mathcal{N}, b)$ is then defined as

$$p^*(\theta) \coloneqq |f^*_\theta|_\theta \text{ for all } \theta \in [0, T).$$

Similarly, one can define the earliest arrival pattern corresponding to a maximum flow over time problem with time horizon T in a dynamic network $\mathcal{N}$. In this case the only thing that changes is that there are no supplies and demands that need to be respected. A network flow over time f with a given time horizon T is an **earliest arrival flow** if its arrival pattern equals the earliest arrival pattern. A dynamic transshipment adhering to the earliest arrival pattern is an **earliest arrival transshipment**.

3.2.1 Earliest Arrival Flows

We start by giving a short introduction to earliest arrival flows. Let $\mathcal{N} = (D, u, \tau, s, t)$ be a dynamic network with only a single source s and a single sink t and T a time horizon. In contrast to a maximum flow over time with time horizon T in $\mathcal{N}$, an **earliest arrival flow** in $\mathcal{N}$ with time horizon T is a flow over time that maximizes the amount of flow that has reached the sink t at each point in time $\theta \in [0, T)$ *simultaneously*. The problem of computing an earliest arrival flow with time horizon T in a dynamic network $\mathcal{N}$ is called **earliest arrival flow problem**.

Earliest Arrival Flow Problem

Instance:	A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, s, t)$ with a single source s, a single sink t, and a time horizon $T \geq 0$ simultaneously
Task:	Compute a flow over time f that is maximal for each point in time $\theta \in [0, T)$

As for the case of the maximum flow over time problem we can assume without loss of generality that the dynamic network we consider has only a single source and a single sink. If this it not the case, we can just attach a super-source and a super-sink by arcs with zero transit time and infinite capacity to the dynamic network.

The history of research regarding earliest arrival flows is nearly as long as the history of flows over time themselves. In fact, these flows have first been studied by Gale [Gal59] in 1959 shortly after Ford and Fulkerson introduced flows over time. Gale showed the existence of earliest arrival flows for the discrete time model. Existence in the continuous setting has been derived by Philpott

in [Phi90] in 1990. Gale's existence proof was however not constructive, it relied on the Supply-and-Demand-Theorem that has been proven by Ford and Fulkerson [FF62]. Thus, it remained an open problem how to compute earliest arrival flows until Minieka [Min73] and Wilkinson [Wil71] came up with the first pseudo-polynomial algorithms to compute such flows. Minieka showed that an earliest arrival flow problem corresponding to $\mathcal{N} = (D, u, \tau, s, t)$ and $T \geq 0$ can be solved by computing a static lexicographically maximal flow in the time-expanded network $\mathcal{N}^T$. To simplify notation, we assume that T is integral. The same algorithm, however, also works for rational time horizons. Recall that in the time-expanded network $\mathcal{N}^T$ there are T copies $t^1, t^2, \dots, t^T$ of the sink t and an over super-source s^*. In order to compute an earliest arrival flow f using $\mathcal{N}^T$, we define a linear order $\prec$ on $\{s^*, t^1, t^2, \dots, t^T\}$ by $s^* \prec t^T \prec t^{T-1} \prec \dots \prec t^1$ and compute the corresponding static lexicographically maximum flow x^T in $\mathcal{N}^T$. Note that we in this case consider $\{t^1, t^2, \dots, t^T\}$ to be the set of sinks of $\mathcal{N}^T$. The earliest arrival flow f is then obtained using Lemma 2.20.

Algorithm 7: Algorithm for the earliest arrival flow problem with time expansion

```
Input  : A dynamic network N = (D = (V, A), u, τ, s, t) and a time horizon T ≥ 0
Output : An earliest arrival flow f
1 N^T ← the time-expanded network corresponding to N and T
2 ≺ ← the linear order on {s*, t^1, t^2, ..., t^T} defined by s* ≺ t^T ≺ t^{T-1} ≺ ... ≺ t^1
3 x^T ← a static lexicographically maximum flow in N^T corresponding to ≺
4 f(a, θ) ← x^T(a^{⌊θ⌋+1}) for all a ∈ A and θ ∈ [0, T)
5 return f
```

That Algorithm 7 indeed returns a discrete earliest arrival flow follows immediately from the properties of static lexicographically maximum flows and Lemma 2.20. However, the returned flow is also an earliest arrival flow in the continuous setting. To see this we have at first to investigate a few properties of the earliest arrival pattern p^* corresponding to a maximum flow over time problem given by $\mathcal{N}$ and a time horizon T. Recall that $o^\theta(\{s\})$ is the maximal amount of flow that can be sent from the source s towards the sink t until time θ.

Observation 3.15. *Let $\mathcal{N} = (D = (V, A), u, \tau, s, t)$ be a dynamic network and T a time horizon. The earliest arrival pattern p^* corresponding to $\mathcal{N}$ and T is given by*

$$p^*(\theta) = o^\theta(\{s\}) \text{ for all } \theta \in [0, T),$$

which is a continuous piecewise linear and convex function.

Proof. The only thing that needs to be shown is that p^* is piecewise linear and convex. We know that $o^\theta(\{s\})$ equals the maximal possible value of

$$\theta \cdot |x| - \sum_{a \in A} \tau_a \cdot x_a = \sum_{P \in \mathcal{P}} (T - \tau(P)) \cdot x_P,$$

where x is a static s-t flow in $\mathcal{N}$. Thus, the problem of computing $o^\theta(\{s\})$ can also be formulated as a parametric linear programming problem, for which it is known that the optimal objective value parametrized by θ is a continuous piecewise linear convex function [DT03]. □

We can now argue that the flow over time f obtained from the static lex-max flow x^T (which can assumed to be an integral flow) in $\mathcal{N}^T$ is also a continuous earliest arrival flow. By definition, in the flow over time f obtained from x^T the rate at which flow travels into an arc $a \in A$ is constant during $[\theta, \theta + 1)$ for all $\theta \in \{0, 1, \dots, T - 1\}$. This implies that the function $\theta \mapsto |f|_\theta$ for $\theta \in [0, T)$ is a piecewise linear function in which breakpoints only occur at integral points in time because of the integrality of the transit times of the arcs. Thus, we have that $o^\theta(\{s\})$ and $|f|_\theta$ are both piecewise linear functions in θ that are equal at all points at which breakpoints can occur. This implies that both functions are equal and thus f is an earliest arrival flow.
Note that this algorithm of Minieka is of pseudo-polynomial running time, as it works in the time-expanded network. Because of this fact, it also needs a pseudo-polynomial amount of space.

Wilkinson [Wil71] observed that it is possible to compute earliest arrival flows without the use of time expansion using the *successive shortest path algorithm*. He showed that sending flow along the paths occurring in the successive shortest path algorithm from s to t that only augments along paths of length at most T yields an earliest arrival flow. The rate at which we send flow along the paths is given by the flow value they obtain during the successive shortest path algorithm.

Algorithm 8: Algorithm for the earliest arrival flow problem without time expansion, $\mathrm{EAF}(\mathcal{N}, T)$

```
Input  : A dynamic network N = (D = (V, A), u, τ, s, t), and a time horizon T ≥ 0
Output : A generalized temporally repeated flow resulting in an earliest arrival flow
x_P ← 0 for all P ∈ P↔
x ← static s-t flow with generalized path decomposition (x_P)_{P∈P↔}
P ← shortest s-t path in N
while τ(P) < T do
    γ ← min{τ(a) | a ∈ P}
    augment x along P by γ
    P ← shortest s-t path in N_x
end
f ← generalized temporally repeated flow with time horizon T corresponding to (x_P)_{P∈P↔}
return f
```

See Figure 3.2 for an example of the earliest arrival flow obtained from Algorithm 8. Comparing Figure 3.2 and Figure 3.1 also shows two maximum flows over time with time horizon 8, where the former fulfills the earliest arrival property and the latter doesn't. Algorithm 8 clearly only requires polynomial space as it is executed on the original network. However, because of the pseudo-polynomial worst case running time of the successive shortest path algorithm, this algorithm is still also of non polynomial running time. The complexity of the earliest arrival flow problem was open until 2015 when Disser and Skutella [DS15] managed to prove that computing an earliest arrival flow is $\mathcal{NP}$-hard by showing that it is $\mathcal{NP}$-hard to obtain the average arrival time of flow in an earliest arrival flow. Note that a flow over time minimizes its average arrival time if and only if it is an earliest arrival flow [JR82]. For special classes of networks it is however possible to find efficient algorithms for the earliest arrival flow problem. For example, for series-parallel graphs such an algorithm was described by Ruzika, Sperber and Steiner [RSS11].

A generalization of earliest arrival flows are flows over time that maximize the amount of flow that has arrived at the sink at multiple different points in time but not necessarily at all point in time. Such flows are called **multiple deadline flows** which have been introduced in 2010 by Stiller and Wiese [SW10]. It is clear that multiple deadline flows can also be computed using the successive shortest path algorithm. However, Stiller and Wiese also give an algorithm that only needs k minimum-cost flows computation in the underlying dynamic networks, and hence their algorithm requires only polynomial space and is even of strongly polynomial running time. Here, k is the number of points in time at which the flow over time is supposed to be maximal.

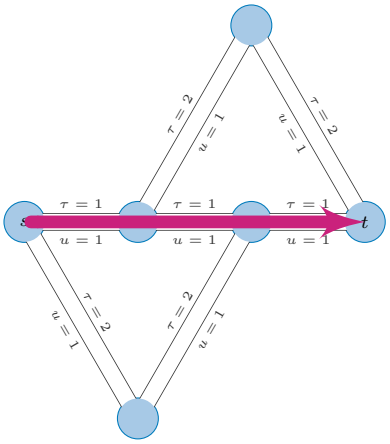
3.2.2 Earliest Arrival Transshipments.

In this section we look at the **earliest arrival transshipment problem**.

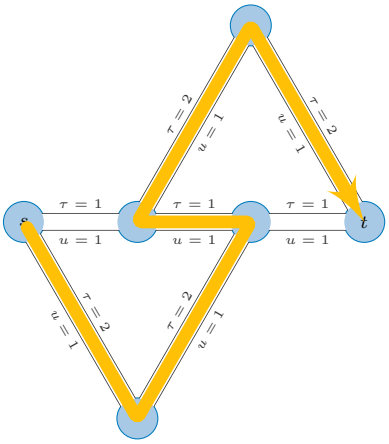
EARLIEST ARRIVAL TRANSSHIPMENT PROBLEM $(\mathcal{N}, b)_{\mathrm{EAT}}$

Instance: A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, t)$ and a supply/demand function $b\colon S^+ \cup S^- \to \mathbb{Z}$ on the terminals

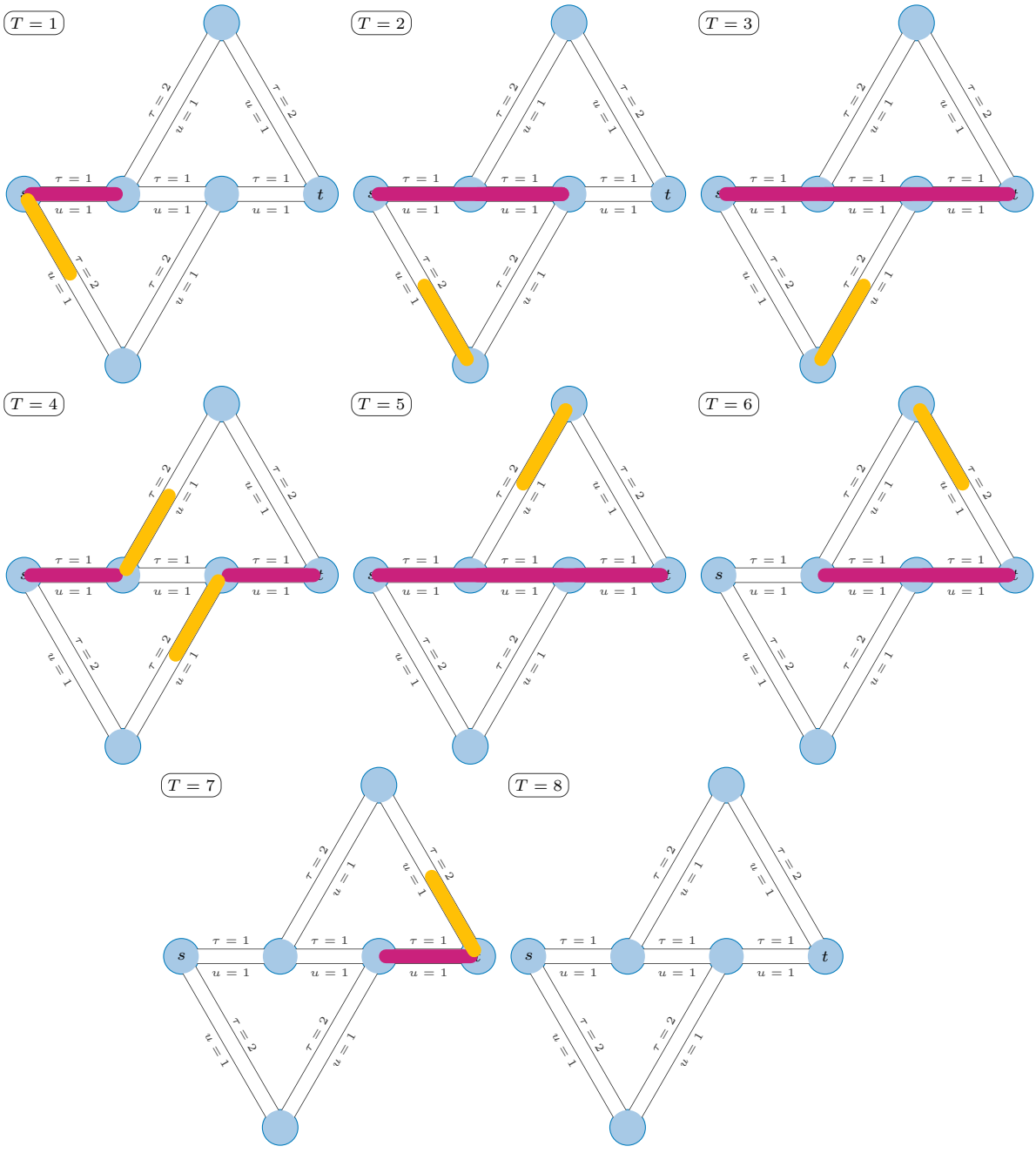
Task: Compute a transshipment over time f that fulfills all supplies and demands with the additional property that for each point in time as much flow as possible has reached the sink t.



(a) In the first iteration of the successive shortest path algorithm in the depicted dynamic network the magenta path is chosen.



(b) In the second iteration the yellow path in the residual network is chosen.



(c) The earliest arrival flow with time horizon T obtained from the results of the successive shortest path algorithm

Figure 3.2: An example for an earliest arrival flow with time horizon 8 obtained using the successive shortest path algorithm

It is not inherently clear that earliest arrival transshipments solving a given problem $(\mathcal{N}, b)_{\text{EAT}}$ do always exist and it turns out that in dynamic networks with multiple sinks, they don't. This was first observed by Fleischer [Fle01] and a simple counter example is given by Baumann and Skutella in [BS09] (see also Figure 3.3). However, in dynamic networks with only a single sink earliest arrival transshipments do always exists.

Networks with a Single Sink. The existence of earliest arrival transshipments in dynamic networks with a single sink was shown in 2002 by Richardson and Tardos [RT] who observed that Minieka's existence proof of earliest arrival flows based on static lexicographically maximal flows in the time-expanded network can be extended to networks with several sources and a single sink. In fact, an earliest arrival transshipment solving a given problem $(\mathcal{N}, b)_{\text{EAT}}$ in a single sink network $\mathcal{N}$ can be solved as in Algorithm 7. We only need to choose a sufficiently large time horizon T (as the minimal feasible time horizon T^* is not known) and the regarded time-expanded network $\mathcal{N}^T$ needs to implement the supplies and demands of the terminals. However, using time expansion to solve earliest arrival transshipment problems does neither lead to a polynomial time algorithm (which cannot be expected due the $\mathcal{NP}$-hardness result of Disser and Skutella [DS15]) nor to an algorithm which requires only polynomial space. Using the successive shortest path algorithm in the original network also does not lead to a suitable algorithm for the earliest arrival transshipment problem as it cannot be made sure that the flow our of the sources does not exceed their supplies.
In 2009 Baumann and Skutella [BS09] present the first algorithm for the earliest arrival transshipment problem that does not rely on explicit time expansion. They achieve an algorithm with a running time bounded by the size of the input *and the output*. The algorithm consists of two parts:
At first the earliest arrival pattern p^* corresponding to an instance $(\mathcal{N}, b)_{\text{EAT}}$ of the earliest arrival transshipment problem is computed. Afterwards $(\mathcal{N}, b)_{\text{EAT}}$ is reduced to a quickest transshipment problem $(\mathcal{N}', b')$ in an extended network $\mathcal{N}'$. While the dynamic network $\mathcal{N}'$ is not a result of time expansion, it is the result of the attachment of several new super-sinks, which might in the worst case be exponentially many. Thus, the algorithm of Baumann and Skutella [BS09], although it is *not* relying on time expansion, does in the worst case need an exponential amount of space.
Understanding the intuition behind the algorithm of Baumann and Skutella will be important to understand the results of this thesis. In particular, we heavily use the structure of the earliest arrival pattern p^* corresponding to a given problem $(\mathcal{N}, b)_{\text{EAT}}$. This is why we describe the pattern computation and the reduction to a quickest transshipment problem in more detail below.

Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single sink t. We start by explaining the intuition behind the pattern computation. Imagine, we start at time $\theta = 0$ to send as much flow as possible from the sources towards the sink t, i.e., we send flow such that the resulting pattern is $o^\theta(S^+)$. For some time we can send flow like this without violating any supplies on the sources, but eventually the supply of some sources will be exceeded. This is the point in time at which we have to stop to send flow with the highest possible rate. What we can do after the overall supply of a subset of sources $S' \subseteq S^+$ is fulfilled at some point in time θ', is to keep on sending flow at the highest possible rate from the remaining sources until the overall supply of the next subset of sources is fulfilled, and so on.
This is the main idea upon which the pattern computation of Baumann and Skutella relies. More precisely, in the first iteration of their algorithm Baumann and Skutella compute the maximal time horizon θ_1 such that the overall supply of *every subset* $S \subseteq S^+$ *of sources* is not violated when the amount of flow sent towards t until time θ_1 is $o^{\theta_1}(S^+)$ (i.e., as much flow as possible), whereas the sources in S send as little flow as possible with respect to this (which means that the sources in S have to send $o^{\theta_1}(S^+) - o^{\theta_1}(S^+ \setminus S)$, whereas the sources in $S^+ \setminus S$ have to send $o^{\theta_1}(S^+ \setminus S)$). Formally, in the first iteration of the pattern computation the maximal value $\theta_1 \geq 0$ such that

$$o^{\theta_1}(S^+) - o^{\theta_1}(S^+ \setminus S) \leq b(S) \text{ for all } S \subseteq S^+ \tag{3.2}$$

is computed. Define $g^\theta \colon 2^{S^+} \to \mathbb{R}$ by $g^\theta(S) \coloneqq o^\theta(S^+ \setminus S) - o^\theta(S^+) + b(S)$. Computing θ_1 thus amounts in finding the maximal value $\theta \geq 0$ such that

$$g^\theta(S) \geq 0 \text{ for all } S \subseteq S^+.$$

Since the function o^θ is submodular and the function $S \mapsto b(S^+ \setminus S) - o^\theta(S^+)$ is modular, the parametrized function g^θ is also submodular for every $\theta \geq 0$. Also $g^\theta(S)$ is monotonically decreasing in θ for a fixed set $X \subseteq S^+$. Hence, determining θ_1 is nothing but a parametrized submodular function minimization problem. We can evaluate g^θ in strongly polynomial time by two minimum-cost flow computations in the original network $\mathcal{N}$ and thus, we can determine θ_1 in strongly polynomial time by using the submodular function minimization algorithm of Lee et al. [LSW15] together with Meggido's parametric search framework. One main result of Baumann and Skutella [BS09] is that there is an inclusion-wise maximal subset $S_1 \subseteq S^+$ with

$$o^{\theta_1}(S^+) - o^{\theta_1}(S^+ \setminus S_1) = b(S_1).$$

Intuitively this means that there is a subset of sources S_1 that have sent exactly their overall supplies to t until time θ_1 when they send as little flow as possible while not violating the earliest arrival property. The set S_1, which is the maximal minimizer of g^{θ_1}, is also computed by the algorithm of Lee at al. during the submodular function minimization. The main result of Baumann and Skutella is that the earliest arrival pattern corresponding to $(\mathcal{N}, b)_{\text{EAT}}$ has the following recursive structure:

Theorem 3.16 (Structure of the Earliest Arrival Pattern, [BS09]).
Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, t)$ be a dynamic network with supplies and demands b on the terminals. Denote by θ_1 the maximal value such that all inequalities in (3.2) hold and denote by $S_1 \subsetneq S^+$ the maximal minimizer of the submodular function $o^{\theta_1}(S^+ \setminus S_1) - o^{\theta_1}(S^+) + b(S_1)$. Let p' be the earliest arrival pattern corresponding to $(\mathcal{N}', b')_{\text{EAT}}$ where $\mathcal{N}'$ is obtained out of $\mathcal{N}$ by removing the sources in $S^+ \setminus S_1$. Then,

$$p^*(\theta) = \begin{cases} o^\theta(S^+) & \text{if } \theta \leq \theta_1, \\ p'(\theta) + b(S_1) & \text{if } \theta > \theta_1. \end{cases}$$

The proof of Theorem 3.16 can be found in the thesis of Nadine Baumann [Bau07]. A straightforward corollary of this theorem is that the pattern p^* corresponding to $(\mathcal{N}, b)_{\text{EAT}}$ is a piecewise linear, monotonically increasing continuous function because o^θ is piecewise linear, convex and continuous. Note that p^* is *not* necessarily convex.

Corollary 3.17. *Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, t)$ be a dynamic network with supplies and demands b on the terminals. The earliest arrival pattern p^* corresponding to $(\mathcal{N}, b)_{\text{EAT}}$ is a piecewise linear, monotonically increasing and continuous function.*

According to Theorem 3.16 computing the earliest arrival pattern p^* corresponding to $(\mathcal{N}, b)_{\text{EAT}}$ amounts in computing subsets of sources $S_1, S_2, \ldots, S_k \subseteq S^+$ and times $\theta_1, \theta_2, \ldots, \theta_k$ such that we have

$$o^{\theta_i}\left(S^+ \setminus \bigcup_{j=1}^{i-1} S_j\right) - o^{\theta_i}\left(S^+ \setminus \left(\bigcup_{j=1}^{i-1} S_j \cup S\right)\right) \leq b(S) \text{ for all } S \subseteq S^+ \setminus (S_1 \cup S_2 \cup \ldots \cup S_{i-1}).$$

and

$$o^{\theta_i}\left(S^+ \setminus \bigcup_{j=1}^{i-1} S_j\right) - o^{\theta_i}\left(S^+ \setminus \left(\bigcup_{j=1}^{i} S_j \cup S_i\right)\right) = b(S_i),$$

together with computing the function $\theta \mapsto o^\theta(S^+ \setminus (S_1 \cup \ldots \cup S_{i-1}))$ on the interval $[\theta_{i-1}, \theta_i)$. Note that it is not obvious that $\theta_i < \theta_{i-1}$ for all $i \geq 0$. This is a property that needs to be proved (see [Bau07]). To compute the function $\theta \mapsto o^\theta(S^+ \setminus (S_1 \cup \ldots \cup S_{i-1}))$ in $[\theta_{i-1}, \theta_i)$ it suffices to compute all breakpoint of this piecewise linear function within this interval. The algorithm to compute these breakpoints is described in [BS09]. Overall, this yields Algorithm 9. For our purpose, only the computation of the sets $S_i \subseteq S^+$ and the times θ_i will be important.
In order to reduce the problem $(\mathcal{N}, b)_{\text{EAT}}$ to a quickest transshipment problem in an extended

Algorithm 9: Algorithm for the earliest arrival pattern, PATTERN$((\mathcal{N}, b)_{\text{EAT}})$

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, s, t)$ and a supply/demand function b on the terminals

Output : The earliest arrival pattern p^* corresponding to $(\mathcal{N}, b)_{\text{EAT}}$

$i \leftarrow 0$, $S_i \leftarrow \varnothing$, $\theta_i \leftarrow 0$

while $S_1 \cup \ldots \cup S_i \neq S^+$ **do**

  Compute the maximal value $\theta_{i+1} \geq 0$ such that

$$o^{\theta_{i+1}}\left(S^+ \setminus \left(\bigcup_{k=1}^{i} S_k\right)\right) - o^{\theta_{i+1}}\left(S^+ \setminus \left(\bigcup_{j=1}^{i} S_j \cup S\right)\right) \leq b(S) \text{ for all } S \subseteq S^+ \setminus \left(\bigcup_{k=1}^{i} S_i\right)$$

  Compute an inclusion-wise maximal set $S_{i+1} \subsetneq S^+ \setminus \left(\bigcup_{k=1}^{i} S_i\right)$ with

$$o^{\theta_{i+1}}\left(S^+ \setminus \left(\bigcup_{k=1}^{i} S_k\right)\right) - o^{\theta_{i+1}}\left(S^+ \setminus \left(\bigcup_{k=1}^{i+1} S_k\right)\right) = b(S_{i+1})$$

  Compute the function $\theta \mapsto o^{\theta}(S^+ \setminus (S_1 \cup \ldots \cup S_i))$ on the interval $[\theta_i, \theta_{i+1})$ and set

$$p^*(\theta) \coloneqq o^{\theta}(S^+ \setminus (S_1 \cup \ldots \cup S_i)) + b(S_i) \text{ for all } \theta \in [\theta_i, \theta_{i+1})$$

  $i \leftarrow i + 1$

end

return p^*

dynamic network, Baumann and Skutella attach one new super-sink to $\mathcal{N}$ for each breakpoint of the earliest arrival pattern p^*. Assume we are given p^* as a list of breakpoints (x_0, f_0), (x_1, f_1),..., (x_l, f_l), i.e.,

$$p(\theta) = \begin{cases} 0 & \text{if } \theta \leq x_0 \\ f_i + \frac{\theta - x_i}{x_{i+1} - x_i}(f_{i+1} - f_i) & \text{if } x_i \leq \theta \leq x_{i+1} \text{ for } 0 \leq i < k \\ f_l & \text{if } \theta \geq x_l. \end{cases}$$

The extended dynamic network $\mathcal{N}'$ is obtained by attaching a super-sink t_i to t for each $i \in \{1, \ldots, l\}$. For all $i \in \{1, \ldots, l\}$ we connect t_i with t by an arc (t, t_i) with transit time $x_l - x_i$ and capacity $(f_i - f_{i-1})(x_i - x_{i-1})$. The extended supply/demand function b' is the result of removing the demand of t and giving t_i the demand $-(f_i - f_{i-1})$ for all $i \in \{1, \ldots, l\}$. Baumann and Skutella now show that a quickest transshipment f' solving $(\mathcal{N}', b')$ can be turned into an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$. We can thus just solve $(\mathcal{N}', b')$ using the algorithm of Hoppe and Tardos [HT00] to solve $(\mathcal{N}, b)_{\text{EAT}}$. Note that Algorithm 10 is only of pseudo-polynomial running time as in the

Algorithm 10: Algorithm for the earliest arrival transshipment problem by Baumann and Skutella

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, s, t)$ and a supply/demand function b on the terminals

Output : An earliest arrival transshipment f

$p^* \leftarrow$ Earliest arrival pattern returned by Algorithm 9

$\mathcal{N}' \leftarrow$ Extended network constructed using the breakpoints of p^*

$b' \leftarrow$ Supply/demand function extended to the added super-sinks of $\mathcal{N}'$

$f' \leftarrow$ Quickest Transshipment solving $(\mathcal{N}', b')$ computed by the algorithm of Hoppe and Tardos

$f \leftarrow$ Restriction of f' to $\mathcal{N}$

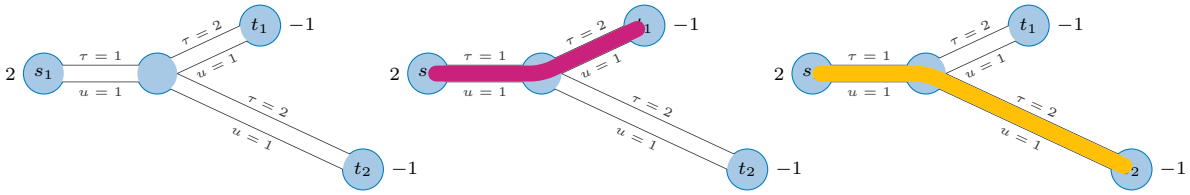
return f

worst case we add pseudo-polynomial many new sinks to the dynamic network $\mathcal{N}$ to obtain $\mathcal{N}'$.

Thus, the algorithm of Baumann and Skutella does require pseudo-polynomial space in the worst case.

Fact 3.18. *Using the algorithm of Baumann and Skutella, a given problem* $(\mathcal{N}, b)_{\text{EAT}}$ *in a dynamic network* $\mathcal{N}$ *with only a single sink* t *can be solved in a running time that is polynomial in the input size plus the number of breakpoints of the corresponding earliest arrival pattern* p^**. Moreover, their algorithm does in general not work in polynomial space.*

Networks with Multiple Sinks. We already stated that in a dynamic network with multiple sinks it can happen that no earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ does exist. The reason is quite simple: In a dynamic network with multiple sinks it might happen that flow on paths arriving at sink t at time θ blocks all paths arriving at another sink t' at the same time. If the earliest arrival property at the first sink t is maintained, all blocked flow units have to wait to reach t' and thus fail to satisfy the earliest arrival property. A simple example for a dynamic network with this property is depicted in Figure 3.3.



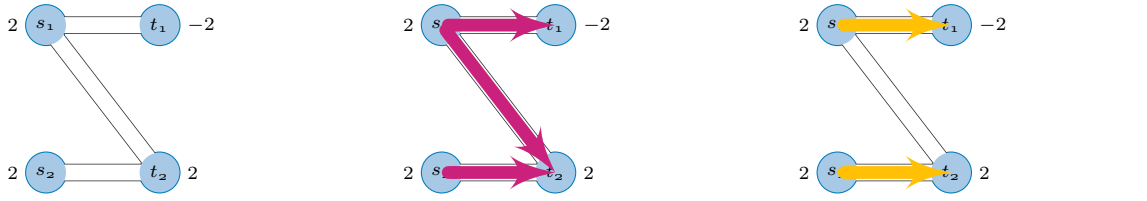
(a) An earliest arrival transshipment problem in a dynamic network with a single source with supply 2 and two sinks, each with demand -1.

(b) At time 3 at most one flow unit can have arrived at the sinks. This is achieved by sending one flow unit into the magenta path during the time interval $[0, 1)$.

(c) At time 4 at most two flow units can have arrived at the sinks. This is achieved by sending one flow unit into the yellow path during the time interval $[0, 1)$ and one flow unit into the magenta path during $[1, 2)$.

Figure 3.3: An example for an earliest arrival transshipment problem in a dynamic network with two sinks that does not have a solution: clearly, we cannot send flow such that one flow unit has arrived at the sinks at time 3 and two flow units have arrived at time 4 while respecting the demands.

That earliest arrival transshipments in networks with multiple sinks do not always exist was first observed by Fleischer [Fle01]. The example in Figure 3.3 is due to Baumann and Skutella [BS09]. Figure 3.3 also shows that even in dynamic networks with a single source earliest arrival transshipment do not always exists while the following example in Figure 3.4 shows that earliest arrival transshipments in networks with multiple sources *and* multiple sinks do not always exist even if all transit times are zero. In [SS14] Schmidt and Skutella give a characterization of dynamic



(a) An earliest arrival transshipment problem in a dynamic network with two sources and two sinks. All arcs are supposed to have zero transit time and capacity one.

(b) At time 1 at most three flow unit can have arrived at the sinks. This is achieved by sending one flow unit into the magenta paths during the time interval $[0, 1)$.

(c) At time 2 at most four flow unit can have arrived at the sinks. This is achieved by sending two flow units into each of the yellow path during the time interval $[0, 2)$.

Figure 3.4: An example for an earliest arrival transshipment problem in a dynamic network with all zero transit times that does not have a solution: clearly, we cannot send flow such that 3 flow units have arrived at the sinks at time 1 and 4 flow units have arrived at time 2 while respecting the demands.

network with $\tau \equiv \mathbf{0}$ that allow for earliest arrival transshipments for all choices of transit times

and supplies and demands. Their results in particular imply that in dynamic networks with a single source and all zero transit times earliest arrival transshipments do always exist. However, all their results are non-constructive, i.e., they do not give an algorithm to compute earliest arrival transshipments. Besides this, not much is known about earliest arrival transshipments in dynamic networks with multiple sinks: The complexity of deciding whether an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ does exist is still an open problem, and it is also not known how to come up with a suitable earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ in case of existence. We focus on solving these problems throughout this thesis.

3.2.3 Approximation of Earliest Arrival Transshipments

For dynamic network with a single sink it was shown by Disser and Skutella [DS15] that solving earliest arrival transshipment problems is $\mathcal{NP}$-hard. It is thus unlikely that algorithms with polynomial running time for solving earliest arrival transshipment or earliest arrival flow problems do exist. In light of this fact it makes sense to come up with polynomial time algorithms that produce flows over time that are not "too far off" from fulfilling the earliest arrival property for every point in time, i.e., that compute an **approximation** of an earliest arrival flow or transshipment.
For networks with multiple sinks, looking at approximations of earliest arrival transshipments makes even more sense, as in this case earliest arrival transshipment do not always exist and it is interesting to see whether flows over time that are "nearly" earliest arrival transshipments do exist and how to compute them efficiently in case of existence.
There are two variants of approximation for earliest arrival flows. The first variant relaxes the time, i.e., we are allowed to send the flow by a factor later than the optimal pattern requires. In the second variant the value of the flow is relaxed, i.e., we only have to send a given factor of the maximum flow at each given point in time.
Since earliest arrival transshipments or earliest arrival flows optimize multiple objectives at once, approximation algorithms for them should also approximate all of these objectives.

Definition 3.19 (α-time-approximation).
*Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and b a supply/demand function. Denote by p^* the corresponding earliest arrival pattern. A flow over time f is a **α-time-approximation** if at every point in time $\theta \in [0, T)$ we have*

$$|f|_\theta \geq p^*\left(\frac{\theta}{\alpha}\right).$$

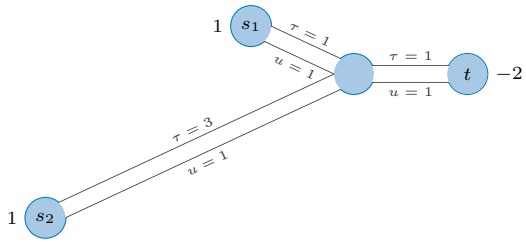
Time-approximation allows flow to be late. The notion of an α-time approximation was introduced by Baumann and Köhler [BK04]. They derived a 4-time-approximation for earliest arrival flows in dynamic networks with flow-dependent transit times. In this model, the transit time of an arc depends on the flow that is already on the arc. In this setting, earliest arrival flows do not always exist.
The notion of β-value-approximation was introduced by Groß et al. [Gro+12] for earliest arrival transshipments in dynamic networks with multiple sinks.

Definition 3.20 (β-value-approximation).
*Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and b a supply/demand function. Denote by p^* the corresponding earliest arrival pattern. A flow over time f is an **β-value-approximation** if at every point in time $\theta \in [0, T)$ we require*

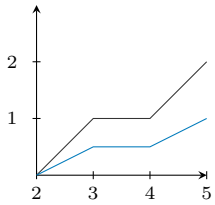
$$|f|_\theta \geq \frac{p^*(\theta)}{\beta},$$

i.e., at least a β-fraction of the maximum possible flow is sent until time θ.

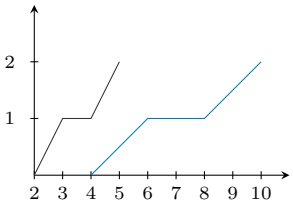
Both variants are visualized in Figure 3.5. These two models of approximation have already been used before without being named explicitly. For example, Hoppe and Tardos already gave a value approximative FPTAS for the earliest arrival flow problem, whereas Fleischer and Skutella [FS07]



(a) An example instance of an earliest arrival transshipment problem



(b) The black line is the earliest arrival pattern, while the blue line is the pattern of a 2-value approximation. Thus, a flow over time whose pattern is between the black and the blue line, is a 2-value approximation.



(c) The black line is the earliest arrival pattern, while the blue line is the pattern of a 2-time approximation. Thus, a flow over time whose pattern is between the black and the blue line, is a 2-time approximation

Figure 3.5: An illustration of both variants of approximation of earliest arrival transshipments.

describe a time-approximative FPTAS for earliest arrival transshipments using a condensed version of the time-expanded network. Groß et al. [Gro+12] showed that even in dynamic networks with multiple sinks and arbitrary transit times it is still possible to find approximations of earliest arrival transshipments. They present a 2-value approximation for earliest arrival transshipments together with a family of instances in which no value-approximation better than 2 is possible. Regarding time-approximation Groß et al. extend the technique of Baumann and Köhler [BK04] to achieve a 4-time-approximation of earliest arrival transshipments in dynamic networks with only a single source and multiple sinks. The best known lower bound for this case is 2. For dynamic networks with multiple sources and sinks they present a lower bound that is linear in the time horizon. An overview of these results is given in Table 3.2. In addition to these results Groß et al. also give an

Table 3.2: An overview of existence results for approximate earliest arrival transshipments

Approximation	single source single sink	multiple sources single sink	single source multiple sinks	multiple sources multiple sink
α**-time**	1 [Gal59; RT]		$[2, 4]$ [Gro+12]	$\Omega(T)$ [Gro+12]
β**-time**	1 [Gal59; RT]		2 [Gro+12]	

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.

3.2.4 Summary and Our Results

In this section we again shortly summarize the problems we presented in this section (see Table 3.3) and describe our contributions.

Table 3.3: Overview over all flow over time problems introduced in this section

Problem	Setting	Objective
Earliest Arrival Flow	single source, single sink with time horizon T	flow over time that is maximal at all times $\theta \in [0, T)$
Earliest Arrival Transshipment	multiple sources and sinks with supplies and demands	a quickest transshipment that is also maximal at all points in time

Earliest Arrival Transshipments in Networks with a Single Sink.

Previously best known algorithm for computing the earliest arrival pattern: The algorithm presented by Baumann and Skutella [BS09] that relies in the worst case on number of terminals many parametrized submodular function minimizations.

Our result: A variation of the algorithm of Baumann and Skutella (see Section 5.1). When computing the earliest arrival pattern for $(\mathcal{N}, b)_{\text{EAT}}$, we replace every parametrized submodular function minimization by at most $|S^+|^2$ many (non parametrized) submodular function minimizations, which results in a speed-up of a factor of at least $|S^+|$ (see Observation 3.9).

Previously best known exact algorithm: The algorithm of Baumann and Skutella [BS09] which requires in the worst case a pseudo-polynomial expansion of the original network.

Our results:

1. A *polynomial space* algorithm that computes an earliest arrival transshipment as convex combination of generalized temporally repeated flows (see Section 5.3.2).
2. A *polynomial space* algorithm that computes an integral earliest arrival transshipment by reducing an earliest arrival transshipment problem to a generalization of the lexicographically maximum flow over time problem (see Section 5.3.3).

Previously best known approximation algorithm: The time-approximative FPTAS of Fleischer and Skutella [FS07] that relies on a condensed version of the time-expanded network.

Our result: A new time-approximative FPTAS that does not require any form of time expansion (see Section 5.4.2).

Earliest Arrival Transshipments in Networks with a Multiple Sinks.

Previously best known algorithm for computing the earliest arrival pattern: Nothing was known about the structure of the earliest arrival pattern corresponding to earliest arrival transshipment problems in dynamic networks with multiple sinks.

Our results: We derive the structure of the earliest arrival pattern corresponding to earliest arrival transshipment problems $(\mathcal{N}, b)_{\text{EAT}}$ in dynamic networks with multiple sources and *only a single sink* and for the special case of tight problems in general dynamic networks (see Section 6.1).

Previous algorithmic or complexity results regarding earliest arrival transshipments in multiple sink networks: Until now, there did not exist any algorithmic results regarding earliest arrival transshipments in multiple sink networks: There is no algorithm known for checking whether a given earliest arrival transshipment has a solution and also none that computes an earliest arrival transshipment in case of existence. Furthermore, the complexity of deciding whether a given earliest arrival transshipment problem has a solution is unknown.

Our results: A polynomial space algorithm that checks whether a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with *only a single source* has a solution and computes a solution in case of existence as a convex combination of special generalized temporally repeated flows.

For tight problems, we also give such an algorithm for general dynamic networks (see Section 6.27). Further, we also show that it is $\mathcal{NP}$-hard to decide whether a given earliest arrival transshipment problem has a solution (see Section 6.4).

Results regarding approximation algorithms: It is known that each earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a single source allows for a 4-time approximation. The best known lower bound is 2 (see [Gro+12]).

Our results: We show that each earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a single source allows for a 2-time approximation. Hence, we present a tight result.

4 Solving Quickest Transshipment Problems

The quickest transshipment problem is a classical flow over time problem that captures an important aspect of evacuation planning: given a dynamic network with supplies on the sources and demands on the sinks, a quickest transshipment is a flow over time that fulfills all supplies and demands as quickly as possible. In a 1995 landmark paper Hoppe and Tardos describe the first strongly polynomial time algorithm solving the quickest transshipment problem. Their algorithm relies on repeatedly calling an oracle for parametric submodular function minimization. The first oracle call is needed to determine the minimal time T^* needed to fulfill all supplies and demands, while the subsequent parametric submodular function minimizations are necessary to come up with the quickest transshipment.
Our results in this chapter are twofold. In the first part we present a new algorithm to determine the minimal feasible time horizon for a given quickest transshipment problem for the special cases in which the underlying network has only a single source *or* a single sink. Instead of solving a parametric submodular function minimization problem to determine this time horizon, our algorithm only needs to solve at most number of terminals many submodular function minimization problems. This makes our approach considerably more efficient.
In the second part of this chapter we present a simpler and more efficient algorithm for the quickest transshipment problem. Our algorithm (i) relies on only one parametric submodular function minimization and, as a consequence, has considerably improved running time, (ii) uses not only the solution of a submodular function minimization but actually exploits the underlying algorithmic approach to determine a quickest transshipment as a convex combination of simple lex-max flows over time, and (iii) in this way determines a structurally easier solution in the form of a generalized temporally repeated flow.

Publication Remark: Some of the results from this chapter have been published in [SS17b].

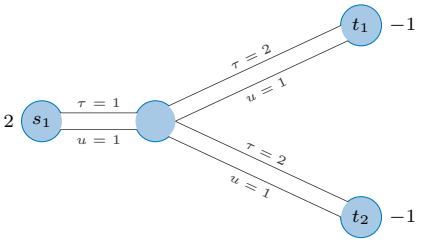
Contents

Quickest transshipments have many important applications: one straightforward example is that they can be used to devise good evacuation strategies, but also many logistic problems that require to send goods from suppliers to customers as quickly as possible essentially result in solving a quickest transshipment problem. Thus, it is relevant to come up with more efficient ways to solve quickest transshipment problems.
Recall, that the objective of a quickest transshipment problem $(\mathcal{N}, b)$ in a dynamic network $\mathcal{N}$ with supplies/demands b is to compute a flow over time f that sends flow from the sources to the sinks such that it meets all supplies and demands as quickly as possible. In 1995 Hoppe and Tardos [HT00] developed the first strongly polynomial time algorithm for this problem. Subsequently, more efficient algorithms for some special cases have been devised [Fle01; KKT06; KKT09; Mam+06], but no improvement has been obtained for the general quickest transshipment problem.

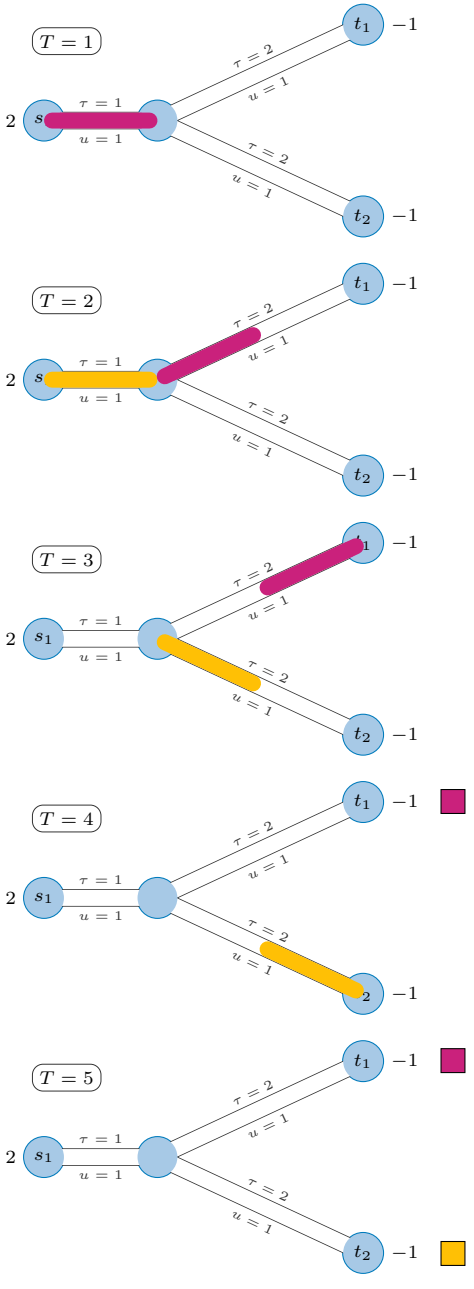
One strategy to solve a given quickest transshipment problem $(\mathcal{N}, b)$ is to at first determine the minimal feasible time horizon T^* such that $(\mathcal{N}, b, T^*)$ has a solution, and, in the second step, to come up with the actual transshipment. We will propose improvements for both parts, determining the minimal feasible time horizon, and computing a transshipment once a feasible time horizon is known. Altogether this leads to a more efficient algorithm for the quickest transshipment problem. In the following we shortly recap the state of the art regarding algorithms for solving quickest transshipment problems and describe our contributions in this context.

Determining the Minimal Feasible Time Horizon. In Section 3.1.1 we introduced the only known efficient way to determine the minimal feasible time horizon T^* for a quickest transshipment problem $(\mathcal{N}, b)$, which relies on coupling the submodular function minimization algorithm of Lee et al. [LSW15] with the parametric search framework of Megiddo [Meg79], see Fact 3.8. However, using the parametric search framework to determine T^* can in the worst case dramatically increase the running time of the algorithm of Lee et al. Moreover, Megiddo's search framework is hard to implement and thus not applicable in practice.
These two facts make it an interesting task to come up with new ways to determine T^* that do not rely on the framework of Megiddo. This is what Section 4.1 focuses on. We describe a new and faster strongly polynomial time algorithm to determine the minimal feasible time horizon T^* for a quickest transshipment problem in a dynamic network with only a single source *or* a single sink. In contrast to the classical approach, our algorithm does not need to call an oracle for parametric submodular function minimization, but only needs number of terminals many (non-parametric) submodular function minimizations in the worst case. This results in a huge running time improvement. The main feature that our algorithm exploits is that the submodular functions occurring in the context of the quickest transshipment problem in a dynamic network with only a single source or a single sink are related by the **strong map property** (see Section 2.3).

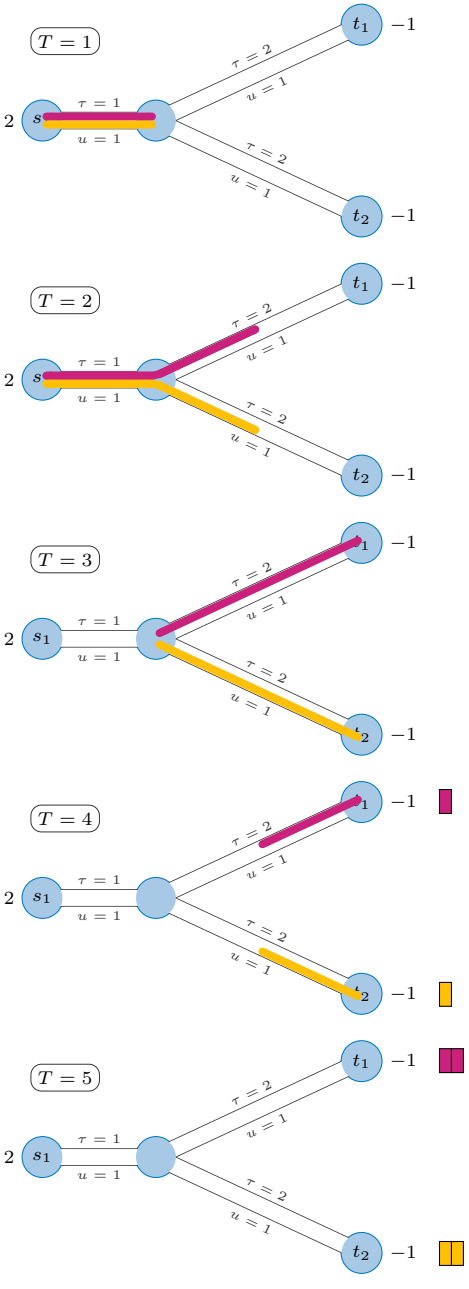
Computing Transshipments. So far, the only known strongly polynomial time algorithm for solving a given transshipment problem $(\mathcal{N}, b, T)$ is the algorithm of Hoppe and Tardos [HT00]. The key idea of this algorithms is to reduce $(\mathcal{N}, b, T)$ to a lexicographically maximum flow over time problem, which is then solved using Algorithm 6 (see Section 3.1.3). However, the reductions in the algorithm of Hoppe and Tardos rely on, in the worst case, $2 \cdot |S^+ \cup S^-|$ many parametric submodular function minimizations. The running time of a single submodular function minimization is already high and solving a parametric submodular function minimization problem is even more time consuming. Thus, it is our main objective to come up with a strongly polynomial time way to solve a given transshipment over time problem $(\mathcal{N}, b, T)$ that needs considerably fewer (parametric) submodular function minimizations. In Section 4.2 and Section 4.3 we explain how we achieve this goal. At first we derive a structural result about transshipments over time solving $(\mathcal{N}, b, T)$ – such flows over time can be obtained as a convex combination of lex-max flows over time with time horizon T (see Section 4.2). In Section 4.3 we then describe two algorithms to compute a suitable convex combination of lex-max flows over time with time horizon T. Both algorithms improve upon the algorithm of Hoppe and Tardos regarding the worst case running time. In Section 4.3.1 we at first describe the more straightforward, but less efficient of the two algorithms: An implementation of Carathéodory's theorem, which relies on a recent result about line search in base polytopes of submodular functions [GGJ17].
The key idea of the faster of the two algorithms (see Section 4.3.2) is to not only use the actual result of the submodular function minimization necessary to check whether T is a feasible time horizon, but to also exploit intermediate steps of this expensive computation. In particular, it uses the dual optimality certificate, which is a vector in the submodular function's base polytope, given as a convex combination of vertices of this polytope. It turns out that the vertices of the considered base polytope correspond to lex-max flows over time. Using this correspondence, we can obtain a convex combination of lex-max flows over time with time horizon T solving a given feasible transshipment over time problem $(\mathcal{N}, b, T)$.
Our algorithms do not only have a faster running time than the algorithm of Hoppe and Tardos, the solutions we achieve are also structurally somewhat simpler, as we compute a generalized temporally repeated flow that keeps sending flow at a constant rate along source-sink pairs as long as possible. On the negative side, these constant flow rates can be fractional (see Figure 4.1 for a simple example), while the solution found by Hoppe and Tardos' algorithm is always integral.



(a) A dynamic network $\mathcal{N}$



(b) The – up to symmetry – only integral solution



(c) The only generalized temporally repeated flow solution

Figure 4.1: An example showing that not every quickest transshipment problem allows for a generalized temporally repeated solution with integral flow rates.

4.1 An Improved Algorithm to Determine the Minimum Feasible Time Horizon

Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and b a supply/demand function on the terminals of $\mathcal{N}$. We consider the quickest transshipment problem $(\mathcal{N}, b)$. The only known strongly polynomial time method to determine the minimal feasible time horizon T^* for $(\mathcal{N}, b)$ is to apply submodular function minimization paired with Megiddo's parametric search framework. The fastest running time that can be obtained in this way is in

$$\mathcal{O}\big(\mathcal{C}_T(\mathrm{SFM}_{\mathrm{Lee}}) \cdot (k^3 \log k \cdot \mathrm{MCF}(n, m) + k^4 \log^{\mathcal{O}(1)} k)\big),$$

where $k := |S^+ \cup S^-|$ and $\mathcal{C}_T(\mathrm{SFM}_{\mathrm{Lee}})$ is the number of comparisons done in the algorithm of Lee et al. [LSW15] that depend on the parameter T. We already deduced that the worst case running time of $\mathrm{SFM}_{\mathrm{Lee}}$ increases by at least a factor of $k^3 \log k$ when pairing it with parametric search to determine T^* (see Observation 3.9). Hence, it makes sense to try to find a strongly polynomial time way to determine T^* without relying on the parametric search framework. In this section we describe such an algorithm for the special case of quickest transshipment problems in dynamic networks with only a single source s *or* a single sink t.
As a main ingredient for our algorithm we exploit the fact that the submodular functions occurring in the context of the quickest transshipment problem in networks with a single source or a single sink fulfill the strong map property. We begin this section by reviewing a central property of submodular functions related by a strong map and by showing that the submodular functions that are relevant when considering the quickest transshipment problem are in fact connected by the strong map property. In the second part of this section we describe our algorithms to determine the minimal feasible time horizon of a quickest transshipment problem.

4.1.1 The Strong Map Property

Let $f_1, f_2 \colon 2^S \to \mathbb{R}$ be two submodular functions over the same ground set S with $f_1 \leftarrow f_2$. A key property of submodular functions related by a strong map is the fact that the minimal minimizer of f_2 is a subset of the minimal minimizer of f_1. The same holds for the maximal minimizers. Note that the submodularity property implies that the minimizers of a submodular function are closed under union and intersection and hence form a distributive lattice. This implies that each submodular function has a *unique* maximal and a *unique* minimal minimizer.
This fact, which was first observed by Topkis [Top78] in 1978, is summarized in the following lemma.

Lemma 4.1 ([Top78]). *Let $f_1, f_2 \colon 2^S \to \mathbb{R}$ be two submodular functions over the same ground set S with $f_1 \leftarrow f_2$. Denote by X_1^{min} and X_1^{max} the minimal and maximal minimizer of f_1, respectively, while X_2^{min} and X_2^{max} are the minimal and maximal minimizer of f_2, respectively. We have $X_1^{max} \supseteq X_2^{max}$ and $X_1^{min} \supseteq X_2^{min}$.*

Proof. Using the strong map property and submodularity yields,

$$\begin{aligned} f_1(X_1^{\min} \cup X_2^{\min}) - f_1(X_1^{\min}) &\overset{\text{strong map}}{\leq} f_2(X_1^{\min} \cup X_2^{\min}) - f_2(X_1^{\min}) \\ &\overset{\text{subm.}}{\leq} f_2(X_2^{\min}) - f_2(X_1^{\min} \cap X_2^{\min}), \end{aligned}$$

which immediately implies that $X_2^{\min} \subseteq X_1^{\min}$. Similarly, $X_2^{\max} \subseteq X_1^{\max}$ can be shown. □

Next, we consider the submodular functions that are relevant in the context of quickest transshipment problems: For this purpose let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network. Recall, that for a fixed time horizon $\theta \geq 0$ the set function $o^\theta \colon 2^{S^+ \cup S^-} \to \mathbb{R}$ is defined by setting $o^\theta(X)$ to be the maximal amount of flow that can be sent from the sources in $X \cap S^+$ to the sinks in $S^- \setminus X$ until time θ in the dynamic network $\mathcal{N}$ for every $X \subseteq S^+ \cup S^-$. The function o^θ is submodular

for every time horizon $\theta \geq 0$, as we showed in Theorem 3.6. For a fixed time horizon $\theta \geq 0$ we define $o_S^\theta \colon 2^S \to \mathbb{R}$ to be the **restriction** of o^θ to $S \subseteq S^+$, that is

$$o_S^\theta(X) \coloneqq o^\theta(X) \text{ for all } X \subseteq S.$$

Clearly, the function o_S^θ also is submodular for every choice of time horizon $\theta \geq 0$ and subset $S \subseteq S^+$. This submodular function is particularly important for us when determining the minimal feasible time horizon for quickest transshipment problems in networks *with only a single sink*. It was already shown by Baumann and Skutella [BS09] that the functions $o_{S^+}^{\theta_1}$ and $o_{S^+}^{\theta_2}$ are related by the strong map property for $\theta_1 \leq \theta_2$. This property is directly inherited by $o_S^{\theta_1}$ and $o_S^{\theta_2}$ for all $S \subseteq S^+$.

Lemma 4.2 ([BS09]). *Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network. For $A \subseteq B \subseteq S \subseteq S^+$ and $\theta_1 \leq \theta_2$ we have*

$$o_S^{\theta_1}(B) - o_S^{\theta_1}(A) \leq o_S^{\theta_2}(B) - o_S^{\theta_2}(A),$$

i.e. $o_S^{\theta_1}$ and $o_S^{\theta_2}$ are connected by a strong map.

When considering networks with only a single source, a different, somehow symmetric, submodular function becomes important. For a fixed subset of sinks $T \subseteq S^-$ and a fixed time horizon $\theta \geq 0$ we define the set function $g_T^\theta \colon 2^T \to \mathbb{R}$ by

$$g_T^\theta(X) \coloneqq o^\theta(S^+ \cup X) \text{ for all } X \subseteq T.$$

The following observation is straightforward:

Observation 4.3. *Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network. The set function g_T^θ is submodular for every time horizon $\theta \geq 0$ and $T \subseteq S^-$.*

The most important result from this section is the fact that for $\theta_1 \leq \theta_2$ and any $T \subseteq S^-$ the submodular functions $g_T^{\theta_1}$ and $g_T^{\theta_2}$ are related by a **reversed strong map property**:

Lemma 4.4. *Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network. For $X \subseteq Y \subseteq T \subseteq S^-$ and $\theta_1 \leq \theta_2$ we have*

$$g_T^{\theta_2}(Y) - g_T^{\theta_2}(X) \leq g_T^{\theta_1}(Y) - g_T^{\theta_1}(X).$$

Proof. Consider an extended network $\overline{\mathcal{N}}$ with a super-source s that is connected to all sources in S^+ by arcs with infinite capacity and transit time $\theta_2 - \theta_1$ (see Figure 4.2). Thus, flow originating from the sources in S^+ that reaches the sinks until time θ_1 can also be started from the super-source s and reach the sinks until time θ_2. We define the set function $\overline{o}^\theta \colon 2^{\{s\} \cup S^+ \cup T} \to \mathbb{R}$ by setting $\overline{o}^\theta(X)$ to be the maximal amount of flow that can be sent from the sources in $X \cap (\{s\} \cup S^+)$ to the sinks in $S^- \setminus X$ until time $\theta \geq 0$ for all $X \subseteq \{s\} \cup S^+ \cup T$. By construction of $\overline{\mathcal{N}}$, the following equations are valid for all $Z \subseteq T$:

$$\overline{o}^{\theta_2}(\{s\} \cup S^+ \cup Z) = o^{\theta_2}(S^+ \cup Z), \tag{4.1}$$

and

$$o^{\theta_1}(S^+ \cup Z) = \overline{o}^{\theta_2}(\{s\} \cup Z). \tag{4.2}$$

By submodularity, we also have

$$\overline{o}^{\theta_2}(\{s\} \cup S^+ \cup X) + \overline{o}^{\theta_2}(\{s\} \cup Y) \geq \overline{o}^{\theta_2}(\{s\} \cup S^+ \cup Y) + \overline{o}^{\theta_2}(\{s\} \cup X). \tag{4.3}$$

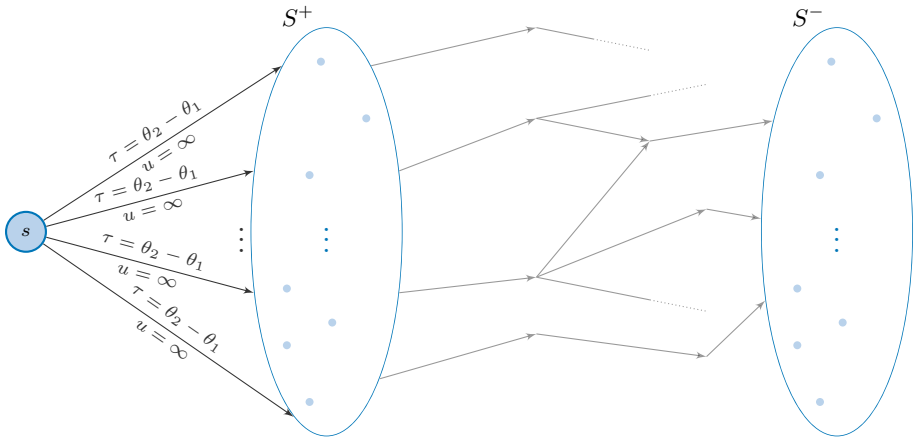


Figure 4.2: The extended dynamic network $\overline{\mathcal{N}}$

Overall,

$$\begin{aligned} g_T^{\theta_1}(X) - g_T^{\theta_1}(Y) &\overset{\text{Def.}}{=} o^{\theta_1}(S^+ \cup X) - o^{\theta_1}(S^+ \cup Y) \\ &\overset{(4.2)}{=} \overline{o}^{\theta_2}(\{s\} \cup X) - \overline{o}^{\theta_2}(\{s\} \cup Y) \\ &\overset{(4.3)}{\leq} \overline{o}^{\theta_2}(\{s\} \cup S^+ \cup X) - \overline{o}^{\theta_2}(\{s\} \cup S^+ \cup Y) \\ &\overset{(4.1)}{=} o^{\theta_2}(S^+ \cup X) - o^{\theta_2}(S^+ \cup Y) \\ &= g_T^{\theta_2}(X) - g_T^{\theta_2}(Y). \end{aligned}$$

□

In Lemma 4.4 we have shown that for every $T \subseteq S^-$ the submodular functions g_T^θ parametrized by θ are related by a reversed strong map property with growing θ. Thus, the complement functions of g_θ^T are again submodular functions that are related by the regular strong map property. For every $\theta \geq 0$ we define this complementary set function $\overline{g}_T^\theta \colon 2^T \to \mathbb{R}$ by

$$\overline{g}_T^\theta(X) = o^\theta(S^+ \cup S^- \setminus X) \text{ for all } X \subseteq T.$$

Corollary 4.5. *Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network. The set function $\overline{g}_T^\theta$ is submodular for every $\theta \geq 0$ and $T \subseteq S^-$, and for every $X \subseteq Y \subseteq T$ and $\theta_1 \leq \theta_2$ we have*

$$\overline{g}_T^{\theta_1}(Y) - \overline{g}_T^{\theta_1}(X) \leq \overline{g}_T^{\theta_2}(Y) - \overline{g}_T^{\theta_2}(X).$$

4.1.2 Computing the Minimum Feasible Time Horizon

We are now ready to describe our new algorithms that determine the minimal feasible time horizon for a given quickest transshipment problem $(\mathcal{N}, b)$ in a network with a single source s *or* a single sink t. We start by considering the case of dynamic networks with only a single sink t but a set of potentially multiple sourcesS^+.

Dynamic Networks With a Single Sink. During this paragraph $\mathcal{N} = (D = (V, A), u, \tau, S^+, t)$ is a dynamic network with a single sink t, supplies/demands b on the terminals, and $(\mathcal{N}, b)$ is the corresponding quickest transshipment problem. To find the minimal feasible time horizon T^* for $(\mathcal{N}, b)$ we exploit the fact that the submodular functions o_S^θ parametrized by θ form a strong map sequence with growing θ for any $S \subseteq S^+$. Recall, that the feasibility criterion of Klinz (see

Theorem 3.5) implies that T^* is the minimal feasible time horizon for $(\mathcal{N}, b)$ if and only if it is the smallest time horizon with the property

$$o^{T^*}(S) \geq b(S) \text{ for all } S \subseteq S^+.$$

The following observation is the key element for our algorithm.

Observation 4.6. *Let $S \subseteq S^+$, $\theta \geq 0$, and $S^{\min}$ the minimal minimizer of o_S^θ. Then, for each $\theta' > \theta$, the minimal minimizer of $o_{S^{\min}}^{\theta'}$ also is the minimal minimizer of $o_S^{\theta'}$.*

Proof. The proof relies on the strong map property: Lemma 4.2 implies that $o_S^\theta \leftarrow o_S^{\theta'}$ and hence, Lemma 4.1 yields that the minimal minimizer of $o_S^{\theta'}$ is a subset of $S^{\min}$. Thus, instead of determining the minimal minimizer of $o_S^{\theta'}$, we can equivalently also determine the minimal minimizer of $o_{S^{\min}}^{\theta'}$. □

The main idea of our algorithm, which is formally given in Algorithm 11, is to successively create a chain of subsets of S^+,

$$S^+ = S_0 \supsetneq S_1 \supsetneq S_2 \supsetneq \ldots \supsetneq S_k$$

and times

$$0 = \theta_0 < \theta_1 < \theta_2 < \ldots < \theta_k = T^*,$$

such that the set S_i, which is computed in iteration i of our algorithm, is the minimal minimizer of $o_{S_{i-1}}^{\theta_i} - b$ for all $i \in \{1, \ldots, k\}$.
In order to make sure that in each iteration i the computed set S_i is a real subset of S_{i-1}, we first determine θ_i to be the minimal value with $o_{S_{i-1}}^{\theta_i}(S_{i-1}) - b(S_{i-1}) = 0$. After that we compute a minimal minimizer S_i of $o_{S_{i-1}}^{\theta_i} - b$. Because we also have $o_{S_{i-1}}^{\theta_i}(\varnothing) - b(\varnothing) = 0$, the set S_{i-1} cannot be a minimal minimizer of $o_{S_{i-1}}^{\theta_i} - b$ and hence $S_i \subsetneq S_{i-1}$. Thus, our algorithm terminates after at most $|S^+|$ iterations. We will show that we have $\theta_{i-1} < \theta_i$ for all $i \in \{1, \ldots, k\}$. Thus, we can apply Observation 4.6, which essentially implies with the feasibility criterion of Klinz (see Theorem 3.5) that Algorithm 11 in fact computes a minimal feasible time horizon T^* for $(\mathcal{N}, b)$ and a minimal minimizer $S^{\min}$ of $o^{T^*} - b$.

Algorithm 11: Algorithm to determine the minimal feasible time horizon T^* for a quickest transshipment problem $(\mathcal{N}, b)$ in a dynamic network with a single sink t

```
Input  : A dynamic network N = (D = (V, A), u, τ, S⁺, t) and a supply/demand function b
Output : The minimal feasible time horizon T* for the quickest transshipment problem (N, b)
1 i ← 0, S_i ← S⁺, θ_i ← 0
2 while S_i ≠ ∅ do
3 |   θ_{i+1} ← Minimal value with o_{S_i}^{θ_{i+1}}(S_i) − b(S_i) = 0
4 |   S_{i+1} ← Minimal minimizer of o_{S_i}^{θ_{i+1}} − b
5 |   i ← i + 1
6 end
7 return T* := θ_i, S* := S_i
```

Before we prove the correctness of Algorithm 11, we look at a small example that illustrates how the algorithm proceeds.

Example 4.7. *We consider the quickest transshipment problem $(\mathcal{N}, b)$ illustrated in Figure 4.3. The algorithm starts with $i = 0$, $S_i = S^+$ and $\theta_i = 0$. In the first iteration the algorithm determines the minimal value θ_1 such that $o_{S^+}^{\theta_1}(S^+) - b(S^+) = 0$. In this example we obtain $\theta_1 = 4$, as can be seen from Figure 4.4a. For the newly determined θ_1 the algorithm next computes a minimal minimizer of $o_{S^+}^{\theta_1} - b$. As $o_{S^+}^{\theta_1}(S^+) - b(S^+) = o_{S^+}^{\theta_1}(\varnothing) - b(\varnothing) = 0$ and $o_{\theta_1}^{S^+}(\{s_1\}) - b(\{s_1\}) = 0$ (see Figure 4.4a), we get that $S_2 = \{s_2\}$, because $o_{\theta_1}^{S^+}(\{s_2\}) - b(\{s_2\}) = -1$ (see Figure 4.4b). This finishes the computations for $i = 0$. In the next iteration, i.e., for $i = 1$, at first the minimal time θ_2 with $o_{\{s_2\}}^{\theta_2}(s_2) - b(s_2) = 0$ is determined. According to Figure 4.4b, we get $\theta_2 = 5$. The minimal minimizer of $o_{\{s_2\}}^{\theta_2} - b$ is now obviously the empty set $\varnothing$. Thus, the algorithm terminates*

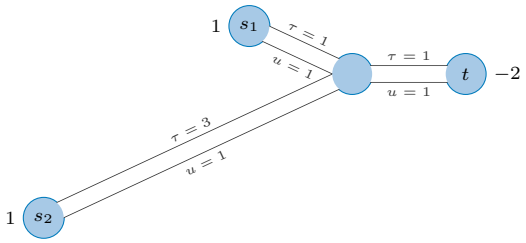
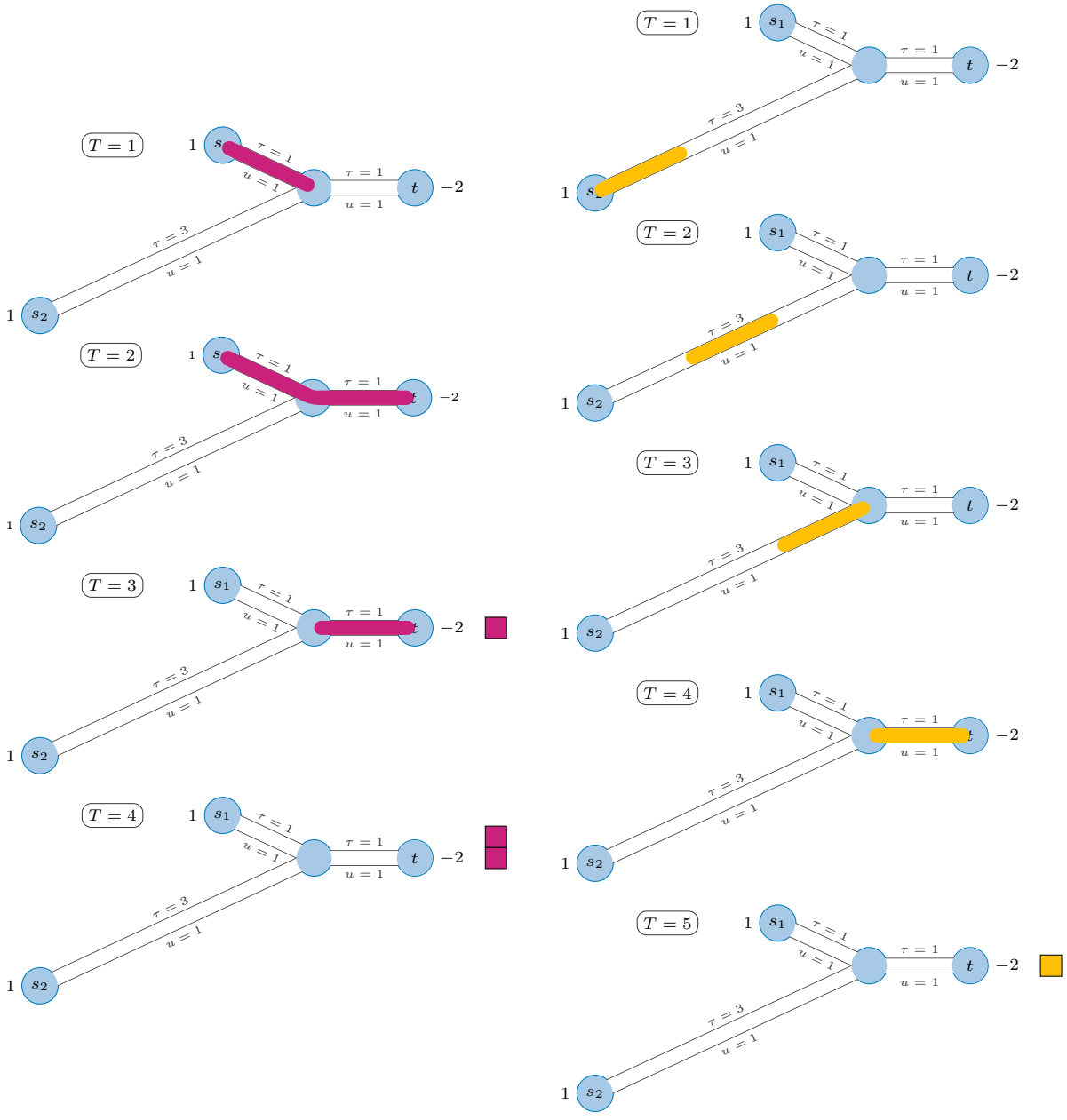


Figure 4.3: A dynamic network $\mathcal{N}$ with supplies and demands b as indicated



(a) Until time $T = 4$ the source s_1 can send at most two flow units to the sink, that is $o^4(S^+) = o^4(\{s_1\}) = 0$.

(b) Until time $T = 4$, no flow unit originating from s_2 can reach the sink. It takes at least until time $T = 5$ until one flow unit has traveled from s_2 to the sink.

Figure 4.4: Illustration of the first iteration of Algorithm 11

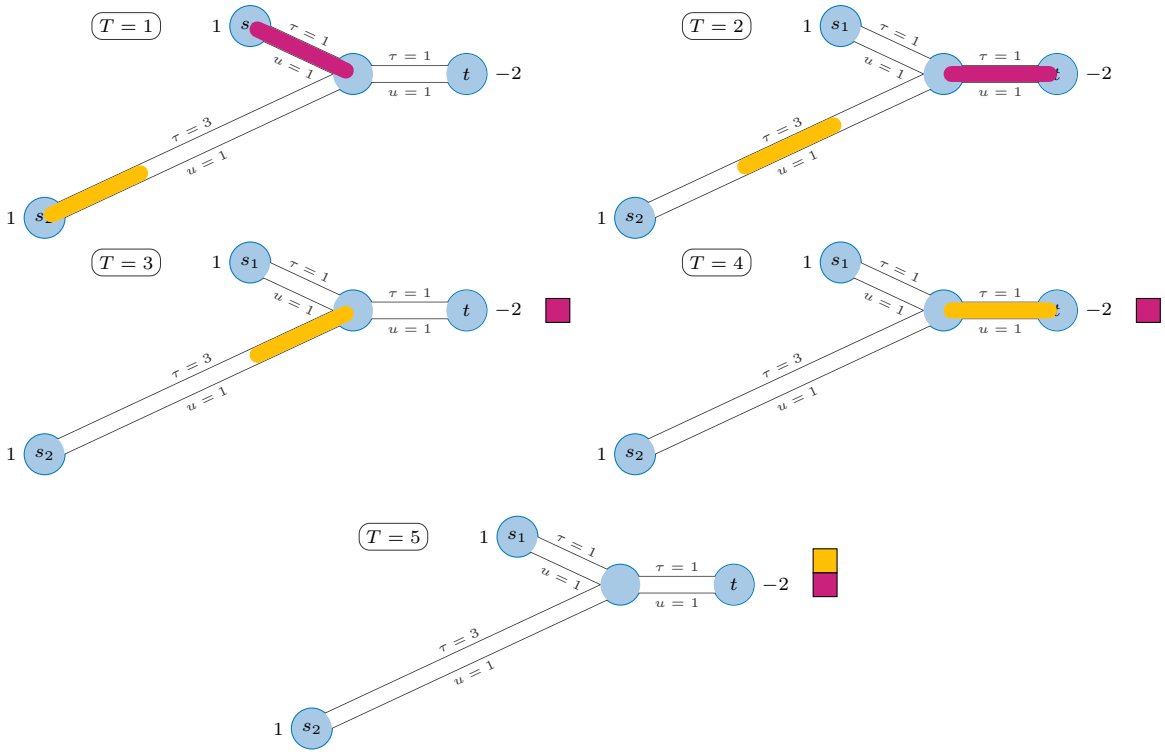


Figure 4.5: A flow over time with time horizon $T = 5$ solving the quickest transshipment problem from Figure 4.4

and returns $T^* = 5$ as minimal feasible time horizon, which is the correct time horizon as can be seen in Figure 4.5.

Theorem 4.8 (Correctness of Algorithm 11).

Let $(\mathcal{N}, b)$ be a quickest transshipment problem in a network with only a single sink t. Algorithm 11 determines the minimal feasible time horizon T^* for $(\mathcal{N}, b)$ and an inclusion-wise minimal subset $S^* \subseteq S^+$ with $o^{T^*}(S^*) = b(S^*)$. In the worst case it requires $|S^+|$ iterations.

Proof. We start by showing that in each iteration $i \geq 0$ the computed set S_{i+1} is a real subset of S_i, i.e., $S_{i+1} \subsetneq S_i$ (by construction S_{i+1} is contained in S_i).
For this purpose, assume that $S_{i+1} = S_i$. We have $S_i \neq \varnothing$ as otherwise the algorithm would have terminated after iteration i. This means that the minimal minimizer of $o_{S_i}^{\theta_{i+1}} - b$ is equal to S_i, and the minimal value of $o_{S_i}^{\theta_{i+1}} - b$ is zero by the definition of θ_{i+1}. However, we also have $o_{S_i}^{\theta_{i+1}}(\varnothing) = 0$. Thus, S_{i+1} is *not* a minimal minimizer of $o_{S_i}^{\theta_{i+1}} - b$, contradiction. This implies $S_{i+1} \subsetneq S_i$ for all $i \geq 0$. Thus, the algorithm terminates after at most $|S^+|$ iterations.
As a second step we show that $\theta_{i+1} > \theta_i$ for all $i \geq 0$. Assume that there exists an $i \geq 0$ with $\theta_{i+1} \leq \theta_i$. We now use the strong map property, and the fact that $o_{S_i}^{\theta_{i+1}}(S_i) = b(S_i)$, to achieve

$$\begin{aligned} o_{S_i}^{\theta_i}(S_{i+1}) &\overset{\text{strong map}}{\leq} o_{S_i}^{\theta_i}(S_i) + o_{S_i}^{\theta_{i+1}}(S_{i+1}) - o_{S_i}^{\theta_{i+1}}(S_i) \\ &= o_{S_i}^{\theta_i}(S_i) + o_{S_i}^{\theta_{i+1}}(S_{i+1}) - b(S_i). \end{aligned}$$

As the function $o_{S_i}^{\theta}(S)$ monotonically grows in θ for every fixed subset S, we have $o_{S_i}^{\theta_{i+1}}(S_{i+1}) \leq o_{S_i}^{\theta_i}(S_{i+1})$. It also holds that $o_{S_i}^{\theta_i}(S_i) - b(S_i) < 0$ as equality would imply that $S_i = \varnothing$. Combining these two facts implies

$$o_{S_i}^{\theta_i}(S_{i+1}) > o_{S_i}^{\theta_i}(S_i) + o_{S_i}^{\theta_{i+1}}(S_{i+1}) - b(S_i),$$

contradiction. It remains to be shown that the returned time is in fact the minimal feasible time horizon of the quickest transshipment problem $(\mathcal{N}, b)$. However, this follows from Observation 4.6, the feasibility criterion of Klinz (Theorem 3.5) and the fact that in each iteration of Algorithm 11 we compute the *minimal* value θ_{i+1} such that $o^{S_i}_{\theta_{i+1}}(S_i) - b(S_i) = 0$. □

The running time of Algorithm 11 now follows easily.

Corollary 4.9. *Using Algorithm 11, the minimal feasible time horizon for a quickest transshipment problem $(\mathcal{N}, b)$ in a dynamic network with only a single sink t can be determined using at most $|S^+|$ submodular function minimizations, that is, in an overall worst case running time of*

$$\mathcal{O}\big(k^4 \log k \cdot \mathrm{MCF}(n, m) + k^5 \log^{\mathcal{O}(1)} k\big) \text{ with } k = |S^+|,$$

or in the same asymptotic running time as the submodular function minimization algorithm of Orlin using the algorithm of Nagano [Nag07].

Proof. To determine the minimal time horizon θ_i with $o^{\theta_i}_{S^+}(S_i) - b(S_i) = 0$ in iteration i of Algorithm 11 we need to solve a quickest flow problem in a network with n vertices. Using the algorithm of Saho and Shigeno [SS17a] (see Fact 3.2) the value θ_i can thus be determined in running time $\mathcal{O}(nm^2(\log n)^2)$. The minimal minimizer of $o^{\theta_{i+1}}_{S_i} - b$ can be computed by one submodular function minimization for example with the algorithm of Lee et al. [LSW15]. This implies the first running time stated in the corollary.
Since all submodular functions minimized in the course of Algorithm 11 are related by the strong map property (when we extend them to be submodular functions over the same ground set), Fact 2.6 implies that all submodular function minimization done during the course of the algorithm can be done in the same asymptotic running time as the algorithm of Orlin using the algorithm of Nagano. □

Dynamic Networks With a Single Source. To describe the algorithm for determining the minimal feasible time horizon of a quickest transshipment problem $(\mathcal{N}, b)$ in a dynamic network with a single source s, the submodular functions $\overline{g}^{\theta}_{T}$ for $\theta \geq 0$ and $T \subseteq S^-$, which we defined in the previous section, are central. Recall that

$$\overline{g}^{\theta}_{T}(X) = o^{\theta}(\{s\} \cup S^- \setminus X) \text{ for all } X \subseteq T.$$

By Corollary 4.5 these functions are related by the strong map property for growing values of θ, and determining the minimal time horizon T^* with the property that $\overline{g}^{T^*}_{S^-}(X) + b(X) \geq 0$ for all $X \subseteq S^-$ is – by the feasibility criterion of Klinz [Kli] – equivalent to determining the minimal feasible time horizon for $(\mathcal{N}, b)$. Thus, we can proceed completely similar to compute the minimal feasible time horizon T^* as in Algorithm 11, which yields Algorithm 12. The correctness of Algorithm 12 also follows similarly as the correctness of Algorithm 11 and the running time we achieve is also the same as the running time of Algorithm 11:

Algorithm 12: Algorithm to determine the minimal feasible time horizon T^* for a quickest transshipment problem $(\mathcal{N}, b)$ in a dynamic network with a single source s

```
Input  : A dynamic network N = (D = (V, A), u, τ, S⁺, t) and a supply/demand function b
Output : The minimal feasible time horizon T* for the quickest transshipment problem (N, b)
1 i ← 0, T_i ← S⁻, θ_i ← 0
2 while T_i ≠ ∅ do
3 |   θ_{i+1} ← Minimal value with ḡ^{θ_i}_{T_i}(T_i) + b(T_i) = 0
4 |   T_{i+1} ← Minimal minimizer of ḡ^{θ_{i+1}}_{T_i} + b
5 |   i ← i + 1
6 end
7 return T* = θ_i, S* = S⁻ \ T_i
```

Corollary 4.10. *Using Algorithm 12 the minimal feasible time horizon of a quickest transshipment problem $(\mathcal{N}, b)$ in a dynamic network with a single sink t can be determined using at most $|S^-|$ submodular function minimization, that is in an overall worst case running time of*

$$\mathcal{O}\big(k^4 \log k \cdot \mathrm{MCF}(n,m) + k^5 \log^{\mathcal{O}(1)} k\big) \text{ with } k = |S^-|,$$

or in the same asymptotic running time of the submodular function minimization algorithm of Orlin using the algorithm of Nagano [Nag07].

4.1.3 Summary, Conclusions and Open Questions

Summarizing, we obtain algorithms for determining the minimal feasible time horizon in dynamic networks with only a single source *or* only a single sink that, instead of needing one parametric submodular function minimization, only requires k (for $k = |S^+|$ or $k = |S^-|$, respectively) non-parametric submodular function minimizations. That is, by Observation 3.9, we improve the worst case running time required to determine the minimal feasible time horizon of a quickest transshipment problem by at least a factor of $k^2 \log k$ in the two considered special cases. For quickest transshipment problems in general networks one could consider an algorithm similar to Algorithm 11 or Algorithm 12 and it is an easy observation that also in general networks such an algorithm terminates and returns the minimal feasible time horizon of a given quickest transshipment problem. It is however an open problem whether this algorithm terminates in weakly or even strongly polynomial running time. This is due to the fact that in general dynamic networks the considered parametric submodular function does not fulfill the strong map property. Overall, it remains an open question to find a general way to determine the minimal feasible time horizon for quickest transshipment problems without making use of Megiddo's parametric search.

4.2 Structure of Quickest Transshipments

We assume throughout this section that $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ is a dynamic network with a supply/demand function $b\colon S^+ \cup S^- \to \mathbb{Z}$ and that $T \geq 0$ is a time horizon such that the dynamic transshipment problem $(\mathcal{N}, b, T)$ is feasible, i.e., there exists a flow over time f in $\mathcal{N}$ with time horizon T fulfilling all supplies and demands.
The main result of this section is the following structural result, which states that a given feasible transshipment problem $(\mathcal{N}, b, T)$ can be solved by a convex combination of lexicographically maximum flows over time with time horizon T.

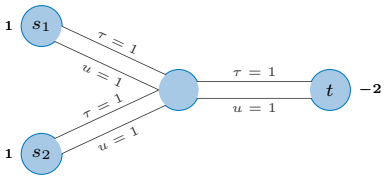
Theorem 4.11 (Structure of Transshipments Over Time).
Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network with a supply/demand function $b\colon S^+ \cup S^- \to \mathbb{Z}$ on the terminals of $\mathcal{N}$ and $T \geq 0$ a time horizon such that $(\mathcal{N}, b, T)$ is a feasible transshipment over time problem.
Then, a flow over time f solving $(\mathcal{N}, b, T)$ can be achieved as a convex combination of lexicographically maximum flows over time with time horizon T.
More precisely, there are $d \leq |S^+ \cup S^-|$ total orders $\prec_1$, $\prec_2$, …, $\prec_d$ on $S^+ \cup S^-$ and coefficients $\lambda_1, \lambda_2, \ldots, \lambda_d \geq 0$ with $\sum_{i=1}^{d} \lambda_i = 1$ such that

$$f = \lambda_1 f_1 + \lambda_2 f_2 + \ldots + \lambda_d f_d$$

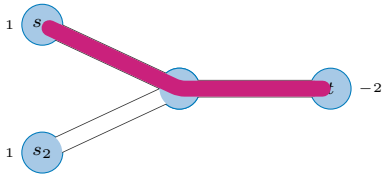
solves the transshipment over time problem $(\mathcal{N}, b, T)$. Here, $f_1, f_2, \ldots, f_d$ are lexicographically maximum flows over time with time horizon T with respect to $\prec_1, \prec_2, \ldots, \prec_d$, respectively.
*If T is the **minimal** feasible time horizon for $(\mathcal{N}, b)$, then it even holds that $d \leq |S^+ \cup S^-| - 1$.*

A small example illustrating the statement of Theorem 4.11 can be found in Figure 4.6.

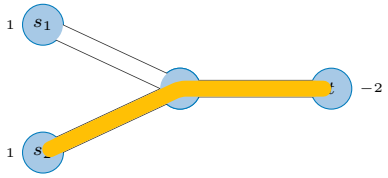
Before we get to the proof of Theorem 4.11, we need to recap some definitions. Recall, that $o^T(X)$, for $X \subseteq S^+ \cup S^-$, is the maximum amount of flow that can be sent from the sources in $X \cap S^+$ to the sinks in $S^- \setminus X$ until time T, disregarding the supplies and demands, and that o^T



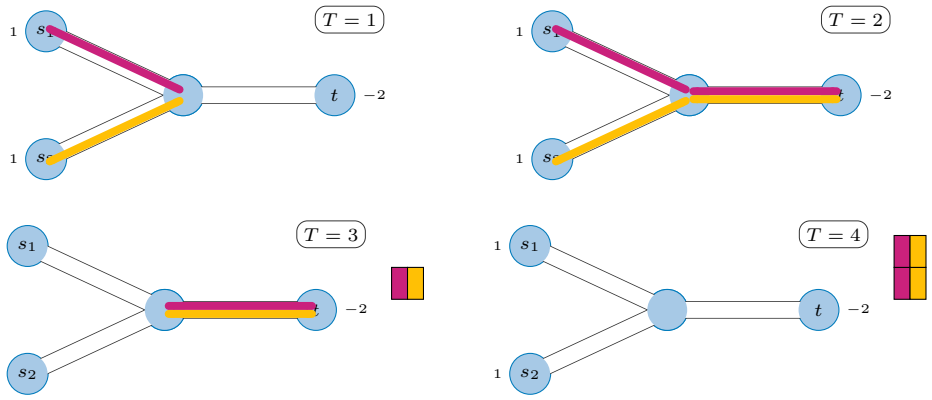
(a) A dynamic network $\mathcal{N}$ with supplies and demands b as indicated



(b) The lex-max flow over time f_1 with respect to the order $s_1 \prec_1 s_2 \prec_1 t$ with time horizon $T = 4$ sends flow into the indicated path with a rate of one during $[0, 2)$.



(c) The lex-max flow over time f_2 with respect to the order $s_2 \prec_1 s_1 \prec_1 t$ with time horizon $T = 4$ sends flow into the indicated path with a rate of one during $[0, 2)$.



(d) The quickest transshipment problem $(\mathcal{N}, b)$ can be solved by $\frac{1}{2}f_1 + \frac{1}{2}f_2$.

Figure 4.6: An example illustrating the statement of Theorem 4.11

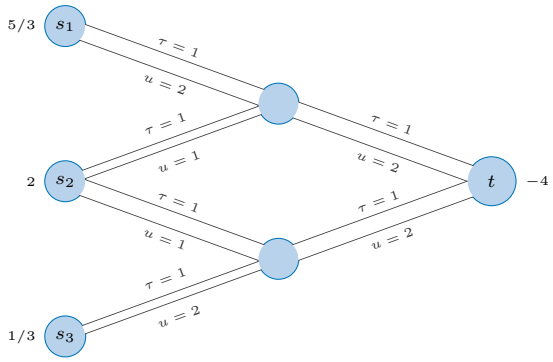
is a submodular function over the ground set $S^+ \cup S^-$ whose base polytope $\mathcal{B}(o^T)$ is defined by

$$\mathcal{B}(o^T) := \{x \in \mathbb{R}^{S^+ \cup S^-} \mid x(X) \leq o^T(X) \; \forall \; X \subseteq S^+ \cup S^- \text{ and } x(S^+ \cup S^-) = o^T(S^+ \cup S^-)\}. \quad (4.4)$$

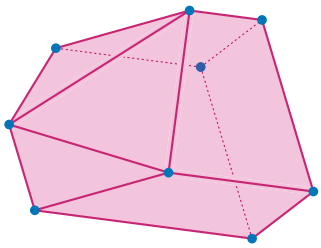
See Figure 4.7 for an example of a dynamic network together with the corresponding base polytope. The main idea for proving Theorem 4.11 is to establish a connection between the base polytope $\mathcal{B}(o^T)$ and flows over time. The proof essentially consists of two parts. The first step is to deduce that the supply/demand vector lies in the base polytope $\mathcal{B}(o^T)$ if and only if the transshipment problem $(\mathcal{N}, b, T)$ is feasible.
In the second part, a correspondence between the vertices of $\mathcal{B}(o^T)$ and lex-max flows over time with time horizon T is shown. The two parts are summarized in the following two lemmas, which are illustrated in Figure 4.8a.

Lemma 4.12. *Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network, b a supply/demand function on the terminals and $T \geq 0$ a time horizon, then*

$$b \in \mathcal{B}(o^T) \text{ if and ony if } (\mathcal{N}, b, T) \text{ is a feasible transshipment over time problem.}$$



(a) A quickest transshipment problem $(\mathcal{N}, b)$ corresponding to the depicted dynamic network $\mathcal{N}$ and the supplies and demands b shown in the picture



(b) The base polytope $\mathcal{B}(o^T)$ corresponding to the dynamic network $\mathcal{N}$ in Figure 4.7a and time horizon $T = 3$

Figure 4.7: A dynamic network $\mathcal{N}$ with time horizon $T = 3$ together with its base polytope

Proof. Assume at first that $(\mathcal{N}, b, T)$ is a feasible transshipment over time problem. Thus, the feasibility criterion of Klinz [Kli] as stated in Theorem 3.5 implies that

$$b(X) \overset{\text{Thm. 3.5}}{\leq} o^T(X) \text{ for all } X \subseteq S^+ \cup S^-.$$

By assumption, we have

$$b(S^+ \cup S^-) = 0,$$

and by definition $o^T(S^+ \cup S^-)$ is the maximum amount of flow that can be sent from S^+ towards an empty set of sinks. Thus, we have

$$o^T(S^+ \cup S^-) = 0 = b(S^+ \cup S^-),$$

which implies that $b \in \mathcal{B}(o^T)$ by the definition of the base polytope in (2.4).
On the other hand, if $b \in \mathcal{B}(o^T)$, then the definition of $\mathcal{B}(o^T)$ in (2.4) immediately implies that we have $b(X) \leq o^T(X)$ for all $X \subseteq S^+ \cup S^-$ and thus $(\mathcal{N}, b, T)$. □

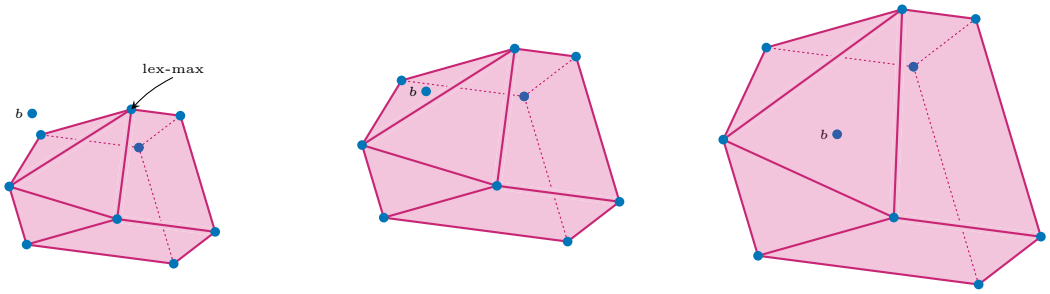
With Lemma 4.12 we have shown that for a given time horizon T the set of all feasible supply/demand vectors is exactly $\mathcal{B}(o^T)$, i.e.,

$$\{b \in \mathbb{Z}^{S^+ \cup S^-} \mid b \text{ is a supply/demand vector on } \mathcal{N} \text{ and } (\mathcal{N}, b, T) \text{ is feasible}\} = \mathcal{B}(o^T).$$

In light of this lemma, determining the minimal feasible time horizon T^* for a quickest transshipment problem can be imagined as "blowing up" the polytope $\mathcal{B}(o^T)$ with growing T until the supply/demand vector b lies inside the polytope. With continuously increasing T the polytope $\mathcal{B}(o^T)$ also grows continuously (this follows basically from the continuity of the function $\theta \mapsto o^\theta(X)$ for all $X \subseteq S^+ \cup S^-$) such that the vector b lies in a facet of $\mathcal{B}(o^{T^*})$ if T^* is the minimal feasible time horizon for $(\mathcal{N}, b)$ (see Figure 4.8). The next step is to derive a correspondence between the vertices of the base polytope $\mathcal{B}(o^T)$ for a given transshipment problem $(\mathcal{N}, b, T)$ and lex-max flows over time with time horizon T. For a given flow over time f in a dynamic network $\mathcal{N}$ with time horizon T, we define the **characteristic vector** $x_f \in \mathbb{R}^{S^+ \cup S^-}$ of f as follows

$$x_f(s) := \text{net}_f(v, T) = \text{out}_f(s, T) - \text{in}_f(s, T) \text{ for all } s \in S^+ \cup S^-.$$

Thus, for $s \in S^+ \cup S^-$ the value $|x_f(s)|$ is the **net amount of flow** sent out of terminal s before time T. So, if $s \in S^-$, then $x_f(s)$ is the overall amount of flow that has traveled into s until time T. Clearly, if the characteristic vector x_f of a flow over time f with time horizon T is equal to b, then f solves the transshipment over time problem $(\mathcal{N}, b, T)$.
For example, the characteristic vector of the flow over time depicted in Figure 4.6 is $(1, 1, -2)$.



(a) The transshipment problem $(\mathcal{N}, b, T_1)$ for $T_1 = 2.7$ is *not* feasible, as $b \notin \mathcal{B}(o^{T_1})$ (see Lemma 4.12). The vertices of $\mathcal{B}(o^{T_1})$ are characteristic vectors of lex-max flows over time with time horizon T_1 in $\mathcal{N}$ (see Lemma 4.13).

(b) The transshipment problem $(\mathcal{N}, b, T_2)$ for $T_2 = 3$ is feasible, as $b \in \mathcal{B}(o^{T_2})$. The fact that T_2 is the minimal feasible time horizon for $(\mathcal{N}, b)$ implies that b has to lie on the boundary of the base polytope.

(c) The transshipment problem $(\mathcal{N}, b, T_3)$ for $T_3 = 3.2$ is feasible, as $b \in \mathcal{B}(o^{T_3})$. But T_3 is not the minimal feasible time horizon for $(\mathcal{N}, b)$ as b lies in the interior of the base polytope.

Figure 4.8: An illustration of Lemmas 4.12 and 4.13 and the proof of Theorem 4.11 showing $\mathcal{B}(o^{T_i})$, for $i = 1, 2, 3$, for a given quickest transshipment problem $(\mathcal{N}, b)$ with time horizons $T_1 = 2.8$, $T_2 = 3$, and $T_3 = 3.2$, where $\mathcal{N}$ is the dynamic network depicted in Figure 4.7a and $b = (5/3, 6/3, 1/3)$: a given transshipment problem $(\mathcal{N}, b, T)$ is feasible if and only if $b \in \mathcal{B}(o^T)$, and the vertices of the base polytope correspond to lex-max flows over time with time horizon T in $\mathcal{N}$.

Recall, that the vertices of the base polytope $\mathcal{B}(o^T)$ correspond to total orders on $S^+ \cup S^-$ and vice versa, and that the vertex $v^\prec$ corresponding to a given total order $\prec$ on $S^+ \cup S^-$ can be computed in strongly polynomial time using the Greedy Algorithm (see Algorithm 2 and Theorem 2.3).

Lemma 4.13. *Let $\mathcal{N}$ be a dynamic network, $T \geq 0$ a time horizon, and $\prec$ a total order on $S^+ \cup S^-$. Denote by $v^\prec$ the vertex of $\mathcal{B}(o^T)$ corresponding to $\prec$. Then*

$$v^\prec = x_{f_\prec},$$

where $f_\prec$ is the lexicographically maximum flow over time with time horizon T with respect to $\prec$. In particular,

$$\mathcal{B}(o^T) = \operatorname{conv}(\{x_{f_\prec} \mid \prec \text{ is a total order on } S^+ \cup S^-\}).$$

Proof. Theorem 2.3 implies that each vertex of the base polytope $\mathcal{B}(o^T)$ corresponds to one or multiple total orders on $\prec$. Conversely, each total order $\prec$ corresponds to a unique vertex $v^\prec$ of $\mathcal{B}(o^T)$, which can be computed using the Greedy Algorithm of Edmonds as stated in Theorem 2.3. This implies

$$\mathcal{B}(o^T) = \operatorname{conv}(\{v^\prec \mid \prec \text{ is a total order on } S^+ \cup S^-\}).$$

Moreover, each total order $\prec$ on $S^+ \cup S^-$ induces a lexicographically maximum flow over time with time horizon T in $\mathcal{N}$, and, on the other, hand each such flow corresponds to one or multiple total orders on $S^+ \cup S^-$. Let $\prec$ be a fixed total order on $S^+ \cup S^-$ and assume that $f_\prec$ is the corresponding lexicographically maximum flow over time with time horizon T, and that $v^\prec$ is the vertex of $\mathcal{B}(o^T)$ defined by $\prec$. Using Lemma 3.11 we get for all $u \in S^+ \cup S^-$,

$$\begin{aligned} x_{f_\prec}(u) &= \operatorname{net}_{f_\prec}(u, T) \\ &\overset{\text{Lem. 3.11}}{=} o^T(\{u' \in S^+ \cup S^- \mid u' \preceq u\}) - o^T(\{u' \in S^+ \cup S^- \mid u' \prec u\}). \end{aligned}$$

Using Theorem 2.3, we also get for all $u \in S^+ \cup S^-$,

$$v^{\prec}(u) \overset{\text{Thm. 2.3}}{=} o^T(\{u' \in S^+ \cup S^- \mid u' \preceq u\}) - o^T(\{u' \in S^+ \cup S^- \mid u' \prec u\}).$$

Thus, it holds that $x_{f_\prec} = v^\prec$ for each fixed order $\prec$ on $S^+ \cup S^-$. This implies that

$$\begin{aligned}\mathcal{B}(o^T) &= \operatorname{conv}\{v^\prec \mid \prec \text{ is a total order on } S^+ \cup S^-\} \\ &= \operatorname{conv}(\{x_{f_\prec} \mid \prec \text{ is a total order on } S^+ \cup S^-\}),\end{aligned}$$

which concludes the proof. □

Lemma 4.12 and Lemma 4.13 essentially imply Theorem 4.11. An example can be seen in Figure 4.9. The last ingredient we need, is a classical theorem from convex geometry – Carathéodory's Theorem.

Theorem 4.14 (Carathéodory's Theorem [Car07]).

Let $P \subseteq \mathbb{R}^d$ be a finite set such that $\operatorname{conv}(P)$ is a full-dimensional polytope. A point $x \in \mathbb{R}^d$ that lies in the convex hull of P, i.e., $x \in \operatorname{conv}(P)$, can be written as a convex combination of at most $d+1$ vertices of the polytope $\operatorname{conv}(P)$.

We are now ready to state the proof of our main result, Theorem 4.11.

Proof of Theorem 4.11. Lemma 4.13 implies that

$$\mathcal{B}(o^T) \overset{\text{Thm. 4.13}}{=} \operatorname{conv}(\{x_{f_\prec} \mid \prec \text{ is a total order on } S^+ \cup S^-\}).$$

Since $(\mathcal{N}, b, T)$ is a feasible transshipment over time problem, Lemma 4.12 implies

$$b \in \mathcal{B}(o^T),$$

i.e., b can be written as a convex combination of vertices of $\mathcal{B}(o^T)$. That is, there are total orders $\prec_1, \prec_2, \dots, \prec_d$ on $S^+ \cup S^-$ and coefficients $\lambda_1, \lambda_2, \dots, \lambda_d \geq 0$ with $\sum_{i=1}^d \lambda_i = 1$ such that

$$b = \sum_{i=1}^{d} \lambda_i v^{\prec_i}.$$

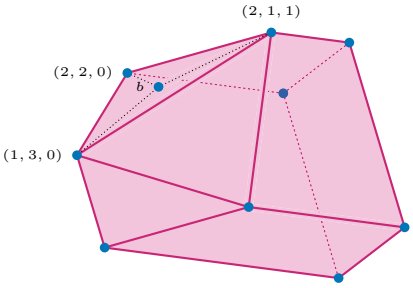
Here $v^{\prec_i}$ is the vertex of $\mathcal{B}(o^T)$ corresponding to $\prec_i$, for $i = 1, \dots, d$. Using Lemma 4.13 yields

$$b \overset{\text{Lem. 4.13}}{=} \sum_{i=1}^{d} \lambda_i x_{f_{\prec_i}},$$

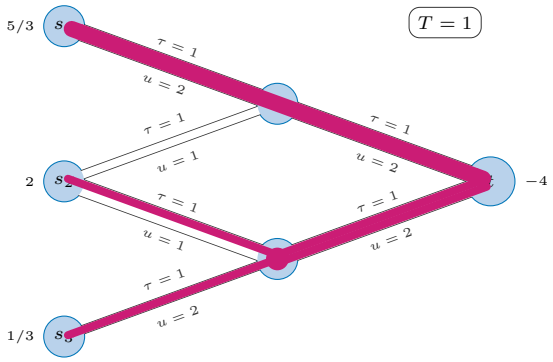
because of $v^{\prec_i} = x_{f_{\prec_i}}$ for $i = 1, \dots, d$. Clearly, $f := \sum_{i=1}^s \lambda_i f_{\prec_i}$ is a feasible flow over time with time horizon T in $\mathcal{N}$ and $x_f = b$. Thus, the flow over time f – a convex combination of lexicographically maximum flows over time with time horizon T – solves the transshipment over time problem $(\mathcal{N}, b, T)$.
Next, we show that at most $|S^+ \cup S^-|$ many lexicographically maximum flows over time are needed in a convex combination solving $(\mathcal{N}, b, T)$. The ambient space of the base polytope $\mathcal{B}(o^T)$ is $\mathbb{R}^{S^+ \cup S^-}$, which is $|S^+ \cup S^-|$-dimensional. By definition, $\mathcal{B}(o^T)$ lies inside the hyperplane defined by $x(S^+ \cup S^-) = 0$. Therefore,

$$\dim(\mathcal{B}(o^T)) \leq |S^+ \cup S^-| - 1.$$

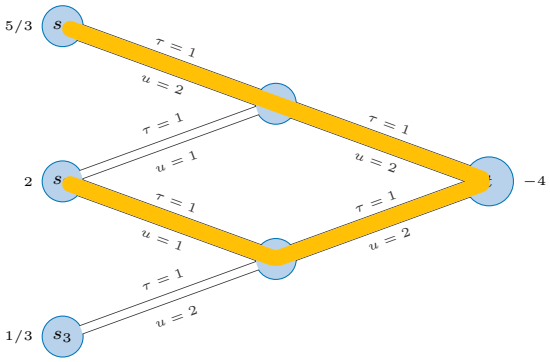
Carathéodory's Theorem (see Theorem 4.14) now yields that each point inside $\mathcal{B}(o^T)$ is a convex combination of at most $|S^+ \cup S^-|$ many vertices. Hence, a solution to the transshipment problem $(\mathcal{N}, b, T)$ can be achieved by a convex combination of at most $|S^+ \cup S^-|$ many lexicographically maximum flows over time with time horizon T.



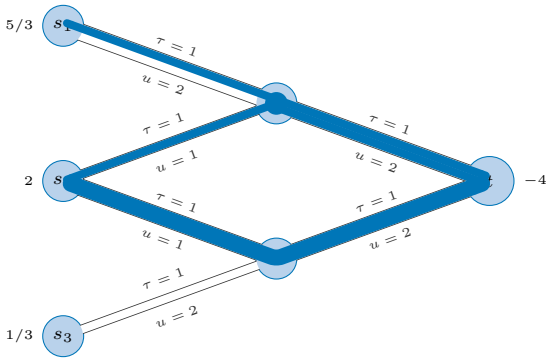
(a) The base polytope $\mathcal{B}(o^T)$ corresponding to the dynamic network shown in Figure 4.7a with time horizon $T = 3$. The supply demand vector $b = (5/3, 2, 1/2)$ of the quickest transshipment problem $(\mathcal{N}, b)$ show in Figure 4.7a lies inside a facet of the base polytope and we have $b = \frac{1}{3}v_1 + \frac{1}{3}v_2 + \frac{1}{3}v_3$ where $v_1 = (2, 1, 1)$ is the vertex corresponding to the order $\prec_1$ with $s_1 \prec_1 s_3 \prec_1 s_2 \prec_2 t$, $v_2 = (2, 2, 0)$ is the vertex corresponding to the order $\prec_2$ with $s_1 \prec_2 s_2 \prec_2 s_3 \prec_2 t$, and $v_3 = (1, 3, 0)$ is the vertex corresponding to the order $\prec_3$ with $s_2 \prec_3 s_1 \prec s_3 \prec_3 t$.



(b) A lex-max flow over time with respect to $\prec_1$ sends flow into the indicated paths for one time unit.



(c) A lex-max flow over time with respect to $\prec_2$ sends flow into the indicated paths for one time unit.



(d) A lex-max flow over time with respect to $\prec_3$ sends flow into the indicated path for one time unit.

Figure 4.9: The convex combination of vertices of $\mathcal{B}(o^T)$ that yields a vector b inside the base polytope gives rise to a convex combination of lex-max flows over time solving the transshipment problem $(\mathcal{N}, b, T)$: the convex combination $\frac{1}{3}f_1 + \frac{1}{3}f_2 + \frac{1}{3}f_3$ of the lex-max flows over time f_i with time horizon $T = 3$ with respect to $\prec_i$, for $i = 1, 2, 3$, solves the quickest transshipment problem $(\mathcal{N}, b)$ shown in Figure 4.7a. As an example, consider the source s_2. In $\frac{1}{3}f_1$ this source sends $\frac{1}{3}$ flow units, in $\frac{1}{3}f_2$ it sends $\frac{2}{3}$ flow units and in $\frac{1}{3}f_3$ it sends one flow unit to the sink t. Overall, 2 flow units are sent by s_2.

Geometrically, finding the minimum feasible time horizon for $(\mathcal{N}, b)$ is the same as finding the minimum T^* such that $b \in \mathcal{B}(o^T)$. Note that the function $o^T(X)$ is continuous in T for each fixed $X \subseteq S^+ \cup S^-$. Thus, with increasing T, the polytope $\mathcal{B}(o^T)$ also "grows" continuously. Hence, there exists a minimal time T^* such that b lies in the boundary of $\mathcal{B}(o^T)$, i.e., in a facet of $\mathcal{B}(o^T)$. With the same reasoning as before – a facet of $\mathcal{B}(o^T)$ has at most dimension $|S^+ \cup S^-| - 2$ –

Carathéodory's Theorem implies that a feasible transshipment solving $(\mathcal{N}, b, T^*)$ can be achieved by finding a convex combination of at most $|S^+ \cup S^-| - 1$ many lexicographically maximum flows over time with time horizon T^*. □

4.3 Solving Transshipment Over Time Problems

Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network and assume that $(\mathcal{N}, b, T)$ is a feasible transshipment over time problem. We have shown in Theorem 4.11 in the previous section that this problem can be solved by a convex combination of at most $|S^+ \cup S^-|$ many lexicographically maximum flows over time with time horizon T. However, it is still completely unclear how to find such a convex combination. In this section we present two different algorithms to solve a given transshipment over time problem that boil down to two methods to determine a suitable convex combination of lex-max flows over time solving a given feasible transshipment problem.
At first we describe the more straightforward, but a less efficient approach to compute a suitable convex combination – an implementation of Carathéodory's Theorem that relies on a recent result by Goemans et al. [GGJ17] about line search over base polytopes of submodular functions. Although this approach already yields an algorithm with improved worst case running time compared to the algorithm of Hoppe and Tardos, the second method we describe in Section 4.3.2 yields an even greater improvement. Our second algorithm for computing a suitable convex combination of lex-max flows over time relies on submodular function minimization algorithms using the framework of Cunningham (see Section 2.3.3). It turns out that a convex combination of lex-max flows over time solving $(\mathcal{N}, b, T)$ in case of feasibility is basically computed as a byproduct of the submodular function minimization that is required to check the feasibility of $(\mathcal{N}, b, T)$.
Since both of our algorithms solve given (quickest) transshipment problems by computing a suitable convex combination of lex-max flows over time, the solutions to $(\mathcal{N}, b)$ we achieve are compared to the flows over time computed by the algorithm of Hoppe and Tardos structurally much simpler: they are generalized temporally repeated flows. The downside of our approach is that the flow over time we compute is in general not integral – it is in general not possible to come up with integral generalized temporally repeated flows solving a given (quickest) transshipment problem (see Figure 4.1).

4.3.1 An Implementation of Carathéodory's Theorem

The results from the previous section imply that a feasible transshipment over time problem $(\mathcal{N}, b, T)$ can be solved by a convex combination of lex-max flows over time with time horizon T. The goal in this section is to describe an efficient way to compute a suitable convex combination of lex-max flows over time. By Lemma 4.12 and Lemma 4.13 it suffices to find a convex combination of vertices of $\mathcal{B}(o^T)$ that yield the supply/demand vector $b \in \mathcal{B}(o^T)$. The first algorithm that comes into mind when having to solve such a problem is an implementation of Carathéodory's Theorem or, more precisely, an implementation of a classical proof of this theorem. Carathéodory's Theorem for this special case states that b can be obtained as convex combination of at most $|S^+ \cup S^-|$ vertices of $\mathcal{B}(o^T)$ and the classical proof of this theorem immediately leads to an algorithm to determine a suitable convex combination. The main insight of this section will be that for our special case the algorithmic version of Carathéodory's Theorem can be implemented in strongly polynomial time.
We start by recalling the statement and the classical proof of Carathéodory's Theorem that leads to the desired algorithm. Recall that Carathéodory's Theorem states the following:
Let $P \subseteq \mathbb{R}^d$ be a finite set such that $\text{conv}(P)$ is a full-dimensional polytope. Then a point $x \in \text{conv}(P)$ can be written as a convex combination of at most $d + 1$ vertices of $\text{conv}(P)$.
A simple proof of this theorem is the following proof by induction on d – an illustration of this proof can be found in Figure 4.10.

Proof of Carathéodory's Theorem (Theorem 4.14). We start with $d = 1$. Let $P \subseteq \mathbb{R}^1$, then $\text{conv}(P)$ has two vertices since $\text{conv}(P)$ is full-dimensional by assumption, which immediately implies the statement of Carathéodory's Theorem.
Next assume that the statement is true for all d' with $d' \leq d$ for some $d > 0$. Let $P \subseteq \mathbb{R}^{d+1}$ be a finite set, $x_1 \in \text{conv}(P)$, and $w_1 \in P$ a vertex of $\text{conv}(X)$. We can assume that x_1 lies in

the interior of $\text{conv}(P)$ as if x_1 lies in a face of $\text{conv}(P)$, we are done by induction. We consider the ray R that starts at the vertex w_1 and goes through the point x_1. The ray R intersects the boundary of $\text{conv}(P)$ at some point x_2. Since x_1 lies in the interior of $\text{conv}(P)$ by assumption, we have $x_1 \neq x_2$. Formally, we consider $R(\mu) \coloneqq w_1 + \mu \cdot (x_1 - w_1)$ for all $\mu \in \mathbb{R}$ and determine the maximum value μ^* such that $R(\mu^*) \in \text{conv}(P)$. Clearly, we have $\mu^* \geq 1$ because $x_1 \in \text{conv}(P)$ by assumption. Define $x_2 = R(\mu^*)$ and compute λ_1 such that $x_1 = \lambda_1 w_1 + (1 - \lambda_1)x_2$.
The point x_2 lies in a face F of $\text{conv}(P)$ with $F \neq \text{conv}(P)$ and thus, in a polytope which is at most d-dimensional. By induction we can achieve the point x_2 as convex combination of at most $d+1$ vertices of the face F. Since the required point x_1 can be obtained as a convex combination of x_2 and w_1, x_1 can be obtained as a convex combination of at most $d+2$ vertices of $\text{conv}(P)$. See Figure 4.10b for a visualization. □

The above proof of Carathéodory's Theorem inspires Algorithm 13 for determining a point $x \in \text{conv}(P)$ as a convex combination of vertices of $\text{conv}(P)$, for a finite subset $P \subseteq \mathbb{R}^d$.

Algorithm 13: Implementation of Carathéodory's Theorem

Input : $Q = \text{conv}(v_1, v_2, \ldots, v_r) \subseteq \mathbb{R}^d$, where $v_1, \ldots, v_r$ are the vertices of Q, $x \in Q$
Output : A convex combination of vertices of Q yielding x

$i \leftarrow 1$, $F_i \leftarrow Q$, $x_i \leftarrow x$, $\lambda_0 \leftarrow 0$
while $i \leq d$ **do**
  $w_i \leftarrow$ arbitrary vertex of F_i
  $\mu^* \leftarrow \max\{\mu \mid w_i + \mu \cdot (x_i - w_i)\} \in Q$
  $x_{i+1} \leftarrow w_i + \mu^* \cdot (x_i - w_i)$
  if $\mu^* = 1$ **then**
    $\lambda_i \leftarrow 0$
  else
    $\lambda_i \leftarrow \left(1 - \sum_{j=0}^{i-1} \lambda_j\right) \cdot (\mu^* - 1)/\mu^*$
  end
  $F_{i+1} \leftarrow$ real face of F_i with $x_{i+1} \in F_{i+1}$
  $i \leftarrow i + 1$
end
$\lambda_{d+1} \leftarrow 1 - (\lambda_1 + \ldots + \lambda_d)$
$w_{d+1} \leftarrow x_{d+1}$
return $w_1, \ldots, w_{d+1}$, $\lambda_1, \ldots, \lambda_{d+1}$

Theorem 4.15.

Let $Q = \text{conv}(v_1, \ldots, v_r) \subseteq \mathbb{R}^d$ where $v_1, \ldots, v_r \subseteq \mathbb{R}^d$ are the vertices of Q and let $x \in Q$. Assume that Q is a $d' \leq d$ dimensional polytope. Algorithm 13 returns a convex combination of at most $d'+1$ many vertices yielding the point x in Q. Furthermore, the algorithm terminates after d iterations.

Proof. That Algorithm 13 terminates after at most d iterations is clear from our arguing above. Denote by $w_1, \ldots, w_{d+1}$ the vertices of Q returned by the algorithm and by $\lambda_1, \ldots, \lambda_{d+1}$ the convex coefficients. Also, by construction the algorithm returns at most $d'+1$ coefficients that are non-zero. Thus, there are $d'+1$ vertices in $\{w_1, \ldots, w_{d+1}\}$ such that a convex combination of them yields x. It remains to be shown that $\lambda_1, \ldots, \lambda_{d+1}$ are the correct convex coefficients such that the convex combinations yields x. We start by showing that $\lambda_1 + \ldots + \lambda_i \leq 1$ for all $i + \{1, \ldots, d\}$ which by the construction of the λ_i immediately implies that $0 \leq \lambda_i \leq 1$ for all $i \in \{1, \ldots, d+1\}$ and $\lambda_1 + \ldots + \lambda_{d+1} = 1$. We show this by induction on i. For $i = 1$ we clearly have

$$0 \leq \frac{\mu^* - 1}{\mu^*} \leq 1,$$

because $\mu^* \geq 1$. Assume now, we have $\lambda_1 + \ldots + \lambda_i \leq 1$ for some $1 \leq i < d$. Then,

$$\begin{aligned}\lambda_1 + \ldots + \lambda_i + \lambda_{i+1} &= \lambda_1 + \ldots + \lambda_i + \frac{\mu^* - 1}{\mu^*}(1 - (\lambda_1 + \ldots + \lambda_i)) \\ &= \frac{1}{\mu^*}\underbrace{(\lambda_1 + \ldots + \lambda_i)}_{\leq 1} + \frac{\mu^* - 1}{\mu^*} \leq 1.\end{aligned}$$

To show that $x = \lambda_1 w_1 + \ldots + \lambda_{d+1} w_{d+1}$ we show by induction on i that $x = \lambda_1 w_1 + \ldots + \lambda_i w_i + (1 - (\lambda_1 + \ldots + \lambda_i))x_{i+1}$ for all $i \in \{1, \ldots, d\}$. We start with $i = 1$. If in iteration $i = 1$ we have $\mu^* = 1$, then $x_2 = w_1$ and $\lambda_1 = 0$ and we are immediately done. Assume $\mu^* > 1$. Then,

$$\begin{aligned}\lambda_1 w_1 + (1 - \lambda_1)x_2 &= \frac{\mu^* - 1}{\mu^*} w_1 + \frac{1}{\mu^*} x_2 \\ &= \frac{\mu^* - 1}{\mu^*} w_1 + \frac{1}{\mu^*}(w_1 + \mu^*(x - w_1)) = x.\end{aligned}$$

Now assume that the statement holds for $1 \leq i < d$. If $\mu^* = 1$ in iteration $i + 1$ of the algorithm, it is again clear that we have $x = \lambda_1 w_1 + \ldots + \lambda_{i+1} w_{i+1} + (1 - (\lambda_1 + \ldots + \lambda_{i+1}))x_{i+2}$. Thus, assume $\mu^* > 1$. By assumption, we have

$$\lambda_1 w_1 + \ldots + \lambda_i w_i + (1 - (\lambda_1 + \ldots + \lambda_i))x_{i+1} = x. \tag{4.5}$$

Also, note that

$$x_{i+2} = w_{i+1} + \mu^*(x_{i+1} - w_{i+1}) \Rightarrow x_{i+1} = \frac{1}{\mu^*}(x_{i+2} - w_{i+1}) + w_{i+1}, \tag{4.6}$$

and

$$\begin{aligned}(1 - (\lambda_1 + \ldots + \lambda_{i+1})) &= (1 - (\lambda_1 + \ldots + \lambda_i)) - \lambda_{i+1} \\ &= (1 - (\lambda_1 + \ldots + \lambda_i)) - (1 - (\lambda_1 + \ldots + \lambda_i)) \cdot \frac{\mu^* - 1}{\mu^*} \\ &= \frac{1}{\mu^*}(1 - (\lambda_1 + \ldots + \lambda_i)).\end{aligned} \tag{4.7}$$

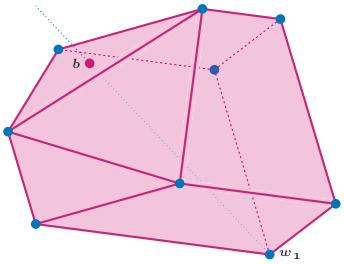
Putting these three equations together yields

$$\begin{aligned}x &\overset{(4.6)}{=} \lambda_1 w_1 + \ldots + \lambda_i w_i + \big(1 - (\lambda_1 + \ldots + \lambda_i)\big) \cdot \left(\frac{1}{\mu^*}(x_{i+2} - w_{i+1}) + w_{i+1}\right) \\ &= \lambda_1 w_1 + \ldots + \lambda_i w_i + \left(\frac{\mu^* - 1}{\mu^*}\big(1 - (\lambda_1 + \ldots + \lambda_i)\big)\right) w_{i+1} + \frac{1}{\mu^*}\Big(1 - (\lambda_1 + \ldots + \lambda_i)\Big) x_{i+2} \\ &\overset{(4.7)}{=} \lambda_1 w_1 + \lambda_2 w_2 + \ldots + \lambda_{i+1} w_{i+1} + \big(1 - (\lambda_1 + \ldots + \lambda_{i+1})\big) x_{i+2}.\end{aligned}$$

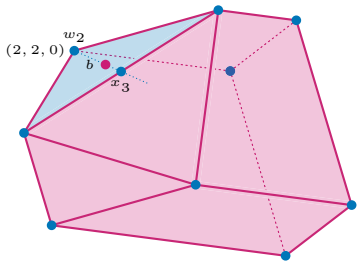
□

For arbitrary convex hulls of finite sets it is not clear how to implement Algorithm 13 efficiently. When a polytope is given as a convex hull of vertices and not as an intersection of half spaces, it is for example usually not straightforward how to determine a facet which contains a given vertex or how to compute μ^*. However, we are only interested in implementing this algorithm for base polytopes of submodular functions and it turns out that in this case the algorithm can be implemented in strongly polynomial time. Overall, we need to be able to solve the following subproblems efficiently in order to obtain a fast implementation of Carathéodory's Theorem for the special case of $Q = \mathcal{B}(o^T)$ for some time horizon $T \geq 0$:

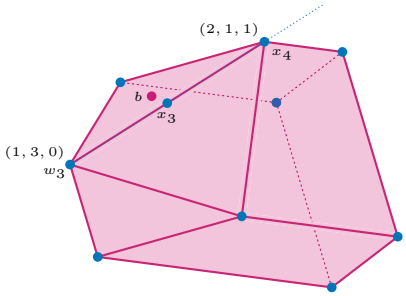
1. choosing an arbitrary vertex w of a given face F of $\mathcal{B}(o^T)$,
2. doing line search over $\mathcal{B}(o^T)$ in an arbitrary direction (needed to determine μ^*),
3. finding a face of $\mathcal{B}(o^T)$ containing the vector x obtained by the line search.



(a) Iteration $i = 1$ of Algorithm 13. A vertex w_1 of $\mathcal{B}(o^3)$ is chosen, here $w_1 = (0, 0, 0)$. Since b lies in a facet of $\mathcal{B}(o^3)$ that does not contain w_1, we get $\mu^* = 1$ (the ray from w_1 going through b is shown in the figure). This implies $x_2 = b$ and $\lambda_1 = 0$.



(b) Iteration $i = 2$ of Algorithm 13: The currently regarded face F_2 of $\mathcal{B}(o^3)$ is the blue colored face shown in the figure. As vertex w_2 in F_2 we choose (2,2,0). The ray from w_2 through b cuts the boarder of F_2 at the point $x_2 = (3/2, 2, 1/2)$ for $\mu^* = 3/2$ which implies $\lambda_2 = 1/3$.



(c) Iteration $i = 3$ of Algorithm 13: $w_3 = (1, 3, 0)$, $\mu^* = 2$, $x_3 = (2, 1, 1)$ and $\lambda_3 = 1/3$. The last convex coefficient is thus $\lambda_4 = 1/3$ and $x_4 = (2, 1, 1)$.

Figure 4.10: Illustration of Algorithm 13. The depicted polytope is the base polytope $\mathcal{B}(o^3)$ of the dynamic network shown in Figure 4.7a and $b = (5/3, 2, 1/3)$. The overall goal is to find a convex combination of vertices yielding the vector b.

To be able to solve the first and the third task, we need to understand how the faces of the base polytope of a submodular function can be characterized. This is explained in the following paragraph. How line search over the base polytope $\mathcal{B}(o^T)$ can be done efficiently, will be explained subsequently.

The Faces of a Base Polytope. Let $g\colon 2^E \to \mathbb{R}$ be a submodular function over a ground set E. Recall that the base polytope $\mathcal{B}(g)$ is defined as follows

$$\mathcal{B}(g) := \{x \in \mathbb{R}^E \mid x(X) \leq f(X) \text{ for all } X \subseteq E \text{ and } x(E) = g(E)\}.$$

In the following we shortly summarize how the faces of $\mathcal{B}(g)$ look like. These results are due to Fujishige [Fuj58]. For $x \in \mathcal{B}(g)$ we define,

$$\mathcal{D}(x) := \{X \subseteq E \mid x(X) = g(X)\}. \tag{4.8}$$

For any $\mathcal{D} \subseteq 2^E$ we additionally define

$$\mathcal{F}(\mathcal{D}) = \{x \in \mathcal{B}(g) \mid x(X) = g(X) \text{ for all } X \in \mathcal{D}\}.$$

Theorem 4.16 ([Fuj58]).

The collection $\mathbb{F}$ of all the nonempty faces of $\mathcal{B}(g)$ is given by

$$\mathbb{F} = \{\mathcal{F}(\mathcal{D}(x)) \mid x \in \mathcal{B}(g)\}.$$

In particular, if we are given a subset $X \subseteq E$, then the set of all vector $x \in \mathcal{B}(g)$ that are **tight** for this set, i.e., all $x \in \mathcal{B}(g)$ with $x(X) = g(X)$, defines a face F of $\mathcal{B}(g)$. The vertices of such

a face F are of course vertices of $\mathcal{B}(g)$, that is, they correspond to total orders $\prec$ of E. Thus, the vertices of F correspond to special orders of E, namely orders in which X is a **lower ideal**.

Observation 4.17. *Each subset $X \subseteq E$ defines a face F of $\mathcal{B}(g)$ given by*

$$F = \{x \in \mathcal{B}(g) \mid x(X) = g(X)\}.$$

The vertices of F correspond to orders $\prec$ on E with the property that for each $x \in X$ we have

$$x \prec e \text{ for all } e \in E \setminus X.$$

Line Search Over the Base Polytope of a Submodular Function. The second problem we need to solve is doing line search over the base polytope of a submodular function. Let $g\colon 2^E \to \mathbb{R}$ be a submodular function over a ground set E. Given an arbitrary $a \in \mathbb{R}^E$ and $x_0 \in \mathcal{B}(g)$ our goal is to compute $\max\{\mu \mid x_0 + \mu a \in \mathcal{B}(g)\}$. Without loss of generality, we can assume that $x_0 = 0$, i.e., we can assume that we want to determine the maximal μ such that $\mu a \in \mathcal{B}(g)$. Thus, we are interested in the problem

$$\mu^* = \max\{\mu \mid \min_{X \subseteq E} g(X) - \mu a(X) \geq 0\}$$

for a non-negative submodular function g. In [GGJ17] Goemans et al. analyze the discrete Newton's method for this problem and show that it terminates after at most $|E|^2 + \mathcal{O}(|E| \log^2 |E|)$ iterations. See Algorithm 14 for the full formulation of the discrete Newton's algorithm used in [GGJ17].

Algorithm 14: Discrete Newton's Algorithm for line search over a submodular function's base polytope [GGJ17]

Input : A submodular function $g\colon 2^E \to \mathbb{R}_{\geq 0}$, $a \in \mathbb{R}^E$
Output : $\mu^* = \max\{\mu \mid \min_{X \subseteq E} g(X) - \mu a(X) \geq 0\}$

$i \leftarrow 0$, $\mu_1 \leftarrow \min_{e \in E\colon a(\{e\})>0} g(\{e\})/a(\{e\})$
do
  $i \leftarrow i + 1$
  $h_i \leftarrow \min_{X \subseteq E} g(X) - \mu_i a(X)$
  $S_i \in \operatorname{argmin}_{X \subseteq E} g(X) - \mu_i a(X)$
  $\mu_{i+1} \leftarrow f(S_i)/a(S_i)$
while $h_i \neq 0$
return $\mu^* = \mu$, $S^* = S_i$

Lemma 4.18 ([GGJ17]). *Algorithm 14 terminates after $|E|^2 + \mathcal{O}(|E| \log^2 |E|)$ iterations, each of which requires one submodular function minimization.*

In each iteration i the algorithm has to do one submodular function minimization to determine h_i and S_i. This computation can be done using the algorithm of Lee et al. [LSW15]. In the end the algorithm returns μ^*, that is, the largest parameter for which $\mu a \in \mathcal{B}(g)$, and S^*, the set for which the minimum of $g - \mu^* \cdot a$ it attained. Note that we always have

$$g(S^*) - \mu^* a(S^*) = 0.$$

Solving a Transshipment Problem. We can now put together the observations from the previous two paragraphs to come up with an efficient implementation of Algorithm 13. For this purpose assume that $(\mathcal{N}, b, T)$ is a feasible transshipment problem. Our goal is to determine b as convex combination of vertices of $\mathcal{B}(o^T)$.
Clearly, one main ingredient is to do the required line search during the course of Algorithm 13 with the help of Algorithm 14 (note that we have to translate our submodular function by $-w_i$ before we can apply this algorithm). In each iteration i the line search returns a set $S_{i+1} \subseteq S^+ \cup S^-$

and $\mu^* \geq 1$ such that μ^* is the maximal value with $w_i + \mu^*(x_i - w_i) \in \mathcal{B}(o^T)$. We use μ^* to compute our convex coefficients and S_{i+1} to define the real face F_{i+1} of $\mathcal{B}(o^T)$ containing x_{i+1}. We set $F_{i+1} = \{x \in \mathcal{B}(o^T) \mid x(S_{i+1}) = o^T(S_{i+1})\}$ which is a face by Observation 4.17. We need to argue why F_{i+1} contains the vector x_{i+1}. During the line search, Algorithm 14 determines S_{i+1} to be the minimizer of the submodular function $(o^T - w_i) - \mu(x_i - w_i)$. That is, we have

$$o^T(S_{i+1}) - w_i(S_{i+1}) - \mu^*(x_i - w_i)(S_{i+1}) = 0,$$

j

Observation 4.19. *Using the line search algorithm in Algorithm 14 to obtain an efficient implementation of Algorithm 13 overall yields an algorithm that requires* $|S^+ \cup S^-|^3 + \mathcal{O}(|S^+ \cup S^-|^2 \log^2 |S^+ \cup S^-|)$ *many submodular function minimization to solve a feasible transshipment problem* $(\mathcal{N}, b, T)$.

In Section 4.3.3 we will compare the running time we achieve here with the running time of the algorithm of Hoppe and Tardos and our other algorithm that we derive in the next section.

4.3.2 Exploiting the Framework of Cunningham

Throughout this section, assume that $(\mathcal{N}, b, T)$ is a transshipment problem in a dynamic network $\mathcal{N}$ with time horizon T. Theorem 4.11 implies that – in case of feasibility – there exists a convex combination of at most $|S^+ \cup S^-|$ many lex-max flows over time with time horizon T solving this problem. In the following we will explain how such a convex combination can be determined during the submodular function minimization that is required to check whether $(\mathcal{N}, b, T)$ is feasible, provided that the submodular function minimization is done with an algorithm using the framework of Cunningham, for example with the algorithm of Orlin [Orl93].
Recall, that by the feasibility criterion of Klinz as stated in Theorem 3.5 the transshipment problem $(\mathcal{N}, b, T)$ is feasible if and only if

$$o^T(X) \geq b(X) \text{ for all } X \subseteq S^+ \cup S^-.$$

Thus, we can check the feasibility by minimizing the submodular function $o^T - b$. If the minimum is at least zero, the problem is feasible, otherwise it is not.
Since we want to show that, in case of a feasible problem, a suitable convex combination of lex-max flows over time solving $(\mathcal{N}, b, T)$ is computed as a byproduct of the submodular function minimization with the algorithm of Orlin, we assume in the following that $(\mathcal{N}, b, T)$ is feasible, i.e., $o^T(X) - b(X) \geq 0$ for all $X \subseteq S^+ \cup S^-$. Because we also have $o^T(S^+ \cup S^-) - b(S^+ \cup S^-) = 0$, this implies that in this case the minimum of the submodular function $o^T - b$ is zero, that is,

$$\min\{o^T(X) - b(X) \mid X \subseteq S^+ \cup S^-\} = 0. \tag{4.9}$$

Recall, that algorithms for submodular function minimization using the framework of Cunningham, which we introduced in Section 4.3.2, rely on Theorem 2.4. In this case it states,

$$\min\{o^T(X) - b(X) \mid X \subseteq S^+ \cup S^-\} = \max\{x^-(S^+ \cup S^-) \mid x \in \mathcal{B}(o^T - b)\}. \tag{4.10}$$

Putting together these observations yields the following lemma.

Lemma 4.20. *If* $(\mathcal{N}, b, T)$ *is a feasible transshipment over time problem in a dynamic network* $\mathcal{N}$*, we have*

$$0 = \min\{o^T(X) - b(X) \mid X \subseteq S^+ \cup S^-\} = \max\{x^-(S^+ \cup S^-) \mid x \in \mathcal{B}(o^T - b)\}$$

and

$$\mathbf{0} = \operatorname{argmax}\{x^-(S^+ \cup S^-) \mid x \in \mathcal{B}(o^T - b)\}.$$

Proof. Putting together (4.9) and (4.10) already yields,

$$0 = \min\{o^T(X) - b(X) \mid X \subseteq S^+ \cup S^-\} = \max\{x^-(S^+ \cup S^-) \mid x \in \mathcal{B}(o^T - b)\}.$$

Let $x^* = \operatorname{argmax}\{x^-(S^+ \cup S^-) \mid x \in \mathcal{B}(o^T - b)\}$. The above equation implies,

$$(x^*)^-(S^+ \cup S^-) = 0.$$

That is, the negative components of x^* sum up to zero. Hence,

$$x^*(u) \geq 0 \text{ for all } u \in S^+ \cup S^-.$$

The vector x^* lies in the base polytope $\mathcal{B}(o^T - b)$, which lies within the hyperplane defined by $x(S^+ \cup S^-) = 0$. Thus, the components of x^* sum up to zero yielding that $x^*(u) = 0$ for all $u \in S^+ \cup S^-$ and hence $x^* = \mathbf{0}$. □

Thus, when minimizing the submodular function $o^T - b$, for example with the algorithm of Orlin, also the zero vector is is computed as a convex combination of vertices of $\mathcal{B}(o^T - b)$.
This fact, together with the following lemma, shows how we can obtain a convex combination of vertices of $\mathcal{B}(o^T)$ giving the supply/demand vector b out of the convex combination of vertices of $\mathcal{B}(o^T - b)$ giving the zero vector. See Figure 4.11 for an illustration.

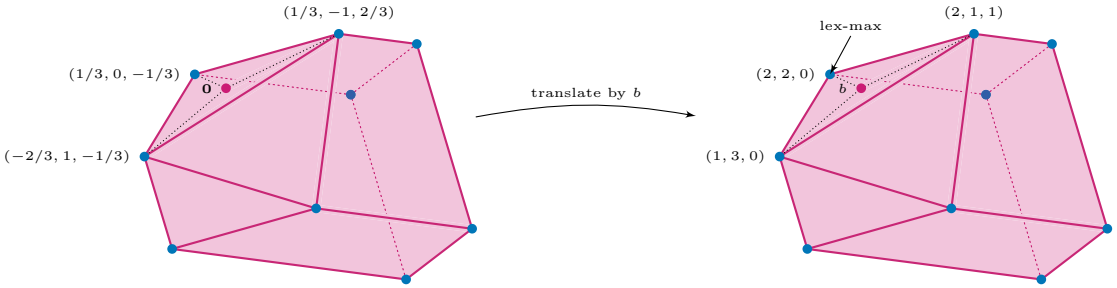


Figure 4.11: Minimizing the submodular function $o^3 - b$ for the transshipment problem depicted in Figure 4.7a yields the zero vector a a convex combination of vertices of $\mathcal{B}(o^3 - b)$ (Lemma 4.20) and translating by b yields a convex combination of vertices of $\mathcal{B}(o^3)$ giving the vector b (Lemma 4.21). The vertices of $\mathcal{B}(o^3)$ are characteristic vectors of lex-max flows over time with time horizon $T = 3$ in $\mathcal{N}$ (see Lemma 4.13). Thus, minimizing the submodular function $o^3 - b$ essentially yields a convex combination of lex-max flows over time solving $(\mathcal{N}, b)$.

Lemma 4.21. *Let $\mathcal{N}$ be a dynamic network, b a supply/demand function on the terminals of $\mathcal{N}$ and $T \geq 0$ a time horizon. The base polytope $\mathcal{B}(o^T)$ is a translation of $\mathcal{B}(o^T - b)$ by the vector b, that is*

$$\mathcal{B}(o^T) = \mathcal{B}(o^T - b) + b.$$

For a fixed total order $\prec$ on $S^+ \cup S^-$, let $v^\prec$ be the corresponding vertex of $\mathcal{B}(o^T)$ and $\overline{v}^\prec$ the corresponding vertex of $\mathcal{B}(o^T - b)$. Then,

$$v^\prec = \overline{v}^\prec + b.$$

Proof. By Theorem 2.3, we have for each $u \in S^+ \cup S^-$

$$v^\prec(u) = o^T(\{u' \in S^+ \cup S^- \mid u' \preceq u\}) - o^T(\{u' \in S^+ \cup S^- \mid u' \prec u\})$$

and

$$\overline{v}^\prec = o^T(\{u' \in S^+ \cup S^- \mid u' \preceq u\}) - o^T(\{u' \in S^+ \cup S^- \mid u' \prec u\}) - b(u').$$

Thus, $v^\prec = \overline{v}^\prec + b$. Since the vertices of both polytopes are completely characterized by the total orders on $S^+ \cup S^-$, this completes the proof. □

Using Lemma 4.20 and Lemma 4.21 we can finally explain our algorithm to compute a convex combination of lexicographically maximum flows over time with time horizon T solving a feasible transshipment over time problem $(\mathcal{N}, b, T)$.
By Lemma 4.20, since $(\mathcal{N}, b, T)$ is a feasible transshipment problem, a submodular function minimization algorithm using the framework of Cunningham to minimize $o^T - b$ returns the zero vector as a convex combination of vertices of $\mathcal{B}(o^T - b)$. An example for such an algorithm is the algorithm of Orlin [Orl09] that we in the following denote by $\mathrm{SFM}_{\mathrm{Orlin}}$. Thus, $\mathrm{SFM}_{\mathrm{Orlin}}(\mathcal{N}, b, T)$ returns the zero vector as a convex combination of vertices of $\mathcal{B}(o^T - b)$ and these vertices are each given by the order $\prec$ on $S^+ \cup S^-$ that they correspond to. That is, we get coefficients $\lambda_1, \dots, \lambda_d \geq 0$ with $\sum_{i=1}^{d} \lambda_i = 1$ and total orders $\prec_1, \dots, \prec_d$ on $S^+ \cup S^-$ such that

$$\mathbf{0} = \lambda_1 \overline{v}^{\prec_1} + \lambda_2 \overline{v}^{\prec_2} + \dots + \lambda_d \overline{v}^{\prec_d}, \tag{4.11}$$

where $\overline{v}^{\prec_i}$ is the vertex of $\mathcal{B}(o^T - b)$ corresponding to $\prec_i$ for $i = 1, \dots, i$. Since the algorithm of Orlin always returns a minimal convex combination of vertices, we get by Theorem 4.14 that $d \leq |S^+ \cup S^-|$. Translating a vertex $\overline{v}^{\prec}$ of $\mathcal{B}(o^T - b)$ by b results in the vertex $v^{\prec}$ of $\mathcal{B}(o^T)$ by Lemma 4.21. Thus, we get

$$\begin{aligned} b &\overset{(4.11)}{=} \lambda_1(\overline{v}^{\prec_1} + b) + \lambda_2(\overline{v}^{\prec_2} + b) + \dots + \lambda_d(\overline{v}^{\prec_d} + b) \\ &\overset{\text{Lem. 4.21}}{=} \lambda_1 v^{\prec_1} + \lambda_2 v^{\prec_2} + \dots + \lambda_d v^{\prec_d}. \end{aligned}$$

Let $f_{\prec_1}, f_{\prec_2}, \dots, f_{\prec_d}$ be the lexicographically maximum flows over time with time horizon T with respect to $\prec_1, \prec_2, \dots, \prec_d$, respectively. Lemma 4.13 implies

$$v^{\prec_i} = x_{f_{\prec_i}} \text{ for all } i \in \{1, \dots, d\},$$

and thus

$$b \overset{\text{Lem. 4.13}}{=} \lambda_1 x_{f_{\prec_1}} + \lambda_2 x_{f_{\prec_2}} + \dots + \lambda_d x_{f_{\prec_d}}.$$

The characteristic vector of the feasible flow over time $f \coloneqq \sum_{i=1}^{d} \lambda_i f_{\prec_i}$ is b and hence the flow over time f solves the transshipment problem $(\mathcal{N}, b, T)$.
Summarizing, for a feasible transshipment problem $(\mathcal{N}, b, T)$ the algorithm of Orlin with input $o^T - b$ returns a suitable convex combination of lexicographically maximum flows over time with time horizon T. More precisely, $\mathrm{SFM}_{\mathrm{Orlin}}(o^T - b)$ returns suitable total orders on $S^+ \cup S^-$ and convex coefficients (see Figure 4.11 for an illustration). The actual lexicographically maximum flows over time with time horizon T have to be computed afterwards (see Algorithm 6).
All these observations lead to the following formal description of an algorithm solving a given transshipment problem $(\mathcal{N}, b, T)$. Note here, that the algorithm of Orlin of course also returns the minimal value $v_{\min}$ of the considered submodular function.

Algorithm 15: Algorithm to solve a given transshipment over time problem $(\mathcal{N}, b, T)$

```
Input  : A dynamic network N = (D = (V, A), u, τ, S⁺, S⁺), a supply/demand function b and a
         time horizon T ≥ 0
Output : A transshipment over time solving (N, b, T) in case of feasibility, INFEASIBLE otherwise
λ_1, ..., λ_d, ≺_1, ≺_2, ..., ≺_d, v_min ← SFM_Orlin(o^T − b)
if v_min < 0 then
    return INFEASIBLE
else
    for i = 1, 2, ..., d do
        f_{≺_i} ← LEXMAX(N, T, ≺_i)
    end
    return λ_1, λ_2, ..., λ_d, f_{≺_1}, f_{≺_2}, ..., f_{≺_d}
end
```

Theorem 4.22 (Correctness of Algorithm 15).

Algorithm 15 solves a given transshipment problem $(\mathcal{N}, b, T)$ within the asymptotic running time of the algorithm of Orlin for submodular function minimization, more precisely, in running time $\mathcal{O}((k^5+k^2)\cdot \mathrm{MCF}(n,m)+n^6)$ with $k = |S^+ \cup S^-|$, $n = |V|$, $m = |A|$, where $\mathcal{O}(\mathrm{MCF}(n,m))$ is the running time required by a minimum-cost flow computation in a network with n vertices and m arcs.
In particular the transshipment over time returned by Algorithm 15 is a generalized temporally repeated flow over time.

Proof. With our observations from above it is immediate that Algorithm 15 solves a given transshipment problem.
In case of feasibility the transshipment over time f computed by Algorithm 15 is a convex combination of lexicographically maximum flows over time with time horizon T. These flows are computed by the algorithm of Hoppe and Tardos (see Algorithm 6). As this algorithm computes a generalized temporally repeated, the flow returned by Algorithm 15 is also a generalized temporally repeated.
An evaluation of $o^T(X) - b(X)$ for some subset $X \subseteq S^+ \cup S^-$ can be achieved by one static minimum-cost flow computation in the static network corresponding to $\mathcal{N}$ (see Observation 3.7). Therefore, for the submodular function $o^T - b$ with ground set $S^+ \cup S^-$ the running time of $\mathrm{SFM}_{\mathrm{Orlin}}(o^T - b)$ is $\mathcal{O}(k^5\mathrm{MCF}(n,m) + k^6)$. In the last step Algorithm 15 requires at most $|S^+ \cup S^-|$ many lexicographically maximum flow over time computations (see Theorem 4.11) and each computations requires $|S^+ \cup S^-|$ many minimum-cost flow computations. Hence, all in all k^2 many additional minimum-cost flow computation are required. □

4.3.3 Summary, Conclusions and Open Questions

In this section we compare the running time of our implementation of Algorithm 13 and the running time of Algorithm 15 with the running time of the algorithm of Hoppe and Tardos for solving the quickest transshipment problem. It turns out that with both algorithms combined with our results from Section 4.1 we obtain significant running time improvements over the algorithm of Hoppe and Tardos, whereas Algorithm 15 is the faster of both algorithms.
The improvements we achieve are summarized in Table 4.1 and Table 4.2

Table 4.1: The running time improvement factors we achieve for quickest transshipment and transshipment over time problems in a dynamic network $\mathcal{N}$ with terminals $S^+ \cup S^-$ by our implementation of Algorithm 13

	single source *or* single sink	**multiple sources *and* sinks**
Transshipment Over Time	$\lvert S^+ \cup S^- \rvert$	$\lvert S^+ \cup S^- \rvert$
Quickest Transshipment	$\lvert S^+ \rvert$ or $\lvert S^- \rvert$	$\lvert S^+ \cup S^- \rvert$

Table 4.2: The running time improvement factors we achieve for quickest transshipment and transshipment over time problems in a dynamic network $\mathcal{N}$ with terminals $S^+ \cup S^-$ by Algorithm 15

	single source *or* single sink	**multiple sources *and* sinks**
Transshipment Over Time	$\lvert S^+ \cup S^- \rvert^2$	$\lvert S^+ \cup S^- \rvert^2$
Quickest Transshipment	$\lvert S^+ \rvert^2$ or $\lvert S^- \rvert^2$	$\lvert S^+ \cup S^- \rvert$

The, until now, best and only known algorithm to solve transshipment over time problems is the algorithm of Hoppe and Tardos, which in the worst case relies on $2|S^+ \cup S^-|$ many parametric submodular function minimizations and one lexicographically maximum flow over time

computation. The fastest known worst case running time that can be achieved for this algorithm is

$$\mathcal{O}\big(2k \cdot \mathcal{C}_T(\mathrm{SFM}_{\mathrm{Lee}}) \cdot (k^3 \log k \cdot \mathrm{MCF}(n,m) + k^4 \log^{\mathcal{O}(1)} k)\big).$$

We already argued in Observation 3.9 that pairing $\mathrm{SFM}_{\mathrm{Lee}}$ with parametric search leads to an increase of the worst case running time of at least $k^3 \log k$. Thus, we obtain the following fact:

Fact 4.23. *Our implementation of Algorithm 13 improves upon the algorithm of Hoppe and Tardos for solving a given transshipment problem $(\mathcal{N}, b, T)$ by* ***at least a factor of*** $S^+ \cup S^-| \cdot \log(|S^+ \cup S^-|)|$.

Fact 4.24. *Algorithm 15 improves upon the algorithm of Hoppe and Tardos for solving a given transshipment problem $(\mathcal{N}, b, T)$ by* ***at least a factor of*** $|S^+ \cup S^-|^2 \cdot \log(|S^+ \cup S^-|)$.

In order to solve a given quickest transshipment problem $(\mathcal{N}, b)$, we at first need to determine the minimal feasible time horizon T^* and afterwards the actual transshipment needs to be computed. With Algorithm 15 we have two possibilities to solve a quickest transshipment problem in strongly polynomial time: On the one hand we can pair the algorithm of Orlin with parametric search to determine T^*. By our results from above, while determining T^*, in this case also a suitable convex combination of lexicographically maximum flows over time solving $(\mathcal{N}, b, T^*)$ is computed.
A faster way is to first determine T^* by pairing the algorithm of Lee et al. [LSW15] with parametric search and by afterwards solving the problem $(\mathcal{N}, b, T^*)$ using Algorithm 15. Compared to the algorithm of Hoppe and Tardos, we then need a factor of $|S^+ \cup S^-|$ less parametric submodular function minimizations which results in a running time improvement of at least a factor of $|S^+ \cup S^-|$. We achieve the same running time improvement factor for our implementation of Algorithm 13. However, Algorithm 15 is more efficient because doing one submodular function minimization with the algorithm of Orlin is faster than doing $|S^+ \cup S^-|^3$ many submodular function minimizations with the algorithm of Lee et al. [LSW15].

Corollary 4.25. *A given quickest transshipment problem $(\mathcal{N}, b)$ can be solved by first computing T^* using SFM_{Lee} paired with parametric search and by afterwards solving $(\mathcal{N}, b, T^*)$ with the help of Algorithm 15. The running time improvement we achieve is* ***at least a factor of*** $|S^+ \cup S^-|$. *We achieve the same running time improvement factor for our implementation of Algorithm 13. However, Algorithm 15 is the faster of both of our algorithms.*

When considering a quickest transshipment problem $(\mathcal{N}, b)$ in a dynamic network with only a single source *or* only a single sink, we can use our results from Section 4.1 to achieve an even larger running time improvement. We know that in these special cases we can determine the minimal feasible time horizon T^* with the help of Algorithm 12 or Algorithm 11, which require overall only $|S^+ \cup S^-|$ many submodular function minimizations. That is, in this case we can, with the help of Algorithm 15, solve the quickest transshipment problem in the same asymptotic running time as the algorithm of Orlin for submodular function minimization. Our implementation of Algorithm 13 still requires in the worst case k^3 many, for $k = |S^+|$ or $k = |S^-|$, submodular function minimizations using the algorithm of Lee et al. [LSW15]. Thus, for this algorithm we can in this case only deduce a running time improvement of at least a factor of $|S^+|$ or $|S^-|$.

Corollary 4.26. *A quickest transshipment problem $(\mathcal{N}, b)$ in a dynamic network with only a single source or only a single sink can be solved in the same asymptotic running time as the algorithm of Orlin for submodular function minimization [Orl93]. This results in an overall running time improvement upon the algorithm of Hoppe and Tardos* ***by at least a factor of*** $|S^+ \cup S^-|^2 \cdot \log(|S^+ \cup S^-|)|$.

Conclusions and Open Questions. Note that all our new algorithms for solving the quickest transshipment problem basically achieve (at least part of) the running time improvement compared to the algorithm of Hoppe and Tardos by trading the computation of an integral solution with the computation of a fractional quickest transshipment. Of course our algorithms significantly minimize the (parametrized) submodular function minimizations required to solve a quickest transshipment problem, but it remains an open questions how find an integral solution to a quickest transshipment problem with less submodular function minimizations than the algorithm of Hoppe and Tardos. One

approach could be to at first compute a convex combination of lex-max flows over time solving a given quickest transshipment problem and then to efficiently construct an integral quickest transshipment out of this convex combination.

5 Earliest Arrival Transshipments in Networks with a Single Sink

Earliest Arrival Transshipments are quickest transshipments that feature particularly desirable properties in the context of evacuation planning. An earliest arrival transshipment – which in general does only exist in dynamic networks with a single sink – is a quickest transshipment maximizing the amount of flow which has reached the sink for every point in time simultaneously. The so far only known algorithm for this problem that does not rely on explicit time expansion is due to Baumann and Skutella [Bau07]. However, in the worst case their algorithm still requires an exponential expansion of the original dynamic network. The main result of this chapter is a novel algorithm for the earliest arrival transshipment problem that solely works on the given network and, as a consequence, only requires polynomial space.
Additionally, we present a faster algorithm for computing the earliest arrival pattern. The currently best known algorithm for this task requires number of sources many parametric submodular function minimizations. We improve upon this by getting rid of the parametrization in the submodular function minimizations.
A consequence of our results is an FPTAS for approximating earliest arrival transshipments that does not require any form of time expansion.

Publication Remark: Some of the results from this chapter have been published in [SS17b].

Contents

When disaster strikes, one goal of an efficient evacuation strategy is of course to rescue all people from the endangered area as quickly as possible. To meet this need, quickest transshipments can be used. However, sometimes it might not be apparent when the actual tragedy will happen. For example, after an earthquake somewhere below the ocean, it might be known that at some point a tsunami will hit the coast, but not exactly when the wave will reach the shore. Only focusing on the aim to rescue all people until a fixed minimal time horizon could be fatal in such a situation because it could happen that the wave hits the coast before all people can be saved. In the worst case this will lead to an unnecessary high death toll. A more efficient strategy in such a scenario is to maximize the number of rescued people for each point in time simultaneously. This way it is ensured that, even if not all people can be rescued, at least the number of saved people is maximized. This property is captured by *earliest arrival transshipments*. In this chapter we focus on developing more efficient algorithms for the earliest arrival transshipment problem in dynamic networks with

only a single sink as in this case such transshipments do exist in general.
Recall, that the objective of an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network $\mathcal{N}$ with supplies/demands b is to compute a quickest transshipment f that satisfies all supplies and demands *and* has the additional property of being maximal at every point in time simultaneously. It is not inherently clear that such transshipments do always exist, and it turned out that for dynamic networks with multiple sinks they don't. This fact was first observed by Fleischer in [Fle01] and in [BS09] Baumann and Skutella give a simple counter example that is depicted in Figure 6.1. For the case of several sources but only a single sink, however, earliest arrival transshipments do always exist [RT]. This can be derived, for example, from the existence of lexicographically maximal flows in time-expanded networks, see [Min73; Meg74]. There are polynomial time algorithms known for solving the earliest arrival transshipment problem in dynamic networks with only a single sink for the special case that all transit times are zero [HO84; Fle01]. However, for general transit times, one cannot hope for a polynomial time algorithm because it was recently shown by Disser and Skutella that it is (already in dynamic networks with a single source and a single sink) $\mathcal{NP}$-hard to solve the earliest arrival flow problem [DS15]. So far, the best known algorithm for the earliest arrival transshipment problem in dynamic networks with a single sink is due to Baumann and Skutella [BS09]. Their algorithm has, however, although it does not rely on explicit time expansion, the huge disadvantage that it nevertheless requires a pseudo-polynomial expansion of the original network in the worst case. Finding an algorithm that only works on the original network without needing any form expansion is thus a desirable goal, which we achieve by our main results from this chapter.
The algorithm of Baumann and Skutella for solving the earliest arrival transshipment problem works in two phases. In the first phase the earliest arrival pattern corresponding to a given problem is derived, and in the second phase the breakpoints of the pattern are used to compute the actual earliest arrival transshipment. Our algorithm also requires these two steps, but we improve both of them.
Relying on our polynomial space algorithm for the earliest arrival transshipment problem in dynamic networks with only a single sink we also develop a new FPTAS for the approximation of such flows over time. Approximating earliest arrival transshipments is in particular interesting as it is unlikely that a polynomial time algorithm for computing such flow does exist (due to the $\mathcal{NP}$-hardness). The so far only known FPTAS for the earliest arrival transshipment problem is due to Fleischer and Skutella [FS07]. Their algorithm relies on a special geometric condensation of the time-expanded network. In contrast to their algorithm our FPTAS does not require any form of time expansion, but only works on the underlying original dynamic network.

Computation of the Earliest Arrival Pattern. The first part of our algorithm for solving the earliest arrival transshipment problem also requires computing (part of) the earliest arrival pattern. However, compared to the algorithm of Baumann and Skutella, the amount of information regarding the structure of the pattern that our algorithm requires is somewhat rougher.
In particular, we do not need to compute all the breakpoints of the earliest arrival pattern, which is the part which prevents the pattern computation needed in the algorithm by Baumann and Skutella from being of strongly polynomial running time. The information about the structure of the earliest arrival pattern we need can be computed in strongly polynomial running time. In Section 5.1 we also provide an improved algorithm for computing this reduced amount of pattern information. The so far best known algorithm for computing the part of the earliest arrival pattern that we need, needs, in the worst case, number of sources many parametric submodular function minimizations. The most efficient way to solve them is using the submodular function minimization algorithm of Lee et al. [LSW15] coupled with Megiddo's parametric search. Using the same approach as in our improved algorithm for determining the minimal feasible time horizon of a quickest transshipment problem we present a way to compute the pattern information that does not need an oracle for *parametric* submodular function minimization, but only needs to solve submodular function minimization problems that are *not* parametric. This results in a huge running time improvement compared to the classical approach. The main feature that our algorithm exploits is again the fact that the parametric submodular functions that occur in the context of the pattern computation are related by the *strong map property*.

Computation of Earliest Arrival Transshipments. The main result of this chapter, presented in Sections 5.2 and 5.3, is an algorithm for computing earliest arrival transshipments that only requires

polynomial space. We follow a similar strategy as for our new algorithm for solving the quickest transshipment problem in the previous chapter. For the quickest transshipment problem we showed that it can be solved by a convex combination of lex-max flows over time. This result strongly relies on a correspondence between the vertices of a certain base polytope and lex-max flows over time. Our algorithm for the earliest arrival transshipment problem is of a similar flavor. However, instead of considering only one base polytope, we define a more complicated polytope which is the product of at most number of sources many different base polytopes of suitably defined submodular functions. The precise definition of this polytope for a given earliest arrival transshipment problem strongly depends on the specific structure of its earliest arrival pattern. We will show that the vertices of this polytope correspond to special generalization of lex-max flows over time. This class of flows over time is defined in Section 5.1, in which we also give a strongly polynomial time algorithm to compute such flows. Using the correspondence between the vertices of the suitably defined polytope and of the generalization of lex-max flows over time we can deduce the structural result that earliest arrival transshipment problems in dynamic networks with only a single sink can always be solved by a convex combination of these flows (see Section 5.3.1). Furthermore, we will deduce that a suitable convex combination can essentially be computed while computing the needed amount of information regarding the earliest arrival pattern (see Section 5.3.2). This approach yields a PSPACE algorithm for solving earliest arrival transshipment problems that also has the nice property that, using our results from Section 5.1, only number of sources many submodular function minimizations are required. The downside of this algorithm is that it does not compute integral earliest arrival transshipments like the algorithm of Baumann and Skutella. In Section 5.3.3 we present an adaptation of the algorithm of Hoppe and Tardos for the quickest transshipment problem to compute integral earliest arrival transshipments in polynomial space. In contrast to our other algorithm, this algorithm requires number of sources many *parametric* submodular function minimization and is thus a lot less efficient. In both algorithms the computed output (the earliest arrival transshipment) requires necessarily a pseudo-polynomial amount of space, but the output is computed sequentially during the course of the algorithms such they indeed only need polynomial space.

Approximation of Earliest Arrival Transshipments Relying on one of our polynomial space algorithm for the earliest arrival transshipment problem we develop a strongly polynomial time algorithm for computing so-called multiple deadline transshipments. Multiple deadline transshipments are a generalization of earliest arrival transshipments – they are quickest transshipments with the property that they are maximal at finitely many points in time that were given in the input of the problem. Using our algorithm for computing multiple deadline transshipments over time, we are able to develop an FPTAS for earliest arrival transshipment problems that does not need any form of time expansion.

5.1 Faster Computation of the Earliest Arrival Pattern

The main result of this section is an improved algorithm for computing the earliest arrival pattern of a given earliest arrival transshipment problem in a dynamic network with a single sink.
During this section let $\mathcal{N} = (D = (V, A), u, \tau, S^+, t)$ be a dynamic network with multiple sources S^+ and a single sink t and $b\colon S^+ \cup \{t\} \to \mathbb{Z}$ a supply/demand function on the terminals. Recall, that by $(\mathcal{N}, b)_{\text{EAT}}$ we denote the corresponding earliest arrival transshipment problem, which is the problem of computing a flow over time f in $\mathcal{N}$ that is maximal at each point in time under the constraint that the supplies and demands need to be respected. The earliest arrival pattern $p^*\colon [0, T) \to \mathbb{R}_{\geq 0}$ corresponding to $(\mathcal{N}, b)_{\text{EAT}}$ is a function with the property that $p(\theta)$ is the maximum amount of flow that can be sent from the sources in S^+ to the sink t in the dynamic network $\mathcal{N}$ until time θ, under the constraint that all supplies and demands are respected. Here T is the minimal feasible time horizon of $(\mathcal{N}, b)_{\text{EAT}}$. Knowing the whole earliest arrival pattern (which is a piecewise linear function) is an essential ingredient of the algorithm of Baumann and Skutella [BS09]. For our approach to solve earliest arrival transshipment problems in dynamic networks with only a single sink it will also be necessary to compute some structural information about the earliest arrival pattern, however, we do not need to compute all the break points of this pattern.
Recall that computing the earliest arrival pattern p^* corresponding to a given earliest arrival

transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ amounts in computing a partition of S^+, $S^+ = S_1 \sqcup S_2 \sqcup \ldots \sqcup S_r$ and times $\theta_1 < \theta_2 < \ldots < \theta_r$ with the property that in an earliest arrival transshipment the sources in S_i have to run empty until time θ_i (i.e., their overall amount of flow has arrived at the sink t at time θ_i) for all $i \in \{1, 2, \ldots, r\}$, even if they sent as little flow as possible without violating the earliest arrival property. The second ingredient to obtain the whole structure of the earliest arrival pattern is to determine all of the break points of the piecewise linear function $\theta \mapsto o^\theta(S^+ \setminus (S_1 \cup \ldots \cup S_i))$ on the interval $[0, \theta_i)$, for all $i \in \{1, \ldots, r\}$. These breakpoints, which might be exponentially many in the worst case, are needed in the algorithm of Baumann and Skutella to compute an earliest arrival transshipment. In particular, it is needed to attach one super-sink to the original dynamic network for each of the breakpoints, which in the worst case leads to an exponential expansion of the original network .
For our algorithm only the sets $S_1, \ldots, S_r$ and the times $\theta_1, \ldots, \theta_r$ are important. Our objective in this section is to make the computation of these sets and times more efficient. In Algorithm 16 we restate the algorithm for the pattern computation (see Algorithm 9) but without the explicit computation of the functions $\theta \mapsto o^\theta(S^+ \setminus (S_1 \cup \ldots \cup S_i))$, for all $i \in \{1, 2, \ldots, r\}$.

Algorithm 16: Algorithm for the earliest arrival pattern - Without computing the breakpoints

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, s, t)$ and a supply/demand function b on the terminals , defining an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$

Output : The sets $S_1, S_2, \ldots, S_r$ and times $\theta_1 < \theta_2 \ldots < \theta_r$ corresponding to the earliest arrival pattern p^* of $(\mathcal{N}, b)_{\text{EAT}}$

1 $i \leftarrow 0$, $S_i \leftarrow \varnothing$, $\theta_i \leftarrow 0$

2 **while** $S_1 \cup \ldots \cup S_i \neq S^+$ **do**

3 Compute the maximal value $\theta_{i+1} \geq 0$ such that

$$o^{\theta_{i+1}}\left(S^+ \setminus \left(\bigcup_{k=1}^{i} S_k\right)\right) - o^{\theta_{i+1}}\left(S^+ \setminus \left(\bigcup_{k=1}^{i} S_k \cup S\right)\right) \leq b(S) \text{ for all } S \subseteq S^+ \setminus \left(\bigcup_{k=1}^{i} S_i\right)$$

4 Compute an inclusion-wise maximal $S_{i+1} \subsetneq S^+ \setminus \left(\bigcup_{k=1}^{i} S_i\right)$ with

$$o^{\theta_{i+1}}\left(S^+ \setminus \left(\bigcup_{k=1}^{i} S_k\right)\right) - o^{\theta_{i+1}}\left(S^+ \setminus \left(\bigcup_{k=1}^{i+1} S_k\right)\right) = b(S_{i+1})$$

5 $i \leftarrow i + 1$

6 **end**

7 $r \leftarrow i$

8 **return** $S_1, S_2, \ldots, S_r$ *and* $\theta_1, \theta_2, \ldots, \theta_r$

When we in the following speak about "computing the earliest arrival pattern" corresponding to a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$, we mean computing the sets S_i and times θ_i as presented in Algorithm 16.
Similar to the faster algorithm for computing the minimal feasible time horizon T of a given quickest transshipment problem, which we presented in Section 4.1, our new algorithm for computing the earliest arrival pattern strongly relies on the fact that the parametric submodular functions that occur during the course of the pattern computation fulfill the *strong map property*. Note that for a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ the pattern computation consists of at most $|S^+|$ many parametric submodular function minimizations.

Observation 5.1. *For a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ the pattern computation as presented in Algorithm 16 consists of one parametric submodular function minimization per iteration: In each iteration i the maximal value θ_{i+1} with the property*

$$o^{\theta_{i+1}}\left(S^+ \setminus \left(\bigcup_{k=1}^{i} S_k\right)\right) - o^{\theta_{i+1}}\left(S^+ \setminus \left(\bigcup_{k=1}^{i} S_k \cup S\right)\right) \leq b(S) \text{ for all } S \subseteq S^+ \setminus \left(\bigcup_{k=1}^{i} S_i\right) \quad (5.1)$$

needs to be determined and an inclusion-wise maximal set S_{i+1} for which equality holds. We can solve each of these problems by coupling the algorithm of Lee et al [LSW15] with parametric search.

As our main result in this section we will show how to solve all of these parametric submodular function minimization problems by at most $|S^+|^2$ many submodular function minimizations, which results in an overall running time improvement.
To achieve this goal we proceed completely similar to what we did in Section 4.1. However, in order to be able to use similar methods, we need to slightly redefine the submodular functions occurring during the pattern computation.
Let U be some upper bound for the minimal feasible time horizon T of the quickest transshipment problem $(\mathcal{N}, b)$. An upper bound for T is for example obtained by

$$SP^+ + \frac{B^+}{\min_{a\in A} u_a}$$

where $B^+ \coloneqq \sum_{s\in S^+} b(s)$ and

$$SP^+ \coloneqq \max\{d(\mathcal{N}, s, t) \mid s \in S^+, t \in S^-\}.$$

A proof of this fact can be found in [Kap14]. Of course, we could also just compute the minimal feasible time horizon T of $(\mathcal{N}, b)_{\text{EAT}}$ with the help of Algorithm 11. However, T is also computed during the pattern computation and thus computing the minimal feasible time horizon beforehand would be an unnecessary effort. In order to simplify notation, we define for each iteration i of Algorithm 16

$$C_i \coloneqq S^+ \setminus \bigcup_{k=1}^{i-1} S_k.$$

Further, for each $X \subseteq S^+$, the set function $g_X^\theta \colon 2^X \to \mathbb{R}$ is defined by,

$$S \mapsto o^{U-\theta}(S) - o^{U-\theta}(X \setminus S) + b(S),$$

for all $0 \le \theta \le U$. The set function g_X^θ is clearly submodular for all $X \subseteq \mathbb{R}^+$ and all $\theta \in [0, U)$. The following lemma is central throughout this section.

Lemma 5.2. *In iteration i of Algorithm 16, instead of computing the maximal value $\theta_{i+1} \ge 0$ with the property that*

$$o^{\theta_{i+1}}(C_{i+1} \setminus S) - o^{\theta_{i+1}}(C_{i+1}) + b(S) \ge 0 \text{ for all } S \subseteq C_{i+1}$$

and determining a maximal set S_{i+1} with equality, we can equivalently also solve the parametric submodular function minimization problem $(g^\theta_{C_{i+1}}, \mathbf{0})$ which returns a time γ_{+1} and a maximal minimizing set X_{i+1}. Then $\theta_{i+1} = U - \gamma_{i+1}$ and $S_{i+1} = X_{i+1}$.

Proof. Let $\theta \in [0, U]$ and define $\gamma = U - \theta$. We have by assumption

$$\begin{aligned}\min_{S\subseteq C_{i+1}} \{-o^\theta(C_{i+1}) + o^\theta(C_{i+1} \setminus S) + b(S)\} &= \min_{S\subseteq S_{i+1}} \{-o^{U-\gamma}(C_{i+1}) + o^{U-\gamma}(C_{i+1} \setminus S) + b(S)\} \\ &= \min_{S\subseteq S_{i+1}} g^\gamma_{C_{i+1}}(S).\end{aligned}$$

This shows the relation between θ_{i+1} and γ_{i+1}. That $S_{i+1} = X_{i+1}$ holds follows immediately. □

Thus, in order to compute the earliest arrival pattern, we can also solve the parametric submodular function minimization problem $(g^\theta_{C_{i+1}}, \mathbf{0})$ in iteration i of Algorithm 16. The most important property that we exploit in order to solve this problem in a more efficient manner is that for $\eta_1 \le \eta_2 \le U$ the functions $g_X^{\eta_1}$ and $g_X^{\eta_2}$ are related by the strong map property for all $X \subseteq S^+$.

Lemma 5.3. *For all $X \subseteq S^+$ and $\eta_1, \eta_2 \in [0, \infty)$ with $\eta_1 \leq \eta_2$, we have*

$$g_X^{\eta_1} \leftarrow g_X^{\eta_2}.$$

Proof. Let $Y \subseteq Z \subseteq X$. Lemma 4.2 implies

$$\begin{aligned} g_X^{\eta_1}(Z) - g_X^{\eta_1}(Y) &= o^{U-\eta_1}(X \setminus Z) - o^{U-\eta_1}(X \setminus Y) + b(Z \setminus Y) \\ &\overset{\text{Lem. 4.2}}{\leq} o^{U-\eta_2}(X \setminus Z) - o^{U-\eta_2}(X \setminus Y) + b(Z \setminus Y) \\ &= g_X^{\eta_2}(Z) - g_X^{\eta_2}(Y). \end{aligned}$$

□

Our Algorithm 17 to solve $(g_X^\theta, \mathbf{0})$ is now completely similar to Algorithm 11 in Section 4.1. The proof that Algorithm 17 works correctly is also completely analogous to the proof of Theorem 4.8.

Algorithm 17: An algorithm that solves the parametric submodular function minimizations needed to compute the earliest arrival pattern more efficiently

```
Input  : A dynamic network 𝒩 = (D = (V, A), u, τ, S⁺, t), a supply/demand function b, X ⊆ S⁺
Output : The minimal time θ with g_X^θ(S) ≥ 0 for all S ⊆ X
i ← 0, X_i ← X, λ_i ← 0
while X_i ≠ ∅ do
    λ_{i+1} ← Minimal value with g_{X_i}^{λ_{i+1}}(X) = 0
    X_{i+1} ← Maximal minimizer of g_{X_i}^{λ_{i+1}}
    i ← i + 1
end
return θ_i = U − λ_i, X_i
```

Theorem 5.4 (Correctness of Algorithm 17). *For each $X \subseteq S^+$ the Algorithm 17 works correctly, i.e., it correctly solves $(g_X^\theta, \mathbf{0})$. The returned time θ^* and the set X^* fulfill $g_X^{U-\theta^*}(S) \geq 0$ for all $S \subseteq S^+$ and X^* is an inclusion-wise maximal set with $g_X^{U-\theta^*}(X^*) = 0$. The algorithm finishes after at most $|X|$ iterations.*

Proof. We refer to the proof of Theorem 4.8 in Section 4.1. □

It remains to deduce the overall running time of Algorithm 17. In iteration i we at first need to determine the minimal value λ_i with

$$\begin{aligned} g_X^{\lambda_i}(X_{i-1}) &= o^{U-\lambda_i}(X \setminus X_{i-1}) - o^{U-\lambda_i}(X) + b(X_{i-1}) \\ &= 0. \end{aligned}$$

Recall, that o^θ can be evaluated at each subset $X \subseteq S^+$ by doing one minimum-cost flow computation in a network in which only the cost of one arc is parametrized by θ (see Fact 2.21). Computing the minimal value λ_i with $g_{X_i}^{\lambda_i}(X_{i-1}) = 0$ is thus the same as solving a parametric minimum-cost flow problem. Burkhard [BDK93] deduced that we can compute λ_i within a worst case running time in $\mathcal{O}(m^2 \log^3 n(m + n \log n))$ by the minimum-cost flow algorithm of Orlin coupled with parametric search. More precisely, coupling Orlin's minimum-cost flow algorithm with parametric search, requires a worst case running time of $\mathcal{O}(\mathcal{C}_\lambda(\mathrm{MCF}(n, m)) \cdot \mathrm{MCF}(n, , m))$. The overall worst case running time of Algorithm 17 thus amounts in the running time of at most $|X|$ submodular function minimization plus $\mathcal{O}(|X| \cdot \mathcal{C}_\lambda(\mathrm{MCF}(n, m)) \cdot \mathrm{MCF}(n, m))$.

Corollary 5.5. *For each $X \subseteq S^+$ the worst running time of Algorithm 17 is in*

$$\mathcal{O}\big(|X|(\mathrm{SFM}_{Lee} + \mathcal{C}_\lambda(\mathrm{MCF}(n, m)) \cdot \mathrm{MCF}(n, m))\big).$$

When applying this to the pattern computation in Algorithm 16, this result implies that instead of doing $|S^+|$ many parametric submodular function minimizations, in the worst case, we also can compute the earliest arrival pattern by doing do $|S^+|^2$ many submodular function minimizations plus $|S^+|^2$ many minimum-cost flow computations coupled with Megiddo's search framework. The overall running time of the old approach is, by Observation 3.9, in

$$\mathcal{O}\big(k \cdot \big(\mathcal{C}_\lambda(\mathrm{SFM}_{\mathrm{Lee}}) \cdot (k^3 \log k \cdot \mathrm{MCF}(n,m) + k^4 \log^{\mathcal{O}(1)} k)\big)\big),$$

with $k = |S^+|$ and $\mathcal{C}_\lambda(\mathrm{SFM}_{\mathrm{Lee}}) \notin \mathcal{O}(k^{3-\varepsilon} \log k \mathcal{C}_\lambda(\mathrm{MCF}(n,m)))$ for all $\varepsilon > 0$. Whereas the running time of our new approach is

$$\mathcal{O}\big(k^2 \cdot (\mathrm{SFM}_{\mathrm{Lee}} + \mathcal{C}_\lambda(\mathrm{MCF}(n,m)) \cdot \mathrm{MCF}(n,m)\big).$$

This shows that with our new approach we obtain a running time improvement of a factor of at least k^2. Throughout this thesis, our method of choice for determining the earliest arrival pattern of a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\mathrm{EAT}}$ will be Algorithm 16 coupled with Algorithm 17. Note that we do not get rid of parametric search completely. We still couple it with an instance of a parametric minimum-cost flow problem. Maybe it is possible to solve the required parametric minimum-cost flow problems efficiently without relying on parametric search.

5.2 Generalizing Lexicographically Maximum Flows Over Time

Throughout this section we assume that $\mathcal{N} = (D = (V, A), u, \tau, S^+, t)$ is a dynamic network with only a single sink t, and b a supply/demand function on the terminals of $\mathcal{N}$. In such a setting there always exists an earliest arrival transshipment solving $(\mathcal{N}, b)_{\mathrm{EAT}}$ (See Section 3.2.2). During the computation of the earliest arrival pattern corresponding to $(\mathcal{N}, b)_{\mathrm{EAT}}$, a partition $S^+ = S_1 \sqcup S_2 \sqcup \ldots \sqcup S_r$ of S^+ and corresponding times $\theta_1 < \theta_2 < \ldots < \theta_r$ are computed that for all $i \in \{1, \ldots, r\}$ have the property that in an earliest arrival transshipment the sources in S_i have to have sent their entire supply to the sink before time θ_i even if they send as little flow as possible while still ensuring the earliest arrival property. This motivates the definition of generalized lexicographically maximum flows over time: On the one hand the partition $S_1 \sqcup S_2 \sqcup \ldots \sqcup S_r = S^+$ induces a linear order on the terminals in which the sources in S_i are ordered before the sources in S_j if $i > j$ for $i, j \in \{1, \ldots, r\}$. On the other hand, each set of sinks S_i has to respect a different time horizon θ_i for all $i \in \{1, 2, \ldots, r\}$. **Generalized lex-max flows over time** are generalizations of lex-max flows over time incorporating these properties.

5.2.1 Basic Properties of Generalized Lexicographically Maximum Flows Over Time

We start with the definition of generalized lexicographically maximum flows over time.

Definition 5.6 (Generalized Lex-Max Flow Over Time).
Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, t)$ be a dynamic network with only a single sink t, $S^+ = S_1 \sqcup S_2 \sqcup \ldots \sqcup S_r$, and $0 < \theta_1 < \theta_2 < \ldots < \theta_r$ rational times. We consider a linear order $\prec$ on S^+ with the following property:

- *The order $\prec$ **respects the given partition** $\prec$, i.e., $s \prec s'$ for all $s \in S_i$ and $s' \in S_j$ with $i > j$ for $i, j \in \{1, \ldots, r\}$*

*A **generalized lexicographically maximum (lex-max) flow over time** f with respect to $S_1, \ldots, S_r$, $\theta_1, \theta_2, \ldots, \theta_r$, and $\prec$ is a feasible flow over time that fulfills the following properties:*

1. *For $i = 1, 2, \ldots, r$, let $\mathcal{N}_i$ be the dynamic network obtained by attaching a new super-source ψ_i to the sources in $S_{i+1} \cup S_{i+2} \cup \ldots \cup S_r$ by arcs with zero transit time and infinite capacity. For all $i \in \{1, \ldots, r\}$ denote by $\prec_i$ the total order on $\{\psi_i\} \cup S_i \cup \{t\}$ that is the*

restriction of $\prec$ to S_i that fulfills $\psi_i \prec s \prec t$ for all $s \in S_i$. We require that the flow sent by f out of $\{\psi_i\} \cup S_i$ into the sink t until time θ_i is a lex-max flow over time with respect to $\prec_i$ and time horizon θ_i, for $i = 1, 2, \ldots, r$. Note that we can regard the flow over time f (which is a flow in $\mathcal{N}$ by definition) as a flow over time in $\mathcal{N}_i$ by assuming that the flow sent out of the sources in $S_{i+1} \cup \ldots \cup S_r$ is sent out of ψ_i in $\mathcal{N}_i$. This works, because the outgoing arcs of ψ_i all have zero transit time and infinite capacity.

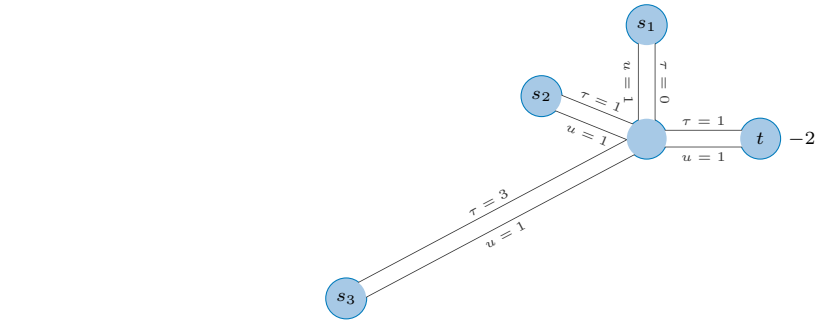
2. *For all $i = 1, 2, \ldots, r$, after time θ_i no flow originating from S_i remains in the network and the sources in S_i do not send new flow into the networks.*

A generalized lex-max flow over time can thus imagined to be a lex-max flow over time in which the sources have different time horizons that respect the given order i.e., the lower the time horizon, the lower is also the priority of the corresponding sources. See Figure 5.1 for an example of a generalized lex-max flow over time. It is not immediately clear that generalized lex-max flows over time do exist for every choice of parameters. In the following we give a proof of their existence using the time-expanded network. Our proof is constructive and leads to an algorithm for computing generalized lex-max flows over time using time expansion. However, this result is not satisfactory for us as we want to use generalized lex-max flows over time for our PSPACE algorithm for solving earliest arrival transshipment problems. For this purpose we need an algorithm for computing generalized lex-max flows over time that requires only polynomial space. Such an algorithm is presented in the following section. Before we get to the proof of the existence of generalized lex-max flows over time, we need to recap some properties of the time-expanded network and make some slight modification to the construction that we described in Section 2.5.2. Assume that $\mathcal{N}$ is a dynamic network with only a single sink and that we are given a partition of S^+ into r disjoints subsets, $S^+ = S_1 \sqcup S_2 \sqcup \ldots \sqcup S_r$, and times $0 < \theta_1 < \theta_2 < \ldots < \theta_r = T$. Let $\prec$ be a total order on S^+ that respects the given partition. To simplify the notation we assume that all given times are integers. For rational times the construction we describe in the following works completely analogous. We consider the time-expanded network $\mathcal{N}^T$ corresponding to $\mathcal{N}$. The version of the time-expanded network we need here slightly differs from the construction we described in Section 2.5.2: Instead of attaching an overall super-sink t^* to the time-expanded network, we attach r different super-sinks $t_1, t_2, \ldots, t_r$, one for each time horizon that is interesting for our generalized lex-max flow over time. Recall, that $t^1, t^2, \ldots, t^T$ are the copies of the sink t in each of the time layers of the time-expanded network. Overall, we add super-sinks $t_1, \ldots, t_r$ and the following arcs with infinite capacity to the static network $\mathcal{N}^T$,

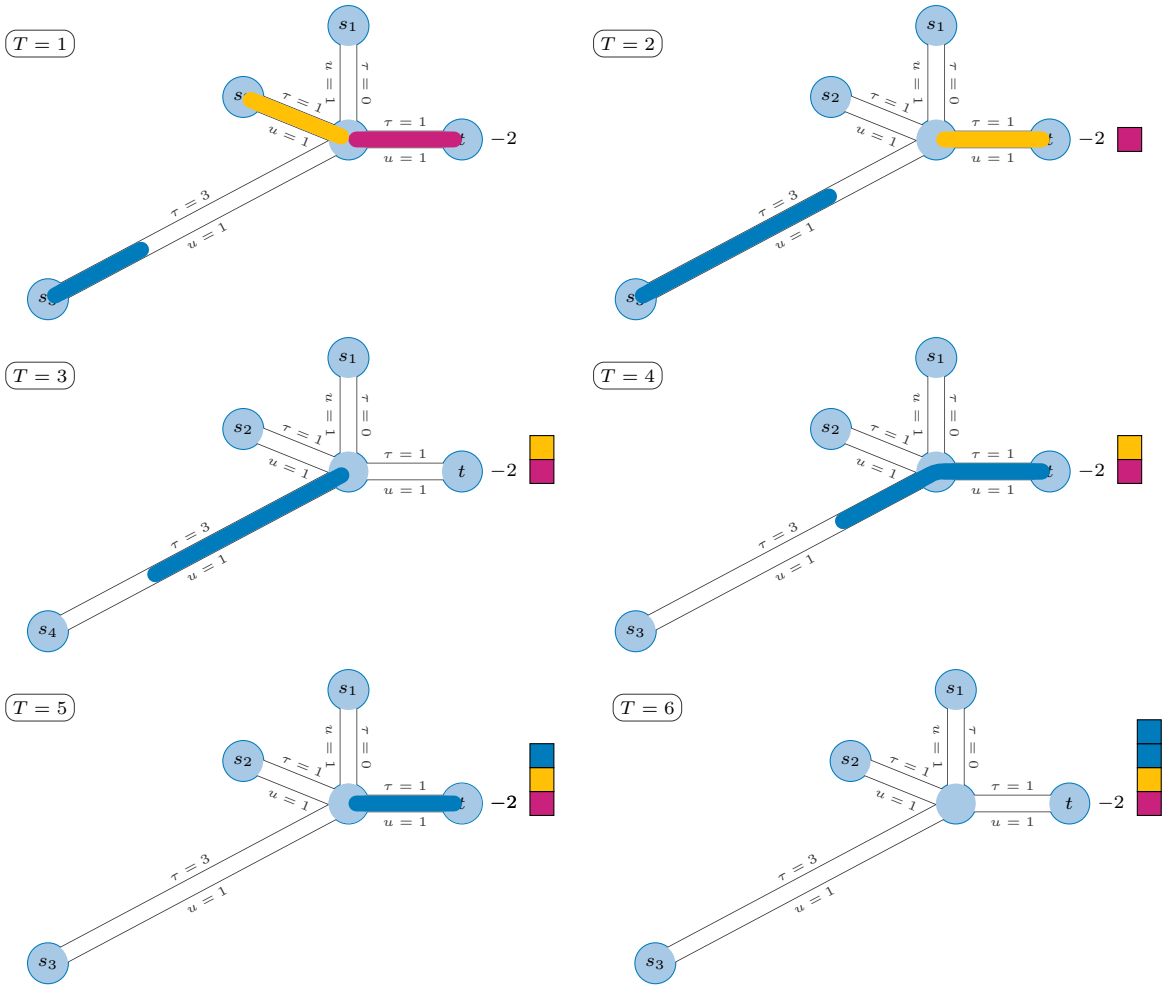
$$
\begin{aligned}
&(t^\theta, t_1) \text{ for } \theta \in \{1, 2, \ldots, \theta_1\},\\
&(t^\theta, t_2) \text{ for } \theta \in \{\theta_1 + 1, \theta_1 + 2, \ldots, \theta_2\},\\
&\qquad \ldots\\
&(t^\theta, t_r) \text{ for } \theta \in \{\theta_{r-1} + 1, \theta_{r-1} + 2, \ldots, \theta_r\}.
\end{aligned}
$$

The set of sinks of our redefined time-expanded network $\mathcal{N}^T$ is $\{t_1, t_2, \ldots, t_r\}$. We need these multiple sinks to mirror the time horizons $\theta_1, \ldots, \theta_r$. Furthermore, we also do not add an overall super-source s^* to the time-expanded network. Recall, that in the construction of the time-expanded network from Section 2.5.2 we added the nodes in S^+ to the time-expanded network by attaching every $s \in S^+$ to each of its copies in the layers of the time-expanded network. Afterwards we added an overall super-source s^* to the nodes in S^+. We skip adding s^* and consider S^+ as the set of sources of the redefined time-expanded network $\mathcal{N}^T$. The time-expanded network corresponding to the example in Figure 5.1 is shown in Figure 5.2. It turns out that a generalized lex-max flow over time in $\mathcal{N}$ with respect to the given parameters corresponds to a certain static lex-max flow with respect to a total order $\prec_T$ in this time-expanded network. Let $S_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,n_i}\}$, for $i \in \{1, 2, \ldots, r\}$. Assume our original order $\prec$ on S^+ is given by

$$
\begin{aligned}
&s_{r,n_r} \prec s_{r,n_r-1} \prec \ldots \prec s_{r,1}\\
&\prec s_{r-1,n_{r-1}} \prec s_{r-1,n_{r-1}-1} \prec \ldots \prec s_{r-1,1}\\
&\prec \ldots\\
&\prec s_{1,n_1} \prec s_{1,n_1-1} \prec \ldots \prec s_{1,1}.
\end{aligned}
$$



(a) A dynamic network $\mathcal{N}$ with sources s_1, s_2, s_3, $S_1 = \{s_1, s_2\}$, $S_2 = \{s_3\}$, $\theta_1 = 3$, $\theta_2 = 6$, and an order $\prec$ on S^+ given by $s_3 \prec s_2 \prec s_1$, i.e., the order respects the given partition of S^+. The goal is to find a generalized lex-max earliest arrival flow with respect to the given parameters.



(b) Until time $\theta_1 = 3$ the depicted flow over time behaves like an ordinary lex-max flow over time with respect to $\prec$. Note that after time θ_1 all flow originating from the the sources s_1 and s_2 has left the network, i.e., s_1 and s_2 respect the time horizon θ_1. The source s_3 sends as much flow as possible towards t until time $\theta_2 = 6$, that is the flow out of s_3 also is a lex-max flow over time with respect to $\prec$ restricted to s_3 and time horizon θ_2. Thus, the flow over time shown in these figures is in fact a generalized lex-max flow over time with respect to the given parameters.

Figure 5.1: Example of a generalized lex-max flow over time

Note that the order $\prec$ gives higher priorities to sources with a larger time horizon because the order $\prec$ is supposed to respect the given partition. We define a total order $\prec_T$ on the sources and sinks of $\mathcal{N}^T$, $S^+ \cup \{t_1, \dots, t_r\}$, as follows

$$\begin{aligned} & s_{r,n_r} \prec_T s_{r,n_r-1} \prec_T \dots \prec_T s_{r,1} \prec_T t_r \\ \prec_T\ & s_{r-1,n_{r-1}} \prec_T s_{r-1,n_r-2} \prec_T \dots \prec_T s_{r-1,1} \prec_T t_{r-1} \\ \prec_T\ & \dots \\ \prec_T\ & s_{1,n_1} \prec_T s_{1,n_1-1} \prec_T \dots \prec_T s_{1,1} \prec_T t_1. \end{aligned} \tag{5.2}$$

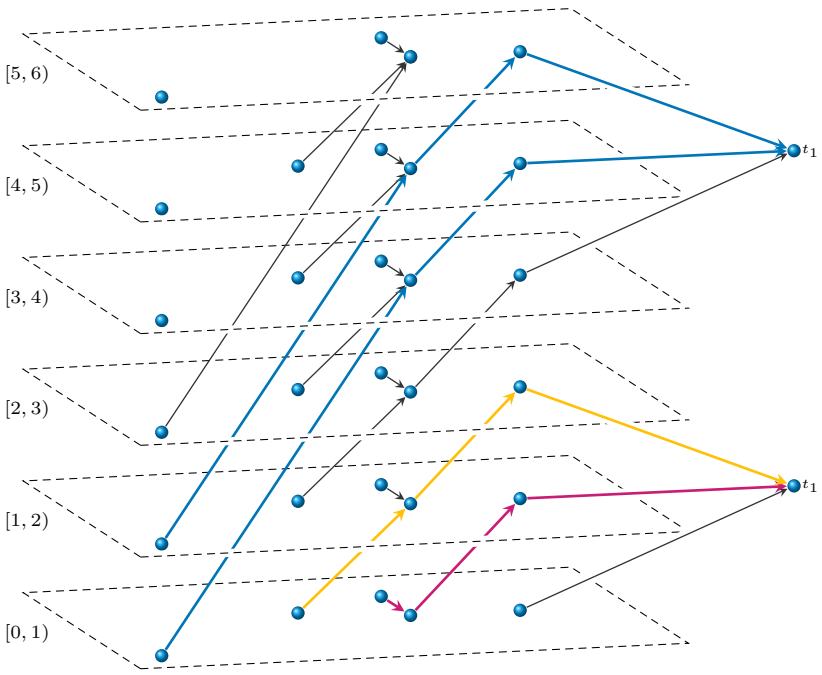


Figure 5.2: The time-expanded network corresponding to the generalized lex-max flow over time problem from Figure 5.1. In this problem the given time horizons are $\theta_1 = 3$ and $\theta_2 = 6$, hence the super-sink t_1 is connected to the first three layers of the network, while t_2 is connected to layers 4, 5 and 6. The super-sources s_1, s_2 and s_3 are not shown in the figure. They would be connected to their respective copies in each of the time layers. The indicated flow in the network is the static flow corresponding to the generalized lex-max flow over time as shown in Figure 5.1 (see Lemma 5.7).

We know that the corresponding static lexicographically maximum flow x in $\mathcal{N}^T$ does exist [Min73; Meg79]. Denote by f the flow over time in $\mathcal{N}$ with time horizon T constructed from x according to Lemma 2.20. We will show that f is a generalized lex-max flow over time with respect to the given parameters.

Lemma 5.7 (Correspondence of generalized lex-max flows and static lex-max flows). *Assume we are given a generalized lex-max flow over time problem by a dynamic network $\mathcal{N}$ with a single sink t, a partition of the sources S^+ into r disjoint subsets $S^+ = S_1 \sqcup S_2 \sqcup \ldots \sqcup S_r$, rational times $0 < \theta_1 < \theta_2 < \ldots < \theta_r = T$, and a suitable total order $\prec$ on S^+ that respects the given partition (see Definition 5.6). Let x be the static lex-max flow with respect to $\prec_T$ in $\mathcal{N}^T$ as defined in (5.2). The corresponding flow over time f is a generalized lex-max flow over time with respect to the given parameters. In particular, generalized lex-max flows over time do always exist and can be computed in the time-expanded network.*

Proof. We have to show that the flow over time f fulfills all the properties required of a generalized lex-max flow over time. In order to simplify the notation during the proof, assume that the times $\theta_1, \ldots, \theta_r$ are integers. For rational times the proof works completely analogously. At first note that by construction of the time-expanded network flow arriving at the sink t_i in $\mathcal{N}^T$ translates to flow arriving during $[\theta_{i-1}, \theta_i)$ in the flow over time f for all $i \in \{1, \ldots, r\}$. By Lemma 2.19 the sources in the set S_i only send flow towards the sinks $t_i, t_{i-1}, \ldots, t_1$ in the static lex-max flow x for all $i \in \{1, \ldots, r\}$. Hence, for all $i \in \{1, \ldots, r\}$ in the flow over time f all the flow originating from the sources in S_i arrives at the sink t before time θ_i and also the sources in S_i will not send any more flow into the network after time θ_i. This implies that the flow over time f fulfills the second property from Definition 5.6 – it respects the time horizon θ_i for each $i \in \{1, \ldots, r\}$. It remains to be shown that also the first property is fulfilled, i.e., we need to show that at time θ_i the flow that has arrived at t originating from $S_i \cup \{\psi\}$ respects the order $\prec_i$ on $\{\psi_i\} \cup S_i \cup \{t\}$, for all $i \in \{1, 2, \ldots, r\}$. For this purpose fix some $i \in \{1, 2, \ldots, r\}$. In the static flow x we are thus interested in the amount of flow sent from $S_i \cup \{\psi_i\}$ towards the sinks $t_1, t_2, \ldots, t_i$. By Lemma 2.19 we know that for each $j \in \{1, 2, \ldots, n_i\}$ the sources in $S_r \cup S_{r-1} \cup \ldots \cup S_{i-1} \cup \{s_{i,n_{r_i}}, s_{i,n_{r_i}-1}, \ldots, s_{i,j}\}$

sent as much flow as possible towards $\{t_1, t_2, \dots, t_i\}$ in the static flow x, i.e., the amount of flow that these sources sent is given by,

$$\max_{\mathcal{N}^T}(S_r \cup S_{r-1} \cup \dots \cup S_{i-1} \cup \{s_{i,n_{r_i}}, s_{i,n_{r_i}-1}, \dots, s_{i,j}\}, \{t_1, t_2, \dots, t_i\}).$$

By the definition of the function o^θ in the dynamic network $\mathcal{N}_i$ (as defined in the definition of generalized lex-max flows over time, Definition 5.6) and by the relation between the flow over time f in $\mathcal{N}_i$ and the static flow x in $\mathcal{N}^T$, this implies that in the flow over time f the amount of flow that has been sent to the sink t by $\{\psi_i\} \cup \{s_{i,n_{r_i}}, s_{i,n_{r_i}-1}, \dots, s_{i,j}\}$ until time θ_i is given by

$$o^{\theta_i}(\{\psi_i\} \cup \{s_{i,n_{r_i}}, s_{i,n_{r_i}-1}, \dots, s_{i,j}\}) \text{ for all } j \in \{1, 2, \dots, n_i\}.$$

By the same reasoning f has sent an amount of $o^{\theta_i}(\{\psi_i\})$ out of ψ_i towards t until time θ_i. Thus, by Lemma 3.11 the flow that has arrived at S_i until time θ_i in the flow over time f respects the given order $\prec$, which finishes the proof. □

The lex-max flow over time with respect to $\prec_T$ that yields a generalized lex-max flow over time for the problem shown in Figure 5.1 is indicated in Figure 5.2. From the definition of generalized lex-max flows over time we immediately can deduce the structure of their characteristic vectors using Lemma 2.19.

Lemma 5.8. *Assume we are given a generalized lex-max flow over time problem by a dynamic network $\mathcal{N}$ with a single sink t, a partition of S^+ into r disjoint subsets $S^+ = S_1 \sqcup S_2 \sqcup \dots \sqcup S_r$, rational times $0 = \theta_0 < \theta_1 < \theta_2 < \dots \theta_r$, and a total order $\prec$ on S^+ that respects the given partition of S^+. For a generalized lex-max flow over time f with respect to these parameters we have for all $i \in \{0, 1, \dots, r\}$,*

$$\begin{aligned} x_f(s) &= \operatorname{net}_f(s, \theta_i) \\ &= o^{\theta_i}\Big(\bigcup_{j=i+1}^{r} S_j \cup \{s' \in S_i \mid s' \preceq s\}\Big) - o^{\theta_i}\Big(\bigcup_{j=i+1}^{r} S_j \cup \{s' \in S_i \mid s' \prec s\}\Big), \end{aligned}$$

for all $s \in S_i$.

As already stated before, the algorithm for computing generalized lex-max flows over time that follows from Lemma 5.7 is not efficient as it uses the time-expanded network. In the following section we present a strongly polynomial time algorithm for computing generalized lex-max flows over time.

5.2.2 Computing Generalized Lexicographically Maximum Flows Over Time in Strongly Polynomial Time

In this section we describe a strongly polynomial time algorithm to compute generalized lex-max flows over time. A main ingredient for our algorithm will be the lexicographically maximum flow over time algorithm of Hoppe and Tardos that we described in Section 3.1.2. Before we start describing our algorithm, we will thus shortly recap the algorithm of Hoppe and Tardos for the special setting we need (i.e., dynamic networks with only a single sink and an order on the terminals that gives t the lowest priority).

Hoppe and Tardos' Algorithm for Lexicographically Maximum Flows over Time. Consider a dynamic network $\overline{\mathcal{N}}$ with a single sink t and let $\prec$ be a total order on S^+. Moreover, let $T \geq 0$ be a time horizon. Our goal is to compute a lex-max flow over time with time horizon T with respect to $\prec$ such that the sink t has the lowest priority.
In this case Hoppe and Tardos' algorithm essentially consists of two phases. In the first phase a maximum flow over time from S^+ to t is computed. The flow found in this phase does not yet respect the order $\prec$ or the time horizon T. Essentially, a temporally repeated flow with infinite time horizon is computed. In the second phase the flow originating from the sources in S^+ is rerouted such that it respects the order $\prec$ as well as the time horizon. This is done in $|S^+|$ many iterations.

The sources are considered in order of increasing priority, and each iteration makes sure that the specific source sends the correct amount of flow by rerouting flow to the other sources. For the source with the highest priority the previous iterations already made sure that this source sends the correct amount of flow up to time T. Thus, the last iteration only has the effect that, after this iteration, the flow out of the source with the highest priority respects the time horizon T. For our algorithm for the generalized lex-max flow over time problem we need a version of the algorithm of Hoppe and Tardos that skips this last iteration. A formal formulation of the algorithm of Hoppe and Tardos for our specific setting without the last iteration can be seen in Algorithm 18.

Algorithm 18: Hoppe and Tardos' lex-max flow over time algorithm without the last iteration, LexMaxSI($\mathcal{N}, \prec, T$)

Input : A dynamic network $\overline{\mathcal{N}} = (D = (V, A), u, \tau, S^-, t)$, a linear order $\prec$ on S^+ with $S^+ = \{s_1, s_2, \dots, s_k\}$ such that $s_1 \prec s_2 \prec \dots \prec s_k$, a time horizon $T \geq 0$

Output : A generalized temporally repeated flow respecting the given order $\prec$

$k \leftarrow |S^+|$
$V \leftarrow V \cup \{s\}$
$A^{k+1} \leftarrow A \cup \{(s, s') \mid s' \in S^+\} \cup \{t, s\}$
Extent u to A^{k+1} by defining $u_{(s,s')} := \infty$ for all $s' \in S^+$ and $u_{(t,s)} = \infty$
Extent τ to A^{k+1} by defining $\tau_{(s,s')} := 0$ for all $s' \in S^+$ and $\tau_{(t,s)} = -T$
$\overline{\mathcal{N}}^{k+1} \leftarrow ((V, A^{k+1}), u, \tau, s, t)$
$y^{k+1} \leftarrow$ minimum-cost circulation in $\overline{\mathcal{N}}^{k+1}$
Compute a path decomposition of the static flow y^{k+1} in $\overline{\mathcal{N}}^{k+1}$ given by $(\mathcal{P}^{k+1}, w^i)$
$Y^{k+1} \leftarrow \{(\mathcal{P}^{k+1}, w^{k+1})\}$
for $i \in \{k, k-1, \dots 2\}$ **do**
  $A^i \leftarrow A^{i+1}$
  $A^i \leftarrow A^i \setminus \{(s, s_i)\}$
  $\overline{\mathcal{N}}^i \leftarrow ((V, A^i), u, \tau, s, t)$
  $x^i \leftarrow$ Minimum-cost maximum s-s_i flow in $\overline{\mathcal{N}}^i_{y^{i+1}}$ with τ as costs
  $y^i \leftarrow y^{i+1} + x^i$
  Compute a path decomposition of the static flow y^i given by $(\mathcal{P}^i, w^i)$
  $Y^{i+1} \leftarrow Y^i \cup (\mathcal{P}^i, w^i)$
end
return y^1, Y^1

Description of our Algorithm. Up to time θ_1 a generalized lex-max flow over time is just a normal lex-max flow over time with respect to the order $\prec_1$ in the network $\mathcal{N}_1$ ($\prec_1$ and $\mathcal{N}_1$ are defined as in Definition 5.6), with the only difference that the time horizon θ_1 is only respected by the sources in S_1. Thus, this is exactly what is computed in the first iteration of our algorithm, which is described formally in Algorithm 19: a super-source ψ_1 is attached to the sources in $S^+ \setminus S_1$ and, with respect to $\prec_1$, a lex-max flow over time with time horizon θ_1 is computed. Here, we have to be careful. We cannot just apply the algorithm of Hoppe and Tardos for the lex-max flow over time problem, as this would give us a flow over time with time horizon θ_1. What we actually want is a flow over time such that only the sources in S_1 respect the time horizon θ_1. This is where the algorithm of Hoppe and Tardos with skipped last iteration comes into play. According to the description of this algorithm in the previous paragraph, in our setting this algorithm computes a flow over time such that all sources respect the time horizon θ_1, except for the super-source ψ_1. In fact, the flow out of this source still has infinite time horizon. The algorithm of Hoppe and Tardos with skipped last iteration is denoted by LexMaxSI. In the first iteration of our algorithm we thus apply this algorithm to the dynamic network $\mathcal{N}_1$ to make sure that the flow out of the sources in S_1 respects the given order and the time horizon θ_1. In the subsequent iterations the flow out of S_1 is not changed. In the second iteration we have to ensure that the sources connected to ψ_1 during the first iteration send their flow in a lex-max way with respect to $\prec_2$, and that the time horizon θ_2 is respected by the sources in S_2. This is again achieved with the help of Algorithm 18: The super-source ψ_1 is removed, and a new super-source ψ_2 is attached to the sources in $S^+ \setminus (S_2 \cup S_1)$.

Algorithm 19: Algorithm for the generalized lex-max flow over time, GENLEXMAX

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, t)$, a partition of S^+, $S^+ = S_1 \sqcup S_2 \sqcup \ldots \sqcup S_r$, times $\theta_r > \theta_{r-1} > \ldots > \theta_1$, a total order $\prec$ on $S^+ \cup \{t\}$ that respects the given partition of S^+

Output : A generalized lex-max flow over time with respect to the given parameters

$\{\psi_0\} \leftarrow \varnothing$, $z^0 \leftarrow 0$, $X^0 \leftarrow \varnothing$, $D^0 \leftarrow D$, $V^0 \leftarrow V$, $A^0 \leftarrow A$
for $i \in \{1, 2, \ldots, r-1\}$ **do**
  $V^i \leftarrow (V^{i-1} \setminus \{\psi_{i-1}\}) \cup \{\psi_i\}$
  $A^i \leftarrow A^{i-1} \cup \{(\psi_i, s) \mid s \in S_r \cup \ldots \cup S_{i+1}\}$
  Extend u to A^i by defining $u_{(\psi_i,s)} = \infty$ for all $s \in S_r \cup \ldots \cup S_{i+1}$
  Extend τ to A^i by defining $\tau_{(\psi_i,s)} = 0$ for all $s \in S_r \cup \ldots \cup S_{i+1}$
  $D^i \leftarrow (V^i, A^i)$, $S_i^+ \leftarrow S_i \cup \{\psi_i\}$
  $\mathcal{N}^i \leftarrow (D^i, u, \tau, S_i^+, t)$
  $\prec_i \leftarrow$ total order on $S_i^+ \cup \{t\}$ that restricts $\prec$ to S_i with $\psi_i \prec s \prec t$ for all $s \in S_i$
  $W^i, w^i \leftarrow \text{LEXMAXSI}(\mathcal{N}^i_{z^{i-1}}, \prec_i, \theta_i)$
  $X^i \leftarrow X^{i-1} \cup W^i$
  $z^i \leftarrow z^{i-1} + w^i$
end
$W^r, w^1 \leftarrow \text{LEXMAX}(\mathcal{N}^r_{z^{r-1}}, \prec_r, \theta_r)$
$X^r \leftarrow X^{r-1} \cup W^r$
$z^r \leftarrow z^{r-1} + w^r$
return *generalized temporally repeated flow corresponding to* X^1

The algorithm LEXMAXSI is used to compute a lex-max flow over time with time horizon θ_2 on the set of sources $S_2 \cup \{\psi_2\}$ with respect to $\prec_2$ in the residual network obtained from the static flow in the first iteration. This makes sure that the sources in $S^+ \setminus S_1$ send as much flow as possible until time θ_2 and that the flow out of the sources in $S_2 \cup \{\psi_2\}$ respects the order $\prec_2$. This process is iterated for $i = 1, 2, \ldots, r-1$. In the last iteration the aim is to make sure that the sinks in S_r respect the given order and the time horizon θ_r. In this last iteration we have to do a lex-max flow computation *with* the last iteration. An illustration of the algorithm is given in Figure 5.3. During the execution of Algorithm 19 the dynamic network $\mathcal{N}$ is regarded as a static network in which the cost function on the arcs is given by the transit times. Algorithm 19 returns a generalized temporally repeated flow.

Theorem 5.9 (Correctness of Algorithm 19).

Let $\mathcal{N}$ be a dynamic network with only a single sink t, $S^+ = S_1 \sqcup S_2 \sqcup \ldots \sqcup S_r$, $0 \leq \theta_1 < \theta_2 < \ldots < \theta_r$ rational times, and $\prec$ a total order on the terminals that respects the given partition of S^+. The generalized temporally repeated flow returned by Algorithm 19 with respect to these parameters is a generalized lex-max flow over time. Furthermore, the algorithm has a worst case running time in $\mathcal{O}(|S^+| \cdot \text{MCF}(|V|, |A|))$, which is a strongly polynomial time running time if the minimum-cost flow computations are done with a suitable algorithm.

All preliminaries needed to prove Theorem 5.9 are described in the following paragraph.

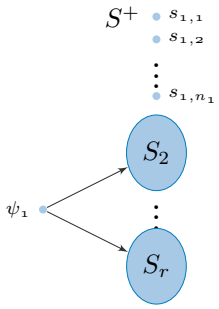
Correctness of Algorithm 19. The **distance** between two nodes v_1 and v_2 in a given dynamic network $\mathcal{N}$ with respect to the transit times τ is denoted by $d(v_1, v_2, \mathcal{N})$. In other words, $d(v_1, v_2, \mathcal{N})$ is the length of a shortest path between v_1 and v_2 in $\mathcal{N}$ with respect to the transit times. Before we can prove the correctness of Algorithm 19, we at first need to recall a few properties of Algorithm 18, all of which can be found in [HT00].

Lemma 5.10 ([HT00], Lemma 4.1). *In any iteration $i \in \{k, k-1, \ldots, 2\}$ of Algorithm 18 the static flow y^i is a minimum-cost circulation in the network $\overline{\mathcal{N}}^i$.*

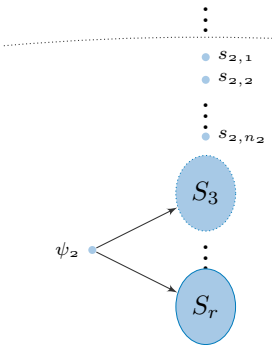
Lemma 5.11 ([HT00], Lemma 4.2). *For any vertex $v \in V$ and any iteration $i \in \{k, k-1, \ldots, 2\}$ of Algorithm 18 it holds that*

$$d(\overline{\mathcal{N}}^{i-1}_{y^{i-1}}, s, v) \geq d(\overline{\mathcal{N}}^{i}_{y^{i}}, s, v).$$

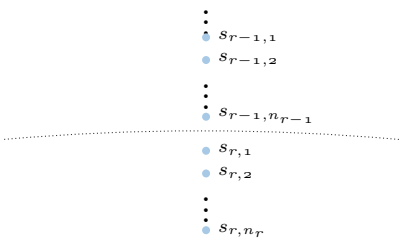
(a) The source set S^+ is partitioned into r subsets



(b) Assume that $S_1 = \{s_{1,1}, \dots, s_{1,n_1}\}$ with $s_{1,n_1} \prec \dots \prec s_{1,1}$. In the first iteration of Algorithm 19 it is ensured that the flow out of $S_1 \cup \{\psi_1\}$ respects $\prec_1$ and that the flow out of S_1 respects the time horizon θ_1. To achieve this, a lex-max flow over time with time horizon θ_1 with respect to $\prec_1$ is computed by LexMaxFlowSI. The skipped last iteration makes sure that the flow out of ψ_1 has infinite time horizon after the first iteration.



(c) In the second iteration of Algorithm 19 the sources in S_1 are disregarded and a lex-max flow over time with time horizon θ_2 with respect to $\prec_2$ is computed by LexMaxFlowSI. The effect is again that the sources in S_2 send flow with respect to $\prec$ and time horizon θ_2.



(d) In the last iteration only the sources in S_r are affected. A lex-max flow over time with respect to $\prec_r$ and θ_r is computed, this time with the lex-max flow over time algorithm *with* the last iteration.

Figure 5.3: An illustration of the generalized lex-mac flow over time algorithm Algorithm 19

Besides needing to show that the flow over time computed by Algorithm 19 fulfills the properties of a generalized lex-max flow over time, it is not even immediately clear that the generalized temporally repeated flow computed by this algorithm results in a feasible flow over time. Showing this fact is the first step towards showing the correctness of Algorithm 19.

Lemma 5.12. *The generalized temporally repeated flow f computed by Algorithm 19 is a feasible flow over time.*

We need to derive some preliminaries before we are able to show this lemma: In each iteration $i \in \{1, 2, \dots, r\}$ the first step of Algorithm 19 is to do a lex-max flow over time computation in the network $\mathcal{N}^i_{z^{i-1}}$ with the help of Algorithm 18. The first step of Algorithm 18 is to attach a super-source s to this network and the additional arc (t, s) with transit time $-\theta_i$. We denote by $\mathcal{N}^i_{z^{i-1}}(s)$ the network $\mathcal{N}^i_{z^{i-1}}$ with the attached super-source s and the additional arc and by $\mathcal{N}^i(s)$ and $\mathcal{N}^i_{z^i}(s)$ the networks $\mathcal{N}^i$ and $\mathcal{N}^i_{z^i}$ with the same super-source and additional arc, respectively. Lemma 5.10 yields that the static flow w^i, needed to compute z^i from z^{i-1} in Line 12 of Algorithm 19, is a minimum-cost circulation in $\mathcal{N}^i_{z^{i-1}}(s)$. Thus, the following lemma holds.

Lemma 5.13. *For each $i \in \{1, 2, \dots, r\}$ the flow z^i, as computed in Line 12 of Algorithm 19, is a feasible minimum-cost circulation in the network $\mathcal{N}^i(s)$.*

Proof. As Lemma 5.10 yields that w^i is a minimum-cost circulation in $\mathcal{N}^i_{z^{i-1}}(s)$ for all $i \in \{1, 2, \dots, r\}$ it follows that $z^i = z^{i-1} + w^i$ is a minimum-cost circulation in $\mathcal{N}^i(s)$. □

Lemma 5.14. *For each node $v \in V$ and each iteration $i \in \{1, 2, \dots, r\}$ of Algorithm 19 we have*

$$d(\mathcal{N}^i_{z^i}(s), s, v) \geq d(\mathcal{N}^{i-1}_{z^{i-1}}(s), s, v).$$

Proof. The first step of each iteration $i \in \{1, 2, \dots, r\}$ of Algorithm 19 is to compute a lex-max flow over time with Algorithm 18 in the network $\mathcal{N}^i_{z^{i-1}}(s)$. In order to obtain $\mathcal{N}^i$ from $\mathcal{N}^{i-1}$, we basically shrink the set of considered sources such that the set of sources of $\mathcal{N}^i$ is a subset of the sources of $\mathcal{N}^{i-1}$. This is done by removing the source ψ_{i-1} and attaching the new source ψ_i. Thus, we essentially construct $\mathcal{N}^i(s)$ from $\mathcal{N}^{i+1}(s)$ by removing arcs from s to the sources in S_{i-1} and by reducing the transit time of the arc (t, s) from $-\theta_{i-1}$ to $-\theta_i$. Each of these constructions does not decrease the length of a shortest path from s to any vertex $v \in V$ in the respective networks. Thus, it holds that

$$d(\mathcal{N}^i_{z^{i-1}}(s), s, v) \geq d(\mathcal{N}^{i-1}_{z^{i-1}}(s), s, v).$$

Using Lemma 5.11 we can thus conclude,

$$d(\mathcal{N}^i_{z^i}(s), s, v) \overset{\text{Lem. 5.11}}{\geq} d(\mathcal{N}^i_{z^{i-1}}(s), s, v) \geq d(\mathcal{N}^{i-1}_{z^{i-1}}(s), s, v).$$

□

In each iteration $i \in \{1, \dots, r\}$ of Algorithm 19 the execution of LexMaxSI$(\mathcal{N}^i_{z^{i+1}}, \prec_i, \theta_i)$ returns W^i and the static flow w^i such that W^i is a generalized path decomposition of W^i in $\mathcal{N}^i$. We will write f_{W^i} for the corresponding generalized temporally repeated flow in $\mathcal{N}^i$.

Lemma 5.15. *For any iteration $i \in \{1, 2, \dots, r\}$ of Algorithm 19 the temporally repeated flow f_{W^i} as computed by this algorithm fulfills*

$$f_{W^i}((v, w), \theta) = 0 \text{ for all } \theta < d(\mathcal{N}^{i-1}_{z^{i-1}}(s), s, v),$$

for all $(v, w) \in A^i$. That is, flow will not arrive at the arc (v, w) before time $d(\mathcal{N}^{i-1}_{z^{i-1}}(s), s, v)$.

Proof. W^i is a path decomposition of the lex-max flow w^i computed in iteration $i \in \{1, 2, \dots, r\}$ of Algorithm 19. Thus, all flow carrying path in W^i are paths in $\mathcal{N}^i_{z^{i-1}}(s)$ (as the lex-max flow over time in iteration i is computed in this dynamic network). However, all these paths are also paths in $\mathcal{N}^{i-1}_{z^{i-1}}(s)$ and thus the length of a path from s to v has length at least $d(\mathcal{N}^{i-1}_{z^{i-1}}, s, v)$, which shows the statement of the lemma. □

Lemma 5.16. *If there is a flow carrying path in W^i for some $i \in \{1, 2, \dots, r\}$ that contains an arc $(v, w) \in A$, then we have*

$$f_{W^i}((v, w), d(\mathcal{N}^i_{z^i}(s), s, v)) > 0,$$

*where f_{W^i} is the generalized temporally repeated in $\mathcal{N}^i$ corresponding to W^i. That is, at time $d(\mathcal{N}^i_{z^i}(s), s, v)$ the arc (v, w) is **covered** by flow.*

Proof. We prove this by contradiction. Assume there is a flow carrying path P in W^i with $(v, w) \in P$ but f_{W^i} does not cover this arc at time $d(\mathcal{N}^i_{z^i}(s), s, v)$. Then,

$$\tau(P_{[s,v]}) > d(\mathcal{N}^i_{z^i}(s), s, v).$$

Hence, $P_{[s,v]}$ induces a residual path in $\mathcal{N}^i_{z^i}$ of length strictly less than $-d(\mathcal{N}^i_{z^i}(s), s, v)$. That however implies that $\mathcal{N}^i_{z^i}(s)$ contains a negative cycle, contradicting Lemma 5.13. □

Putting together all the lemmas we derived above, finally enables us to give a proof of Lemma 5.12.

Proof of Lemma 5.12. For $i \in \{1, 2, \dots, r\}$ we denote by f^i the generalized temporally repeated flow in $\mathcal{N}$ defined by X^i as computed in iteration $i \in \{1, 2, \dots, r\}$ of Algorithm 19.
We prove by induction on i that f^i obeys all capacity constraints and is non-negative, i.e., is a feasible flow over time. Flow conservation follows from the fact that f^i is a generalized temporally repeated flow.
The flow over time f^1 is essentially just a lex-max flow over time in $\mathcal{N}$, thus, f^1 does not violate any capacity constraints by construction and also by construction f^1 is non-negative.
Next, we show that f^{i+1} is a feasible flow over time under the assumption that f^i is feasible for $1 < i \leq r$. By construction, $X^{i+1} = X^i \cup W^i$, where W^i is the path decomposition of the static flow w^i computed in iteration i of Algorithm 19. Thus,

$$f^{i+1} = f^i + f_{W^i}.$$

We will show that for all θ and any arc $(v, w) \in A$ the generalized temporally repeated flow f^{i+1} fulfills $0 \leq f^{i+1}((v, w), \theta) \leq u_{(v,w)}$. For this purpose, fix an arc a arc $(v, w) \in A$. We make a case distinction regarding the considered time θ:

1. $\theta \leq d(\mathcal{N}^i_{z^i}(s), s, v)$:
   By Lemma 5.15 we know that in $f_{W^{i+1}}$ flow does not reach the arc (v, w) before time $d(\mathcal{N}^i_{z^i}(s), s, v)$. Thus, we have

   $$f^{i+1}((v, w), \theta) = f^i((v, w), \theta)$$

   and f^i fulfills $0 \leq f^i((v, w), \theta) \leq u_{(v,w)}$ by assumption.

2. $\theta \geq d(\mathcal{N}^{i+1}_{z^{i+1}}(s), s, v)$:
   In this case Lemma 5.16 implies that flow sent by the generalized temporally repeated flow $f_{W^{i+1}}$ arrives at arc (v, w) at time $d(\mathcal{N}^{i+1}_{z^{i+1}}(s), s, v)$ if there is a flow carrying path in W^{i+1} that contains this arc. Hence,

   $$f_{W^{i+1}}((v, w), \theta) = w^{i+1}((v, w)).$$

   and thus $f^{i+1}((v, w), \theta) = z^i((v, w)) + w^{i+1}((v, w)) = z^{i+1}((v, w))$. Clearly, we have $0 \leq z^{i+1}((v, w)) \leq u_{(v,w)}$, showing feasibility.

3. $d(\mathcal{N}^i_{z^i}(s), s, v) \leq \theta \leq d(\mathcal{N}^{i+1}_{z^{i+1}}(s), s, v)$:
   Using the same argument as above, we see that $f^i((v, w), \theta) = z^i((v, w))$. At time θ a subset of the flow carrying paths in the path decomposition of W^{i+1} may be covering (v, w). Thus, we have

   $$f^{i+1}((v, w), \theta) \leq f^i((v, w), \theta) + u_{(v,w)} - z^i((v, w)) = u_{(v,w)},$$

   because $u_{(v,w)} - z^i((v, w))$ is the residual capacity of the arc (v, w) in $\mathcal{N}^{i+1}_{z^i}$. Thus, the capacity constraint is satisfied by f^{i+1}. We also have

   $$f^{i+1}((v, w), \theta) \geq f^i((v, w), \theta) - z^i((v, w)) \geq 0,$$

   because $z^i((v, w))$ is the residual capacity of the arc (w, v) in $\mathcal{N}^{i+1}_{z^i}$. This shows the non-negativity of f^{i+1}.

With this case distinction we have shown that the flow over time f^{i-1} is a non-negative flow over time that respects the capacities. By induction it follows that the flow over time obtained from Algorithm 19 is a feasible flow over time. □

The final step towards proving the correctness of Algorithm 19 is the following lemma.

Lemma 5.17. *Denote by f the generalized temporally repeated flow defined by X^r returned by Algorithm 19. Then f is a generalized lex-max flow over time with respect to $S_1, S_2, \dots, S_r$, $\theta_1, \theta_2, \dots, \theta_r$ and $\prec$.*

Proof. For $i \in \{1, 2, \dots, r\}$ we denote by f^i the generalized temporally repeated flow defined by X^i in iteration i of Algorithm 19. At first note that, after iteration i of the algorithm, the amount of flow sent out of the sources in S_i is not changed. In order to show the claim, we need to show two things: At first we need to deduce that in the flow over time f^i the amount of flow that the sources in $S_r \cup \ldots \cup S_i$ send towards the sink until time θ_i is maximal and that the amount of flow sent from the sources in S_i respects the given order $\prec$ and the time horizon θ_i. As after iteration i the amount of flow sent out of the sources in S_i is not changed, showing these facts suffices to show the statement of the lemma.
We proceed by induction on i. For $i = 1$, the flow over time f^1 defined by X^1 essentially is a lex-max flow over time in the dynamic network $\mathcal{N}^1$ with respect to the order $\prec_1$. However, since the last iteration of the algorithm of Hoppe and Tardos is skipped in Algorithm 18, only the flow out of the sources in S^1 respects the time horizon θ_1. So, we have shown that the flow out of S_1 respects the given order and the time horizon θ_1.
Assume that we have shown the statement for $1 \leq i - 1 < r$. By assumption, the flow over time f^{i-1} is a lex-max flow over time on $\{\psi_{i-1}\} \cup S_{i-1}$ with respect to the given order and the flow sent from the sources in S_{i-1} towards the sink t respects the time horizon θ_i. For the static flow z^{i-1} this has the implication that in *this static flow* no flow is sent out of the sources in $S_r \cup \ldots \cup S_{i-1}$. Thus, a path decomposition of z^{i-1} in $\mathcal{N}^i$ would not contain any flow carrying path from these sources towards t. Hence, z^{i-1} can in particular also be considered to be a static minimum-cost flow that only sends flow originating from the sources in $S_r \cup \ldots \cup S_i$. For the lex-max flow computation in iteration i it is now apparent that the resulting static flow z^i can also be considered to be a minimum-cost circulation in $\mathcal{N}^i(s)$ that only sends flow out of the sources in $S_r \cup \ldots \cup S_i$. Thus, by the correspondence of maximum flows over time and certain static minimum-cost flow that we derived in Section 2.5 (see also Fact 2.21), this implies that f^i sends the maximum possible amount of flow from the sources in $S_1 \cup \ldots \cup S_i$ towards the sink t until time θ_i. That the sources in S_i respects the time horizon θ_i and the order $\prec$ follows from the properties of the lex-max flow over time algorithm (see Algorithm 18). That we do the lex-max flow over time computation *with* the last iteration in iteration r makes sure that all the sources in S_r respect the time horizon θ_r. □

Thus, in order to prove Theorem 5.9, it only remains to check the running time of Algorithm 19.

Proof of Theorem 5.9. Algorithm 19 calls Algorithm 18 as a subroutine r times and in iteration i Algorithm 18 needs running time $\mathcal{O}(|S_i| \cdot \mathrm{MCF}(|V|, |A|))$. Thus, all in all Algorithm 19 has a worst case running time in $\mathcal{O}(|S^+| \cdot \mathrm{MCF}(|V|, |A|))$. □

5.3 Computing Earliest Arrival Transshipments in PSPACE

In this section we describe two polynomial space algorithms for solving a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\mathrm{EAT}}$ in a dynamic network $\mathcal{N}$ with only a single sink t. Both algorithms rely on the algorithm for generalized lex-max flows over time that we derived in the previous section. The first algorithm is of a similar flavor as one of our new algorithms for the quickest transshipment problem, which we introduced in Section 4.3.2. We show that a given earliest arrival transshipment problem can be solved by a convex combination of generalized lex-max flows over time that respect the earliest arrival pattern p^* corresponding to $(\mathcal{N}, b)_{\mathrm{EAT}}$. It turns out that a suitable convex combination can essentially be determined during the computation of p^*. We also show that we can compute such a convex combination in $|S^+|$ times the the worst case running time of the submodular function minimization algorithm of Orlin [Orl09].
The second algorithm produces, in contrast to our first approach, an integral solution of a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\mathrm{EAT}}$. The trade-off to obtain the integral solution is that we need to do multiple parametric submodular function minimizations using the parametric search framework of Megiddo. This results in a much less efficient algorithm. The main idea of this algorithm is similar to the algorithm of Hoppe and Tardos for the quickest transshipment problem, in which a quickest transshipment problem is reduced to a lex-max flow over time problem. We show how to reduce $(\mathcal{N}, b)_{\mathrm{EAT}}$ to a generalized lex-max flow over time problem by applying the algorithm of Hoppe and Tardos several times. To obtain the actual earliest arrival transshipment we can then use a slightly modified variant of Algorithm 19. We start by developing the first

variant. At first we need to come up with some results regarding the structure of earliest arrival transshipments.

5.3.1 The Structure of Earliest Arrival Transshipments

In this section we use generalized lex-max flows over time to describe the structure of earliest arrival transshipments. Assume that $(\mathcal{N}, b)_{\text{EAT}}$ is an earliest arrival transshipment problem in a dynamic network $\mathcal{N}$ with only a single sink t. We also define $S_r \sqcup S_{r-1} \sqcup \ldots \sqcup S_1 = S^+$ and $\theta_r > \theta_{r-1} > \ldots > \theta_1 > \theta_0 = 0$ to be the partition of S^+ and the times as returned by the computation of the earliest arrival pattern p^* in Algorithm 16. Recall, that by Theorem 3.16, we have for all $i \in \{0, 1, 2, \ldots, r-1\}$ and for each $\theta \in [\theta_i, \theta_{i+1})$,

$$p^*(\theta) = o^\theta\Big(\bigcup_{j=i+1}^{r} S_j\Big) + b\Big(\bigcup_{j=1}^{i} S_j\Big). \tag{5.3}$$

For all $i \in \{1, \ldots, r\}$ we get

$$b(S_i) = o^{\theta_i}\Big(\bigcup_{j=i}^{r} S_j\Big) - o^{\theta_i}\Big(\bigcup_{j=i+1}^{r} S_j\Big). \tag{5.4}$$

By the constructions in Algorithm 16 it also holds that

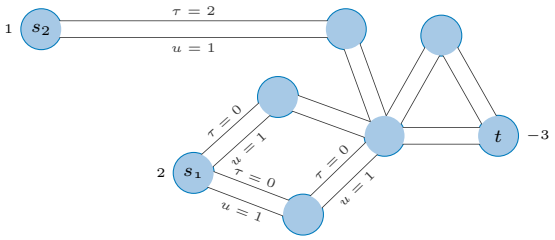
$$o^{\theta_i}\Big(\bigcup_{j=i}^{r} S_j\Big) - o^{\theta_i}\Big(\bigcup_{j=i}^{r} S_j \setminus S\Big) \leq b(S) \text{ for all } S \subseteq \bigcup_{j=i}^{r} S_j, \tag{5.5}$$

for all $i \in \{1, \ldots, r\}$. We start by deriving a first connection between earliest arrival transshipments solving $(\mathcal{N}, b)_{\text{EAT}}$ and generalized lex-max flows over time. One straightforward thing to do is to compute a generalized lex-max flow over time in $\mathcal{N}$ with respect to $S_1, S_2, \ldots, S_r$, $\theta_1, \theta_2, \ldots, \theta_r$ and some total order $\prec$ on S^+ that respects the partition of S^+ corresponding to the earliest arrival pattern of $(\mathcal{N}, b)_{\text{EAT}}$. In general, such a generalized lex-max flow over time as obtained by Algorithm 19 does not respect the pattern p^* corresponding to $(\mathcal{N}, b)_{\text{EAT}}$ if an arbitrary algorithm for the minimum-cost flow computations needed in Algorithm 19 is used (see Figure 5.4). The straightforward idea in order to obtain a generalized lex-max flow over time with pattern p^*, is to do every minimum-cost flow computation in Algorithm 19 by the successive shortest path algorithm. We denote the resulting algorithm by GenLexMaxEA. It turns out that a flow over time obtained this way indeed has pattern p^*. See Figure 5.4c for an illustration.

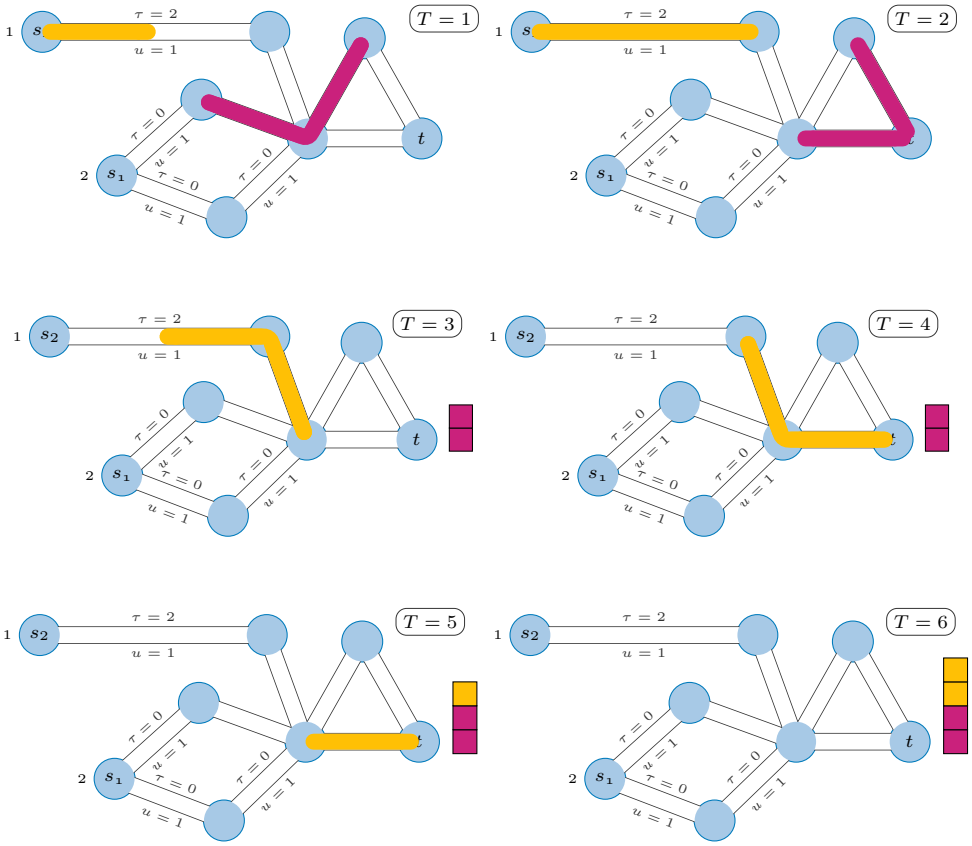
Lemma 5.18. *Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network $\mathcal{N}$ with only a single sink t, and let $S_1 \sqcup \ldots \sqcup S_r = S^+$ and $0 = \theta_0 < \theta_1 < \theta_2 < \ldots < \theta_r$ be the partition of S^+ and the times as corresponding to the earliest arrival pattern of $(\mathcal{N}, b)_{\text{EAT}}$ as returned by Algorithm 16. Further, let $\prec$ be any total order on S^+ that respects the given partition. Denote by f the temporally repeated flow returned by* GenLexMaxEA$(S_r, \ldots, S_1, \theta_r, \ldots, \theta_1, \prec)$. *Then, the flow over time f has pattern p^*, i.e., $|f|_\theta = p^*(\theta)$ for all $\theta \geq 0$. Here, p^* is the earliest arrival pattern corresponding to $(\mathcal{N}, b)_{\text{EAT}}$.*

Proof. Fix some $i \in \{1, 2 \ldots, r\}$. Since the computed flow f is a generalized lex-max flow over time with respect to the given times and sets, it holds that no flow originating from the sources in S_j remains in the networks after time θ_j for $j \in \{1, \ldots, r\}$. Denote by f' the restriction of f to the sources $S_{i+1} \cup \ldots \cup S_r$. We have

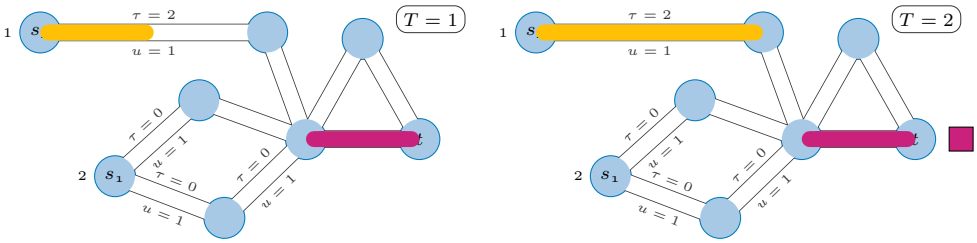
$$|f|_{\theta_i} = \sum_{k=1}^{i} \text{net}_f(S_k, \theta_k) + |f'|_{\theta_i}.$$



(a) An earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$. Each arc with no label has unit transit time and capacity. The pattern computation yields $S_1 = \{s_1\}$, $S_2 = \{s_2\}$, $\theta_1 = 3$, $\theta_2 = 6$ and $s_2 \prec s_1 \prec t$.



(b) An generalized lex-max flow over time corresponding to the parameters given above. This generalized lex-max flow over time does not respect the earliest arrival pattern p^* as no flow has arrived at the sink until time $T = 2$.



(c) When computing the generalized lex-max flow over time with GenEALexMax only how the flow is sent during the first time steps changes such that the resulting flow now respects the pattern p^*

Figure 5.4: An example of a generalized lex-max flow over time

Using Lemma 5.8 and (5.4) we can derive for all $j \in \{1, 2, \dots, i\}$,

$$\operatorname{net}_f(S_j, \theta_j) \overset{\text{Lem. 5.8}}{=} o^{\theta_j}\Big(\bigcup_{k=j}^{r} S_k\Big) - o^{\theta_j}\Big(\bigcup_{k=j+1}^{r} S_k\Big) \overset{(5.4)}{=} b(S_j).$$

Lemma 5.8 also implies that

$$|f'|_{\theta_i} \overset{\text{Lem. 5.8}}{=} o^{\theta_i}\Big(\bigcup_{k=i+1}^{r} S_k\Big).$$

Thus, overall we get using (5.3)

$$|f|_{\theta_i} = o^{\theta_i}\Big(\bigcup_{k=i+1}^{r} S_k\Big) + b\Big(\bigcup_{k=1}^{i} S_k\Big) \overset{(5.3)}{=} p^*(\theta_i).$$

In particular, we have shown that $|f|_{\theta_i} = p^*(\theta_i)$ for all $i \in \{0, 1, \dots, r\}$. Since the successive shortest path algorithm is used to compute all minimum-cost flows in the algorithm GenLexMaxEA, it turns out that for all $\theta \in [\theta_i, \theta_{i+1})$ we have

$$|f'|_{\theta} = o^{\theta}\Big(\bigcup_{k=i+1}^{r} S_k\Big).$$

Together, with Lemma 5.8 and (5.3) this shows the statement of the lemma. □

In the remainder of this section we show the following structural result about earliest arrival transshipments.

Theorem 5.19 (Structure of Earliest Arrival Transshipments).

Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network $\mathcal{N}$ with only a single sink t, and let $S_1, S_2, \dots, S_r$ and $\theta_1 < \theta_2 < \ldots < \theta_r$ be the partition of S^+ and the times as returned by the earliest arrival pattern computation in Algorithm 16.
A flow over time f solving the earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ can be achieved as a convex combination of generalized lex-max flows over time with respect to $S_1, \dots, S_r$ and $\theta_1, \dots, \theta_r$ with pattern p^*, where p^* is the earliest arrival pattern of $(\mathcal{N}, b)_{\text{EAT}}$.
More precisely, there are $d \leq |S^+| - r + 1$ total orders $\prec_1, \dots, \prec_d$ on S^+ respecting $\prec$, and convex coefficients $\lambda_1, \dots, \lambda_d \geq 0$ such that

$$f \coloneqq \lambda_1 f_1 + \lambda_2 f_2 + \ldots + \lambda_d f_d$$

is an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$. Here, f_i is a generalized lex-max flow over time with respect to $\prec_i$, $S_1, \dots, S_r$ and $\theta_1, \dots, \theta_r$ with pattern p^* for all $i \in \{1, \dots, d\}$.

To show that $(\mathcal{N}, b)_{\text{EAT}}$ can in fact be solved by a suitable convex combination of generalized lex-max flows over time, we define the following polyhedron:

$$\begin{aligned}
\mathfrak{P}^{\text{EAT}}_{S^+,t} \coloneqq \{x \in \mathbb{R}^{S^+} \mid\ & \\
& x(S) \leq o^{\theta_1}\Big(\bigcup_{j=2}^{r} S_j \cup S_1 \setminus S\Big) - o^{\theta_1}\Big(\bigcup_{j=1}^{r} S_j\Big) \text{ for all } S \subseteq S_1 \text{ and } x(S_1) = -b(S_1), \\
& x(S) \leq o^{\theta_2}\Big(\bigcup_{j=3}^{r} S_j \cup S_2 \setminus S\Big) - o^{\theta_2}\Big(\bigcup_{j=2}^{r} S_j\Big) \text{ for all } S \subseteq S_2 \text{ and } x(S_2) = -b(S_2), \\
& \dots \\
& x(S) \leq o^{\theta_r}(S_r \setminus S) - o^{\theta_r}(S_r) \text{ for all } S \subseteq S_r \text{ and } x(S_r) = -b(S_r)\}.
\end{aligned} \tag{5.6}$$

For $i \in \{1, 2, \dots, r\}$, we define the set function $g_i \colon 2^{S_i} \to \mathbb{R}$ by

$$g_i(S) \coloneqq o^{\theta_i}\left(\bigcup_{j=i+1}^{r} S_j \cup S_i \setminus S\right) - o^{\theta_i}\left(\bigcup_{j=i}^{r} S_j\right). \tag{5.7}$$

The function g_i is submodular for all $i \in \{1, 2, \dots, r\}$ as o^{θ_i} is submodular (see Theorem 3.6). We also have

$$g_i(S_i) \overset{(5.4)}{=} -b(S_i),$$

for all $i \in \{1, 2, \dots, r\}$. This implies

$$\mathfrak{P}^{\text{EAT}}_{S^+,t} = \bigtimes_{i=1}^{r} \mathcal{B}(g_i),$$

and thus $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ is a polytope. In order to prove Theorem 5.19, we will proceed similarly as for the proof of Theorem 4.11. We show that the vector b lies within $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ and that the vertices of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ correspond to generalized lex-max flows over time with respect to $S_1, S_2, \dots, S_r$ and $\theta_1, \theta_2, \dots, \theta_r$. These results are summarized in the following two lemmas and also illustrated in Figure 5.5.

Lemma 5.20. *Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single sink t and $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ the corresponding polytope as defined in (5.6), then we have $-b \in \mathfrak{P}^{\text{EAT}}_{S^+,t}$.*

Proof. Since $\mathfrak{P}^{\text{EAT}}_{S^+,t} = \bigtimes_{i=1}^{k} \mathcal{B}(g_i)$, it is sufficient to show that for each fixed $i \in \{1, \dots, r\}$ it holds that

$$-b(S) \leq o^{\theta_i}\left(\bigcup_{j=i+1}^{r} S_j \cup S_i \setminus S\right) - o^{\theta_i}\left(\bigcup_{j=i}^{r} S_j\right) \text{ for all } S \subseteq S_i,$$

with equality for S_i. This fact follows with (5.5),

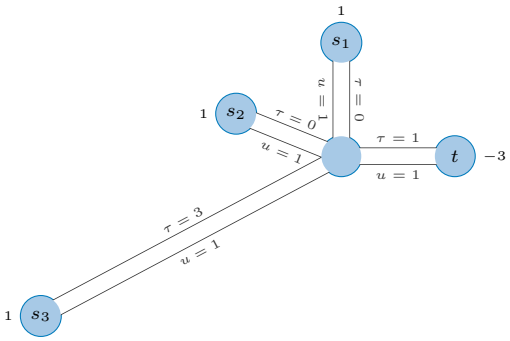
$$o^{\theta_i}\left(\bigcup_{j=i+1}^{r} S_j \cup S_i \setminus S\right) - o^{\theta_i}\left(\bigcup_{j=i}^{r} S_j\right) \overset{(5.5)}{\geq} -b(S), \text{ for all } S \subseteq S_i$$

The equality for S_i follows with (5.4). □

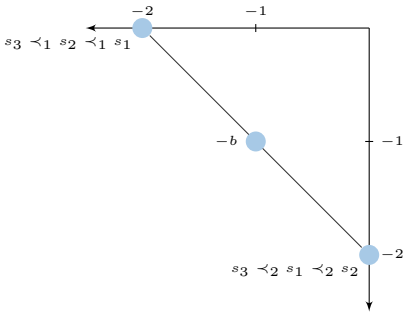
Our next goal is to show that the vertices of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ in fact correspond to generalized lex-max flows over time and vice versa. We know that $\mathfrak{P}^{\text{EAT}}_{S^+,t} = \bigtimes_{i=1}^{r} \mathcal{B}(g_i)$ and also that for each $i \in \{1, \dots, r\}$ the vertices of $\mathcal{B}(g_i)$ are completely characterized by total orders on the ground set of g_i (see Theorem 2.3). Let $\prec$ be a total order on S^+ that respects the given partition. We can split $\prec$ into r total orders $\prec^1, \dots, \prec^r$ on $S_1, \dots, S_r$, respectively, which induce vertices of the corresponding base polytopes. Thus, a total order $\prec$ on S^+ that respects the given partition of S^+ induces a vertex of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ and vice versa. If $\prec$ is a total order on S^+ that respects the given partition, we define the **reversed** order $\overline{\prec}$ that **respects the same partition** by reversing the order $\prec^i$ on each of the subsets S_i for all $i \in \{1, \dots, r\}$. The reversed order of $\prec^i$ is denoted by $\overline{\prec}^i$ for all $i \in \{1, \dots, r\}$. To illustrate this definitions consider again the example shown in Figure 5.5. We have $S^+ = \{s_1, s_2, s_3\}$ with $S_1 = \{s_1, s_2\}$ and $S_2 = \{s_3\}$. The order $\prec$ on S^+ defined by $s_3 \prec s_2 \prec s_1$ clearly respects the given partition and in this example, we have that $\prec^1$ is defined on S_1 by $s_2 \prec^1 s_1$ while $\prec^2$ is the only possible order on S_2. Thus, $\overline{\prec}^1$ is defined by $s_1 \overline{\prec}^1 s_2$ and $\overline{\prec}$ is given by $s_3 \overline{\prec} s_1 \overline{\prec} s_2$.

Lemma 5.21. *Denote by $u^{\prec}$ the vertex of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ corresponding to a total order $\prec$ on S^+ that respects the given partition of S^+. Then, we have*

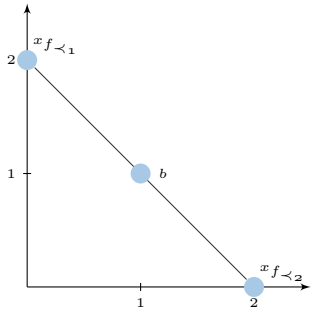
$$-u^{\overline{\prec}} = x_{f_{\prec}},$$



(a) Consider the earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$, depicted in this figure. The pattern computation yields $S_1 = \{s_1, s_2\}$, $S_2 = \{s_3\}$, $\theta_1 = 3$ and $\theta_2 = 5$.



(b) Corresponding to to $(\mathcal{N}, b)_{\text{EAT}}$, we obtain $\mathcal{B}(g_1) = \text{conv}((-2, 0), (0, -2))$ and $\mathcal{B}(g_2) = \{(-1)\}$. Thus, $\mathfrak{P}^{\text{EAT}}_{S^+,t} = \text{conv}((-2, 0, -1), (0, -2, -1))$. A projection can be seen in this figure. The vector $-b = (-1, -1, -1)$ is contained in $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ as expected according to Lemma 5.20. In particular, $-b = 1/2 \cdot (-2, 0, -1) + 1/2 \cdot (0, -2, -1)$. If we assume that the x-coordinate corresponds to s_1, the y-coordinate to s_2 and the z-coordinate to s_3, then the vertex $(-2, 0, -1)$ corresponds to the order $s_3 \prec_1 s_2 \prec_1 s_1$ and the vertex $(0, -2, -1)$ to $s_3 \prec_2 s_1 \prec_2 s_2$.



(c) In this figure we see the convex hull of the characteristic vectors of the generalized lex-max flows over time corresponding to $\prec_1$ and $\prec_2$ with respect to the other given parameters. We see that $-x_{f_{\prec_1}}$ is equal to the vertex of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ corresponding to $\prec_2$ and $-x_{f_{\prec_2}}$ is equal to the vertex of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ corresponding to $\prec_1$ (see also Lemma 5.21). In particular, we have $b = 1/2 x_{f_{\prec_1}} + 1/2 x_{f_{\prec_2}}$ and thus $(\mathcal{N}, b)_{\text{EAT}}$ can be solved by a convex combination of generalized lex-max flows over time which have pattern p^* (See Theorem 5.19).

Figure 5.5: An illustration of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$, Lemma 5.20, Lemma 5.21 and Theorem 5.19.

where $f_\prec$ is a generalized lex-max flow over time with respect to $S_1, \dots, S_r$, $\theta_1, \dots, \theta_r$ and $\prec$. In particular,

$$-\mathfrak{P}^{\text{EAT}}_{S^+,t} = \text{conv}(\{x_{f_\prec} \mid \prec \text{ is a total order on } S^+ \text{ respecting the partition}\}).$$

Proof. Let $\prec$ be a total order on S^+ that respects the given partition. Denote by $\prec^1, \prec^2, \dots, \prec^r$ the induced total orders on $S_1, S_2, \dots, S_r$, respectively. We know that, for each $i \in \{1, \dots, r\}$, the total order $\prec^i$ induces a a vertex $v^{\prec^i}$ of $\mathcal{B}(g^i)$. Since $\mathfrak{P}^{\text{EAT}}_{S^+,t} = \bigtimes_{i=1}^{r} \mathcal{B}(g_i)$, the total order $\prec$ thus induces a vertex $u^\prec$ of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ by

$$u^\prec = (v^{\prec_r}, v^{\prec_{r-1}}, \dots, v^{\prec_1}).$$

On the other hand, each vertex of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ induces a total order $\prec$ on S^+ that respects the given partition. Thus, we obtain

$$\mathfrak{P}^{\text{EAT}}_{S^+,t} = \text{conv}\{u^\prec \mid \prec \text{ is a total order on } S^+ \text{ respecting the partition}\}.$$

Moreover, each total order $\prec$ that respects the given partition also induces a generalized lex-max flow over time with respect to $S_1, \dots, S_r$, $\theta_1, \dots, \theta_r$ and $\prec$ and vice versa.
Denote by $f_\prec$ the generalized lex-max flow over time with respect to $S_1, \dots, S_r$, $\theta_1, \dots, \theta_r$ and $\prec$. Using Lemma 5.8, we get for $s \in S_i$ and for all $i \in \{0, 1, \dots, r-1\}$

$$\begin{aligned} x_{f_\prec}(s) &= \operatorname{net}_f(s, T) \\ &\overset{\text{Lem 5.8}}{=} o^{\theta_i}\Big(\bigcup_{j=i+1}^{r} S_j \cup \{s' \in S_i \mid s' \prec s\} \cup \{s\}\Big) - o^{\theta_i}\Big(\bigcup_{j=i+1}^{r} S_j \cup \{s' \in S_i \mid s' \prec s\}\Big). \end{aligned}$$

For the vertex $u^{\overline{\prec}}$ of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ corresponding to $\overline{\prec}$, we get by Theorem 2.3,

$$\begin{aligned} u^{\overline{\prec}}(s) &= v^{\overline{\prec}^i}(s) \\ &\overset{\text{Thm. 2.3}}{=} o^{\theta_i}\Big(\bigcup_{j=i+1}^{r} S_j \cup S_i \setminus \{s' \in S_i \mid s' \succeq s\}\Big) - o^{\theta_i}\Big(\bigcup_{j=i+1}^{r} S_j \cup S_i \setminus \{s' \in S_i \mid s' \succ s\}\Big) \\ &= o^{\theta_i}\Big(\bigcup_{j=i+1}^{r} S_j \cup \{s' \in S_i \mid s' \prec s\}\Big) - o^{\theta_i}\Big(\bigcup_{j=i+1}^{r} S_j \cup \{s' \in S_i \mid s' \prec s\} \cup \{s\}\Big) \\ &= -x_{f_\prec}(s) \end{aligned}$$

and thus $x_{f_\prec} = -u^{\overline{\prec}}$. Recall, that $v^{\overline{\prec}^i}$ is the vertex of $\mathcal{B}(g_i)$ corresponding to the total order $\overline{\prec}^i$ on S_i. This implies

$$-\mathfrak{P}^{\text{EAT}}_{S^+,t} = \operatorname{conv}(\{x_{f_\prec} \mid \prec \text{ is a total order on } S^+ \text{ respecting the partition}\}),$$

which concludes the proof. □

With the help of these lemmas we can now give the proof of Theorem 5.19.

Proof of Theorem 5.19. Lemma 5.21 implies that

$$\mathfrak{P}^{\text{EAT}}_{S^+,t} \overset{\text{Lem. 5.21}}{=} \operatorname{conv}(\{x_{f_\prec} \mid \prec \text{ is a total order on } S^+ \text{ respecting the partition}\}).$$

Additionally, Lemma 5.20 yields

$$-b \in \mathfrak{P}^{\text{EAT}}_{S^+,t}, \tag{5.8}$$

which means that b can be achieved as convex combination of vertices of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$. That is, there are total orders $\prec_1, \dots, \prec_d$ on S^+ that all respect the given partition and coefficients $\lambda_1, \dots, \lambda_d \geq 0$ with $\sum_{i=1}^{d} \lambda_i = 1$, such that

$$-b = \sum_{i=1}^{d} \lambda_i u^{\prec_i}.$$

Here, $u^{\prec_i}$ is the vertex of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ corresponding to $\prec_i$ for $i = 1, 2, \dots, d$. Using Lemma 5.21 this yields

$$b \overset{\text{Lem. 5.21}}{=} \sum_{i=1}^{d} \lambda_i x_{f_{\overline{\prec}_i}},$$

because of $u^{\overline{\prec}_i} = -x_{f_{\prec_i}}$ for $i = 1, 2, \dots, d$ according to Lemma 5.21. Clearly, $f \coloneqq \sum_{i=1}^{d} \lambda_i f_{\overline{\prec}_i}$ is a feasible flow over time with pattern p^* if the generalized lex-max flows over time have pattern p^*. Thus, a solution of $(\mathcal{N}, b)_{\text{EAT}}$ can be obtained by a convex combination of generalized lex-max flows over time with pattern p^*. It remains to be shown that $d \leq |S^+| - r + 1$. This follows again with Carathéodory's theorem as $\dim(\mathcal{B}(g_i)) \leq |S_i| - 1$. □

5.3.2 PSPACE Computation of Earliest Arrival Transshipments

We have shown that an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ can be solved by a convex combination of generalized lex-max flows over time with respect to the sets $S_1, \dots, S_r$ and times $\theta_1, \dots, \theta_r$ corresponding to the earliest arrival pattern of the given problem as computed by Algorithm 16. We will now show that during the course of Algorithm 16 also a suitable convex combination can be determined. For $i = 1, 2, \dots, r$ we define

$$C_i \coloneqq S^+ \setminus \bigcup_{j=1}^{i-1} S_j,$$

and the function $h_i^\theta \colon 2^{C_i} \to \mathbb{R}$ as follows,

$$h_i^\theta(S) \coloneqq -o^\theta\left(S^+ \setminus \bigcup_{j=1}^{i-1} S_j\right) + o^\theta\left(S^+ \setminus \left(\bigcup_{j=1}^{i-1} S_j \cup S\right)\right) + b(S).$$

Clearly, the function h_i^θ is submodular for all $i \in \{1, \dots, r\}$ and for all $\theta \geq 0$. Recall, that Algorithm 16 determines $S_1, \dots, S_r$ and the times $\theta_1, \dots, \theta_r$ by iteratively doing parametric submodular function minimization of the functions h_i^θ: starting with $i = 1$, Algorithm 16 finds, for each $i = 1, 2, \dots, r$, a maximal $\theta_i \geq 0$ such that $h_i^{\theta_i}(S) \geq 0$ for all $S \subseteq C_i$, and an inclusion-wise maximal subset $S_i \subseteq C_i$ such that $h_i^{\theta_i}(S_i) = 0$. We now explain how to find a convex combination of generalized lex-max flows over time solving a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ while computing its pattern. Again we will take advantage of the additional information that are computed throughout the process of submodular function minimization when an algorithm relying on the framework of Cunningham is used. In this case in each iteration i a vector

$$x_i^* = \operatorname{argmax}\{x^-(C_i) \mid x \in \mathcal{B}(h_i^{\theta_i})\}$$

is determined as a convex combination of vertices of $\mathcal{B}(h_i^{\theta_i})$. More precisely, total orders $\prec_{i,1}, \dots, \prec_{i,d_i}$ on C_i and convex coefficients $\lambda_{i,1}, \dots, \lambda_{i,d_i} \geq 0$ with

$$x_i^* = \lambda_{i,1} u_i^{\prec_{i,1}} + \dots + \lambda_{i,1} u_i^{\prec_{i,d_i}},$$

are computed, where $u_i^{\prec_{i,j}}$ is the vertex of $\mathcal{B}(h_i^{\theta_i})$ corresponding to $\prec_{i,j}$ for $j \in \{1, \dots, d_i\}$. At first, we note that x_i^* restricted to S_i is the zero vector.

Lemma 5.22. *We have $x_i^*(s) = 0$ for all $s \in S_i$ and $i = 1, \dots, r$.*

Proof. The set S_i is a minimizer of $h_i^{\theta_i}$ and by construction we have $h_i^{\theta_i}(S_i) = 0$. The vector x_i^* lies inside the base polytope $\mathcal{B}(h_i^{\theta_i})$ and thus, in particular, has to fulfill $x_i^*(S_i) \leq h_i^{\theta_i}(S_i) = 0$. By Theorem 2.4 we have $(x_i^*)^-(S_i) = 0$ and thus $x_i^*(s) = 0$ for all $s \in S_i$. □

The next important observation is that in each of the orders $\prec_{i,j}$ the set S_i can assumed to be a **lower ideal** for all $j \in \{1, \dots, d_i\}$, i.e., $s \prec_{i,j} s'$ for all $s \in S_i$ and all $s' \in C_i \setminus S_i$.

Lemma 5.23. *Let $i \in \{1, \dots, r\}$ be fixed. For all $j \in \{1, 2, \dots, d_i\}$, the set S_i can assumed to be a lower ideal in each order $\prec_{i,j}$*

Proof. The vertex $u_i^{\prec_{i,j}}$ lies inside the polytope $\mathcal{B}(h_i^{\theta_i})$ and thus in particular fulfills $u_i^{\prec_{i,j}}(S_i) \leq h_i^{\theta_i}(S_i) = 0$ for all $j \in \{1, \dots, d_i\}$. Since $x_i^*(S_i) = 0$, the vertex $u_i^{\prec_{i,j}}$ of $\mathcal{B}(h_i^{\theta_i})$ corresponding to $\prec_{i,j}$ has to fulfill

$$u_i^{\prec_{i,j}}(S_i) = 0 = h_i^{\theta_i}(S_i),$$

for each $j \in \{1, 2, \dots, d_i\}$. The fact that $u_i^{\prec_{i,j}}$ is computed by the Greedy Algorithm with respect to $\prec_{i,j}$ now implies that in each iteration the set S_i can be assumed to be a lower ideal of the

total order $\prec_{i,j}$. If it is not, we can just redefine the order $\prec_{i,j}$ to fulfill this property, without changing the resulting corresponding vertex. To see this, assume that S_i is *not* a lower ideal of $\prec_{i,j}$. For simplicity assume that there is exactly one element $s \in C_i \setminus S_i$ that destroys the lower ideal property. More precisely, let $S_i = S_i^1 \sqcup S_i^2$ such that $s' \prec_{i,j} s$ for all $s' \in S_i^1$ and $s \prec_{i,j} s'$ for all $s' \in S_i^2$. The general case can be proven similarly. We have that $u_i^{\prec_{i,j}}(S_i \cup \{s\}) = h_i^{\theta_i}(S_i \cup \{s\})$ and $u_i^{\prec_{i,j}}(S_i) = 0 = h_i^{\theta_i}(S_i)$. Thus,

$$u_i^{\prec_{i,j}}(s) = h_i^{\theta_i}(S_i \cup \{s\}) - h_i^{\theta_i}(S_i) = h_i^{\theta_i}(S_i \cup \{s\}). \tag{5.9}$$

This already implies that swapping s behind all elements in S_i^2 in the order $\prec_{i,j}$ does not change the corresponding vertex of $\mathcal{B}(h_i^{\theta_i})$ at the component corresponding to s. Clearly, this vertex does also not change at all components corresponding to $s' \in S_i^1$ and at all components corresponding to $s' \in C_i \setminus (S_i \cup \{s\})$. It remains to be shown that this is also the case for all $s' \in S_i^2$. Let $s' \in S_i^2$. Define $\overline{S}_i^2 := \{t \in S_i^2 : t \preceq s'\}$. For the original order $\prec_{i,j}$ we have

$$\begin{aligned} u_i^{\prec_{i,j}}(S_i^1 \cup \overline{S}_i^2) &= u_i^{\prec_{i,j}}(S_i^1 \cup \{s\} \cup \overline{S}_i^2) - u_i^{\prec_{i,j}}(s) \\ &\overset{(5.9)}{=} h_i^{\theta_i}(S_i^1 \cup \{s\} \cup \overline{S}_i^2) - h_i^{\theta_i}(S_i \cup \{s\}). \end{aligned}$$

In order to show that by swapping s behind all elements in S_i^2 in the order $\prec_{i,j}$ the vertex of $\mathcal{B}(h_i^{\theta_i})$ corresponding to this order does not change at the component corresponding to s', it thus suffices to show that

$$h_i^{\theta_i}(S_i^1 \cup \overline{S}_i^2) = h_i^{\theta_i}(S_i^1 \cup \{s\} \cup \overline{S}_i^2) - h_i^{\theta_i}(S_i \cup \{s\}) = u_i^{\prec_{i,j}}(S_i^1 \cup \overline{S}_i^2).$$

Since $u_i^{\prec_{i,j}}$ lies in $\mathcal{B}(h_i^{\theta_i})$, we have $u_i^{\prec_{i,j}}(S_i^1 \cup \overline{S}_i^2) \leq h_i^{\theta_i}(S_i^1 \cup \overline{S}_i^2)$. Using submodularity, we also get

$$h_i^{\theta_i}(S_i^1 \cup \overline{S}_i^2) + h_i^{\theta_i}(S_i^1 \cup S_i^2 \cup \{s\}) \overset{\text{subm.}}{\leq} h_i^{\theta_i}(S_i^1 \cup \overline{S}_i^2 \cup \{s\}) + h_i^{\theta_i}(S_i^1 \cup S_i^2) = h_i^{\theta_i}(S_i^1 \cup \overline{S}_i^2 \cup \{s\}),$$

and thus $u_i^{\prec_{i,j}}(S_i^1 \cup \overline{S}_i^2) \geq h_i^{\theta_i}(S_i^1 \cup \overline{S}_i^2)$. □

Recall, that for all $i \in \{1, \dots, r\}$ we defined the set function g_i with $\mathfrak{P}^{\text{EAT}}_{S^+,t} = \bigtimes_{i=1}^{r} \mathcal{B}(g_i)$ by

$$g_i(S) := o^{\theta_i}\left(\bigcup_{j=i+1}^{r} S_j \cup S_i \setminus S\right) - o^{\theta_i}\left(\bigcup_{j=i}^{r} S_j\right) \text{ for all } S \subseteq S_i.$$

Thus, the submodular function g_i can clearly be obtained by restricting $h_i^{\theta_i}$ to S_i and by translating it by $-b$, i.e.

$$g_i = h_i^{\theta_i}\big|_{S_i} - b\big|_{S_i}.$$

By $\big|_{S_i}$ we indicate the restriction of the functions to S_i, for all $i \in \{1, \dots, r\}$. Lemma 5.23 now implies that the restrictions $\hat{\prec}_{i,1}, \dots, \hat{\prec}_{i,d_i}$ of the orders $\prec_{i,1}, \dots, \prec_{i,d_i}$, respectively, to S_i induce vertices $v_i^{\hat{\prec}_{i,1}}, \dots, v^{\hat{\prec}_{i,d_i}}$ of $\mathcal{B}(g_i)$ such that

$$-b\big|_{S_i} \overset{\text{Lem. 5.23}}{=} \lambda_{i,1} v_i^{\hat{\prec}_{i,1}} + \dots + \lambda_{i,1} v_i^{\hat{\prec}_{i,d_i}}.$$

This argumentation implies the following lemma:

Lemma 5.24. *Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single sink t. While determining the earliest arrival pattern with Algorithm 16 consisting of $S_1, \dots, S_r$ and $\theta_1, \dots, \theta_r$, we can also determine a convex combination of vertices of $\mathcal{B}(g_i)$ yielding $-b\big|_{S_i}$ for each $i \in \{1, \dots, r\}$, provided that an algorithm using the framework of Cunningham is*

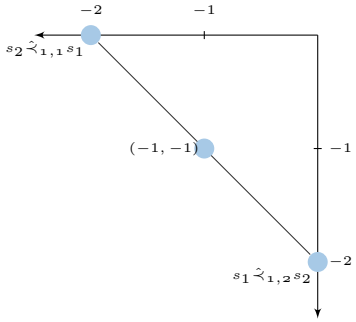
used to do the required submodular function minimizations. More precisely, for all $i \in \{1, \dots, r\}$ *we get total orders* $\prec_{i,1}, \dots, \prec_{i,d_i}$ *on* C_i *and convex coefficients* $\lambda_{i,1}, \dots, \lambda_{i,d_i} \geq 0$ *such that*

$$-b\big|_{S_i} = \lambda_{i,1} v_i^{\hat{\prec}_{i,1}} + \dots + \lambda_{i,1} v_i^{\hat{\prec}_{i,d_i}},$$

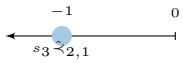
with $d_i \leq |S_i|$ *for all* $i \in \{1, \dots, r\}$. *Here* $\hat{\prec}_{i,j}$ *is the restriction of* $\prec_{i,j}$ *to* S_i *for all* $i \in \{1, \dots, r\}$ *and all* $j \in \{1, \dots, d_i\}$.

Proof. The statement of this theorem follows from the argumentation above. That $d_i \leq |S_i|$ follows again with Carathéodory's theorem and from the fact that in submodular function minimization algorithms relying on the framework of Cunningham it is ensured that a minimal convex combination is computed (for example in $\text{SFM}_{\text{Orlin}}$). □

Regarding the example of an earliest arrival transshipment shown in Figure 5.5, we obtain the vector $(-1, -1)$ as convex combination of vertices of $\mathcal{B}(g_1)$ from the first iteration of the earliest arrival pattern computation, which, in this example, is just $\text{conv}((-2, 0), (0, -2))$. More precisely, we get $\lambda_{1,1} = 1/2$, $\lambda_{1,2} = (1/2)$ while the order $\hat{\prec}_{1,1}$ on S_1 is given by $s_2 \hat{\prec}_{1,1} s_2$ and $\hat{\prec}_{1,2}$ is given by $s_1 \hat{\prec}_{1,2} s_2$ (see Figure 5.6a). In the second iteration of the earliest arrival pattern computation we then just obtain the vector $(-1) = \mathcal{B}(g_2)$, i.e., $\lambda_{2,1} = 1$ while $\hat{\prec}_{2,1}$ is the only possible order on $S_2 = \{s_3\}$ (see Figure 5.6b). For an illustration, see Figure 5.6. After the computation



(a) The base polytope $\mathcal{B}(g_1)$ corresponding to the example shown in Figure 5.5. This figure also shows $-b\big|_{S_1}$ as convex combination of vertices of this polytope as computed during the first iteration of the earliest arrival pattern computation.



(b) The base polytope $\mathcal{B}(g_2)$ corresponding to the example shown in Figure 5.5. This figure also shows $-b\big|_{S_2} = (-1)$ (as a convex combination of the only vertex of this polytope).

Figure 5.6: An illustration of Lemma 5.24

of the earliest arrival pattern for a specific earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$, we thus obtain in total r convex combinations. We will next describe how we can combine these convex combinations to obtain a convex combination of vertices of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ yielding the vertex $-b$. Fix $i \in \{1, \dots, r\}$. By Lemma 5.24 we have convex coefficients $\lambda_{i,1}, \dots, \lambda_{i,d_i} \geq 0$ and total orders $\hat{\prec}_{i,1}, \dots, \hat{\prec}_{i,d_i}$ on S_i such that the corresponding convex combination of vertices of $\mathcal{B}(g_i)$ yields the vector $-b\big|_{S_i}$, for each $i \in \{1, \dots, r\}$. For $i_1 \in \{1, \dots, d_1\}$, $i_2 \in \{1, \dots, d_2\}, \dots, i_r \in \{1, \dots, d_r\}$ we define

$$\lambda_{i_1,\dots,i_r} := \prod_{j=1}^{r} \lambda_{j,i_j}.$$

The vertex of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ corresponding to the convex coefficient $\lambda_{i_1,\dots,i_r}$ is given by

$$(u_r^{\hat{\prec}_{r,i_r}}, \dots, u_2^{\hat{\prec}_{2,i_2}}, u_1^{\hat{\prec}_{1,i_1}}). \tag{5.10}$$

Then,

$$\prod_{i_1=1}^{d_1}\prod_{i_2=1}^{d_2}\cdots\prod_{i_r=1}^{d_r}\lambda_{i_1,\dots,i_r}\cdot(u_r^{\hat{\prec}_{r,i_r}},\dots,u_2^{\hat{\prec}_{2,i_2}},u_1^{\hat{\prec}_{1,i_1}})=-b,$$

is a convex combination of vertices yielding the vector $-b$. However, this convex combination contains $\prod_{i=1}^{r}|S_i|$ many vertices. In order to achieve only $|S^+|-r+1$ many elements, we can use a reduction procedure that gets a convex combination of vectors and computes a new convex combination of affinely independent vectors that yields the same vector. The procedure Reduce$(v_1,\dots,v_k,\lambda_1,\dots,\lambda_k)$, where $v_1,\dots,v_k$ are are chosen from any affine space of dimension d, returns $I\subseteq\{1,2,\dots,k\}$ and convex coefficients λ'_i for all $i\in I$ such that the vectors $(v_i)_{i\in I}$ are an affinely independent family of vectors and

$$\sum_{i=1}^{k}\lambda_i v_i=\sum_{i\in I}\lambda'_i v_i.$$

The procedure Reduce relies on Gaussian Elimination and can be implement in running time $\mathcal{O}(d^2\cdot k)$ (see [Nag07]). In order to construct a suitable convex combination of vertices of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ in a reasonable running time, we use this reduction procedure in every iteration of our combination algorithm described in Algorithm 20. Regarding our example shown in Figure 5.5, the combination of the

Algorithm 20: Algorithm for combining convex combinations, Combine

```
Input  : Sets S_1,...,S_r and times θ_1 < ... < θ_r corresponding to the earliest arrival pattern p* of
         an earliest arrival transshipment problem (N,b)_EAT in a dynamic network N with only a
         single sink t. Convex coefficients λ_{i,1},...,λ_{i,d_i} and total orders ≺_{i,1},... ≺_{i,d_i} on S_i for
         all i ∈ {1,...,r}
Output : A convex combination of affinely independent vertices of P^EAT_{S+,t}
Γ ← (λ_{1,1},...,λ_{1,d_1})
u^{≻_{1,j}} ← the vertex of B(g_1) corresponding to ≺_{1,j} for j ∈ {1,...,d_1}
T ← (u_1^{≻_{1,1}},...,u_{d_1}^{≻_{1,d_1}})
for i ∈ {2,...,r} do
    for j ∈ {1,...,|Γ|} do
        for k ∈ {1,...,d_i} do
            μ_{j,k} ← Γ(j)·λ_{i,k}
            (u_{j,k}) ← (T(j), u_i^{≻_{i,k}}),  u^{≻_{i,k}} is the corresponding vertex of B(g_i)
        end
    end
    Γ ← ((μ_{i_1,i_2})_{i_1∈{1,...,|Γ|},i_2∈{1,...,d_i}})
    T ← ((u_{i_1,i_2})_{i_1∈{1,...,|Γ|},i_2∈{1,...,d_i}})
    Γ,T ← Reduce(Γ,T)
end
d ← |Γ|
return Γ, T
```

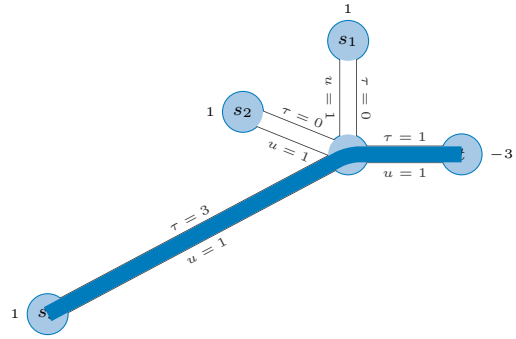
convex combination shown in Figure 5.6 just yields the final convex combination of vertices of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ shown in Figure 5.5b. The solution this convex combination implies for the depicted earliest arrival transshipment problem is illustrated in Figure 5.7.

Putting together everything we explained in this section and by using the fact that every vertex of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ corresponds to a generalized lex-max flow over time in $\mathcal{N}$ with respect to $S_1,\dots,S_r$ and $\theta_1,\dots,\theta_r$, we obtain Algorithm 21 for solving earliest arrival transshipment problems in dynamic networks with only a single sink. In the while loop in Algorithm 21 the earliest arrival pattern is computed. A fast method to do this pattern computation is to use our results from Section 5.1. When using our results from this section together with the algorithm of Nagano [Nag07], we can do every parametric submodular function minimization from the while loop in the same asymptotic running time as required by the algorithm of Orlin.

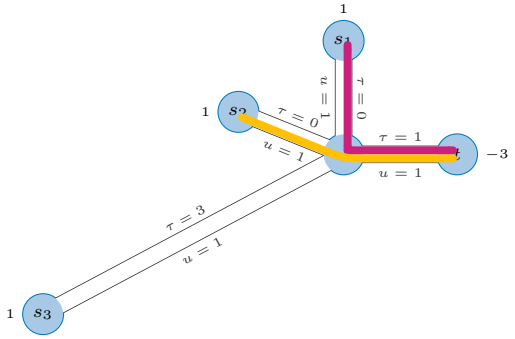
Algorithm 21: Algorithm for solving earliest arrival transshipment problems

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, t)$ and a supply/demand function b
Output : A earliest arrival transshipment f

$S' \leftarrow S^+$
$i \leftarrow 1$
while $S' \neq \varnothing$ **do**
  $\lambda_{i,1}, \dots, \lambda_{i,d_i}, \prec_{i,1}, \dots, \prec_{i,d_i}, \theta_i, S_i \leftarrow$ parametric SFM of h_i^θ on S'
  $\hat{\prec}_{i,j} \leftarrow$ restriction of $\prec_{i,j}$ to S_i for all $j = 1, \dots, d_i$
  $S' \leftarrow S' \setminus S_i$
  $i \leftarrow i + 1$
end
$\lambda_1, \dots, \lambda_d, v_1, \dots, v_d \leftarrow$ COMBINE$((\lambda_{i,1}, \dots, \lambda_{i,d_i} \mid i \in \{1, \dots, r\}), (\hat{\prec}_{i,1}, \dots, \hat{\prec}_{i,d_i} \mid i \in \{1, \dots, r\}))$
$\prec_1, \dots, \prec_d \leftarrow$ orders on S^+ respecting $S_1 \sqcup \dots \sqcup S_r$ cor. to the vertices $v_1, \dots, v_d$ of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$
$\overline{\prec}_1, \dots, \overline{\prec}_d \leftarrow$ reverse orders of $\prec_1, \dots, \prec_d$ respecting the partition of S^+
for $i \in \{1, 2, \dots, d\}$ **do**
  $f_i \leftarrow$ GENEALEXMAX$(\mathcal{N}, S_1, \dots, S_r, \theta_1, \dots, \theta_r, \overline{\prec}_i)$
end
return $\lambda_1 f_1 + \dots + \lambda_d f_d$



(a) In the solution for the earliest arrival transshipment problem shown in Figure 5.5 as computed by Algorithm 21, flow is sent from s_3 into the blue path at rate one for one time unit.



(b) In the solution for the earliest arrival transshipment problem shown in Figure 5.5 as computed by Algorithm 21, flow is sent from s_1 into the yellow path at rate 1/2 for 2 time units and from s_2 into the magenta path at rate 1/2 also for 2 time units.

Figure 5.7: A solution for the earliest arrival transshipment problem from Figure 5.5 as computed by Algorithm 21.

Theorem 5.25 (Correctness of Algorithm 21).

Given an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network $\mathcal{N}$ with only a single sink t Algorithm 21 returns an earliest arrival transshipment solving this problem. Overall, the algorithm does only require polynomial space and computes a generalized temporally repeated solution.

Proof. In the while loop of Algorithm 21 the sets and times $S_1, \dots, S_r$, $\theta_1 \dots, \theta_r$ corresponding to the earliest arrival pattern are computed by doing parametric submodular function minimization of h_i^θ. That this yields the correct sets and times was already argued before. Since we assume that an algorithm using the framework of Cunningham is used to do the required submodular function minimization, also a suitable convex combination of vertices of $\mathcal{B}(g^i)$ is computed that yields the vector $-b\big|_{S_i}$ for all $i \in \{1, \dots, r\}$ by $\lambda_{i,1}, \dots, \lambda_{i,d_i}$ and $\hat{\prec}_{i,1}, \dots, \hat{\prec}_{i,d_i}$ (see Theorem 5.24).
The procedure COMBINE combines all the computed convex combinations to a minimal convex combination of vertices of $\mathfrak{P}^{\text{EAT}}_{S^+,t}$ yielding the vector $-b$.
Lemma 5.21 implies that the generalized lex-max flows over time computed during the algorithm fulfill $-x_{f_i} = v_i$ and thus $\lambda_1 f_1 + \dots + \lambda_d f_d$ has characteristic vector b, which means that this flow satisfies all supplies of the sources. By Lemma 5.18 each of the flows f_i has pattern p^* and thus

the convex combination also has this pattern, implying that it is an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$. □

5.3.3 An Adaptation of the Algorithm of Hoppe and Tardos

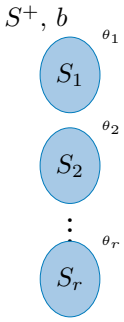
In this section we describe a PSPACE way to compute an integral earliest arrival transshipment solving a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network $\mathcal{N}$ with only a single sink t.
Assume that $S_1, S_2, \dots, S_r$ and $\theta_1, \theta_2, \dots, \theta_r$ are the partition of S^+ and the times as returned when computing the earliest arrival pattern p^* of $(\mathcal{N}, b)_{\text{EAT}}$. The main idea of our algorithm for solving $(\mathcal{N}, b)_{\text{EAT}}$ is to reduce this problem to a generalized lex-max flow over time problem in a modified dynamic network $\mathcal{N}'$. Our reduction will make use of the algorithm of Hoppe and Tardos that we described in Section 3.1.3. We start with a verbal description of our algorithm, the formal algorithm is given in Algorithm 22. An illustration can be found in Figure 5.8.

Description of the Algorithm. We are given $(\mathcal{N}, b)_{\text{EAT}}$, $S_1, S_2, \dots, S_r$ and $\theta_1, \theta_2, \dots, \theta_r$. Our algorithm works in r iterations. In the first iteration, i.e., for $i = 1$, consider the dynamic network $\mathcal{N}^1$ obtained from $\mathcal{N}$ by attaching a super-source ψ_1 to the sources in $S^+ \setminus S_1$ by arcs with zero transit time and infinite capacity. We define $o_1^\theta(X)$ to be the maximal amount of flow that can be sent from the sources in X to the sinks not in X for all $X \subset \{\psi_1\} \cup S_1 \cup \{t\}$ and $\theta \geq 0$. We also modify our supply/demand function b to fit the structure of the dynamic network $\mathcal{N}^1$. The supply of ψ_1 is defined to be $o_1^{\theta_1}(\psi_1)$, i.e., $b^1(\psi_1) = o_1^{\theta_1}(\psi_1)$, and we also set $b^1(s) = b(s)$ for all $s \in S_1$. Looking at the structure of the earliest arrival pattern of $(\mathcal{N}, b)_{\text{EAT}}$ we see that the quickest transshipment problem $(\mathcal{N}^1, b^1)$ is a tight quickest transshipment problem with minimal feasible time horizon θ_1, i.e., $o_1^{\theta_1}(S_1 \cup \{\psi_1\}) = b(S_1 \cup \{\psi_1\})$. We denote the sources of $\mathcal{N}^1$ by S_1^+, i.e. $S_1^+ := S_1 \cup \{\psi_1\}$.
Next, we apply the algorithm of Hoppe and Tardos to the tight transshipment problem $(\mathcal{N}^1, b^1, \theta_1)$. The algorithm of Hoppe and Tardos in two phases creates a modified network from $\mathcal{N}^1$ by attaching new super terminals to the terminals of $\mathcal{N}^1$. More precisely, in the first phase a new super-sink t_0 is attached to t by an arc with zero transit time and infinite capacity, while a super-source s_0 is attached to each $s \in S_1^+$ also by an arc with zero transit time and infinite capacity. Denote this newly created set of sources by $\overline{S}_1^+$. The supplies and demands b^1 are shifted to the newly created terminals. The second phase initializes the chain

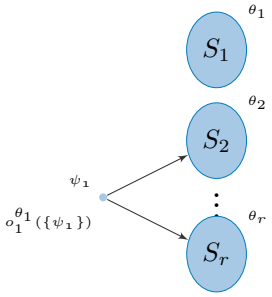
$$\mathcal{C} = \{\varnothing, \{\psi_1\}, \overline{S}_1^+, \overline{S}_1^+ \cup \{t_0\}\}$$

consisting of four tight subsets of terminals of $\mathcal{N}^1$ with respect to $o_1^{\theta_1}$ and b^1.
In each iteration of the second phase of the algorithm of Hoppe and Tardos, which we denote by HoppeTardos, the algorithm adds additional terminals to the network that are always connected to one of the original terminals in $S_1 \cup \{t\}$ by an arc with non-negative capacity and non-negative transit time. In each iteration also the supply/demand function is updated. The algorithm terminates with a chain $\mathcal{C}$ of nested tight subsets such that $|\mathcal{C}| = k + 1$ where k is the number of terminals of the original dynamic network $\mathcal{N}^1$. Since our original problem $(\mathcal{N}^1, b^1)$ was tight and ψ_1 also is a tight source, in the second phase of the algorithm of Hoppe and Tardos new additional super-sources are only attached to the nodes in S_1. In particular, no additional super-sinks are added to the dynamic network and the unique sink t_0 remains at the end of the total order on the terminals implied by $\mathcal{C}$. Denote by $\overline{\mathcal{N}}^1$ the dynamic network resulting from an execution of the algorithm of Hoppe and Tardos on $(\mathcal{N}^1, b^1, \theta_1)$ and denote by $\overline{S}_1$ the set of sources of this dynamic network, while $\overline{b}^1$ are the new supplies and demands.
In the next iteration of our algorithm we proceed working in $\overline{\mathcal{N}}^1$ but we "ignore" the sources in $\overline{S}_1$ in the following iterations. In iteration $i = 2$ we also modify the dynamic network $\overline{\mathcal{N}}^1$ a bit. We remove ψ_1 and instead attach a super-source ψ_2 to the nodes in $S^+ \setminus (S_1 \cup S_2)$. The resulting dynamic network $\mathcal{N}^2$ has a single sink and sources $\{\psi_2 \cup S_2\}$. We define a new supply function b^2 on this source set as in the first iteration, and again apply the algorithm of Hoppe and Tardos to the tight quickest transshipment problem $(\mathcal{N}^2, b^2, \theta_2)$ which gives us a new tight chain of subsets and a new set of sources $\overline{S}_2$ with supplies and demands $\overline{b}^2$. In the subsequent iterations we proceed similarly. After the last iteration for $i = r$, we thus get a completely modified dynamic network $\overline{\mathcal{N}} := \overline{\mathcal{N}}^r$

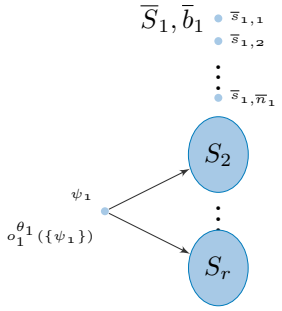
with source sets $\overline{S}_1, \ldots, \overline{S_r}$ and a new supply function $\overline{b}$. Additionally, we also get a total order $\prec$ on the new set of sources $\overline{S}^+ = \overline{S}_1 \sqcup \ldots \sqcup \overline{S}_r$ that respects the partition given by $\overline{S}_1, \ldots, \overline{S_r}$. The order $\prec$ is implied by all the tight chains of subsets created during the r executions of the algorithm of Hoppe and Tardos. It is immediate that a flow over time in $\overline{\mathcal{N}}$ implies a flow over time in $\mathcal{N}$ of the same value and the same time horizon. It remains to be shown that the earliest arrival pattern also does not change.



(a) Initial situation in Algorithm 22: We are given a partition of S^+ by the earliest arrival pattern computation and time horizons $\theta_1 < \ldots < \theta_r$ for the subsets $S_1, \ldots, S_r$ of S^+.



(b) The modified network $\mathcal{N}_1$ at the beginning of the first iteration of Algorithm 22. In particular, the super-sink ψ_i gets supply $o_1^{\theta_1}(\{\psi_i\})$ such that the resulting quickest transshipment problem is tight.



(c) The dynamic network $\overline{\mathcal{N}}_1$ after the first iteration of Algorithm 22. In particular, we get a new set of sources $\overline{S}_1$ with new supplies $\overline{b}_1$ and a tight chain of subsets of the sources of this network. In particular, this complete chain induces a total order $\prec_1$ on $\overline{S}_1$ (here $\overline{s}_{1,\overline{n}_1} \prec_1 \ldots \prec_1 \overline{s}_{1,2} \prec_1 \overline{s}_{1,1}$) with the property that in a lex-max flow over time with time horizon θ_1 in $\overline{\mathcal{N}}_1$ with respect to $\prec_1$ (such that ψ_1 is ordered before all other sources) all supplies $\overline{b}_1$ are satisfied.

Figure 5.8: An illustration of the first iteration of our algorithm for computing integral earliest arrival transshipments in PSPACE (see Algorithm 22)

The integral earliest arrival transshipment that Algorithm 22 computes for the earliest arrival transshipment problem depicted in Figure 5.5, is shown in Figure 5.9.

Theorem 5.26 (Correctness of Algorithm 22).

Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with multiple sources and only a single sink and let $S_1, \ldots, S_r$ and $\theta_1, \ldots, \theta_r$ be the partition of S^+ and the times computed when deriving the earliest arrival pattern corresponding to $(\mathcal{N}, b)_{\text{EAT}}$.
Let f be the flow over time in $\mathcal{N}$ as returned by Algorithm 22 with respect to the parameters given above. The flow over time f is a flow over time solving the earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$.

Algorithm 22: Computing integral solutions to an earliest arrival transshipment problem

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, t)$ and a supply/demand function b
Output : An integral earliest arrival transshipment f

$S_1, \dots, S_r, \theta_1, \dots, \theta_r \leftarrow$ sets and times corresponding to the earliest arrival pattern p^* of $(\mathcal{N}, b)_{\text{EAT}}$
$V_0 \leftarrow V$
$\{\psi_0\} \leftarrow \varnothing$
for $i = 1, 2, \dots, r$ **do**
  $V_i \leftarrow (V_{i-1} \setminus \{\psi_{i-1}\}) \cup \{\psi_i\}$
  Remove all arcs from A that were attached to ψ_{i-1}
  Attach ψ_i to the sources in $S^+ \setminus (S_1 \cup \dots \cup S_i)$ by arcs with zero transit time and infinite capacity
  $b_i(\psi_i) \leftarrow o_i^{\theta_i}(\psi_i)$
  $b_i(s) \leftarrow b(s)$ for all $s \in S_i$
  $\mathcal{N}_i \leftarrow (D = (V_i, A), u, \tau, \{\psi_i\} \cup S_i, t)$
  $\mathcal{C}_i, \overline{\mathcal{N}}_i \leftarrow$ HoppeTardos$((\mathcal{N}_i, b_i))$
  $\overline{S}_i \leftarrow$ sources of $\overline{\mathcal{N}}_i$
  $\prec_i \leftarrow$ total order on $\overline{S}_i$ induced by the chain $\mathcal{C}_i$
end
$\overline{\mathcal{N}} \leftarrow \overline{\mathcal{N}}_r$
$\overline{S}^+ \leftarrow \overline{S}_1 \cup \dots \cup \overline{S}_r$
$\prec \leftarrow$ total order on $\overline{S}^+$ induced by $\prec_1, \dots, \prec_r$ that respects the given partition of $\overline{S}^+$
$\overline{f} \leftarrow$ GenEALexMax$(\overline{\mathcal{N}}, \overline{S}_1, \dots, \overline{S}_r, \theta_1, \dots, \theta_r, \prec)$
$f \leftarrow$ restriction of $\overline{f}$ to a flow over time in $\mathcal{N}$
return f

Proof of Correctness of Algorithm 22 Before we can derive the correctness of the algorithm we need to deduce a few lemmas. The following lemma is immediate by construction.

Lemma 5.27. *Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with multiple sources and only a single sink t and let $S_1, \dots, S_r$ and $\theta_1, \dots, \theta_r$ be the partition of S^+ and the times computed when deriving the earliest arrival pattern corresponding to $(\mathcal{N}, b)_{\text{EAT}}$.*
Denote by $\overline{\mathcal{N}}$ the resulting dynamic network after iteration $i = r$ of Algorithm 22 with sources $\overline{S}_1 \cup \dots \cup \overline{S}_r$ and let $\prec$ be the induced total order on the set of terminals.
Then, the quickest transshipment problem $(\overline{\mathcal{N}}, \overline{b})$ can be solved by a generalized lex-max flow over time $\overline{f}$ with respect to $\overline{S_1}, \dots, \overline{S_r}$, $\theta_1, \dots, \theta_r$ and $\prec$. The flow over time f in $\mathcal{N}$ induced by $\overline{f}$ solves the quickest transshipment problem $(\mathcal{N}, b)$.

Proof. Follows by construction and Lemma 3.12. □

Lemma 5.28. *Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with multiple sources and only a single sink t and let $S_1, \dots, S_r$ and $\theta_1, \dots, \theta_r$ be the partition of S^+ and the times computed when deriving the earliest arrival pattern corresponding to $(\mathcal{N}, b)_{\text{EAT}}$.*
Denote by $\overline{\mathcal{N}}$ the resulting dynamic network after iteration $i = r$ of Algorithm 22 with sources $\overline{S}_1 \cup \dots \cup \overline{S}_r$ and let $\prec$ be the induced total order on the set of terminals. The earliest arrival pattern of $(\overline{\mathcal{N}}, \overline{b})_{\text{EAT}}$ as computed in Algorithm 16 is given by

$$\overline{p}^*(\theta) = \overline{o}^{\theta}\left(\bigcup_{j=i+1}^{r} \overline{S}_j\right) + \overline{b}\left(\bigcup_{j=1}^{i} \overline{S}_r\right) \text{ for all } \theta \in [\theta_i, \theta_{i+1}), \tag{5.11}$$

for all $i \in \{0, \dots, r-1\}$. In particular the earliest arrival pattern of $(\overline{\mathcal{N}}, \overline{b})_{\text{EAT}}$ is the same as the earliest arrival pattern of $(\mathcal{N}, b)_{\text{EAT}}$.

Proof. We at first show that the earliest arrival pattern of the modified problem $(\overline{\mathcal{N}}, \overline{b})_{\text{EAT}}$ is in fact given by (5.11). We prove that when applying Algorithm 16 to the problem $(\overline{\mathcal{N}}, \overline{b})_{\text{EAT}}$, this is

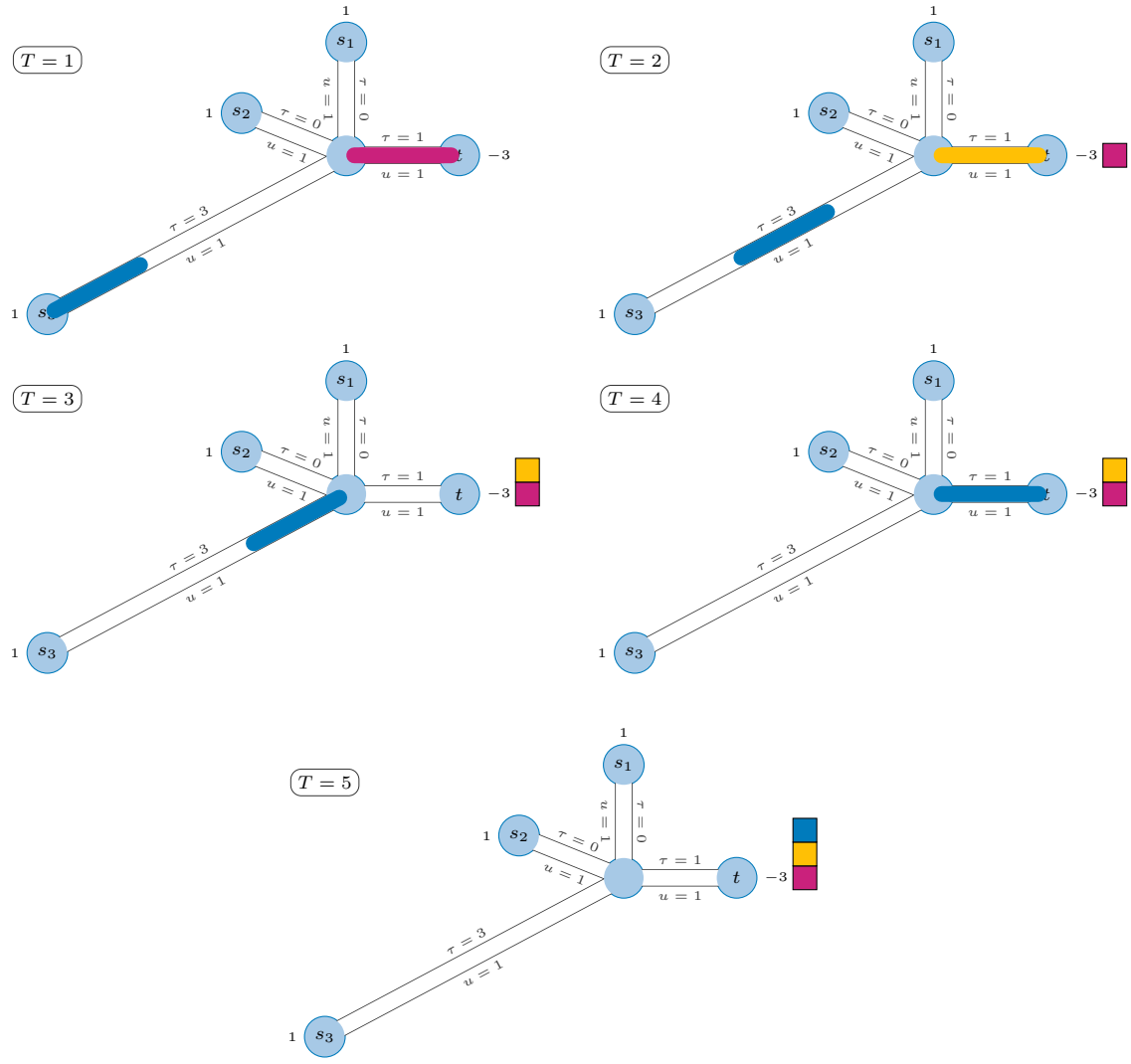


Figure 5.9: The integral earliest arrival transshipment solving the earliest arrival transshipment problem shown in Figure 5.5 as computed by Algorithm 22.

exactly the pattern that is computed. This fact is shown by induction on i. Let $i = 1$. We want to show that

$$\overline{o}^{\theta_1}(\overline{S}^+) - \overline{o}^{\theta_1}(\overline{S}^+ \setminus S) \leq \overline{b}(S) \text{ for all } S \subseteq \overline{S}^+.$$

To prove this, we fix a subset $S \subseteq \overline{S}^+$ and subdivide it as $S = \overline{T}_1 \sqcup \overline{T}_2 \sqcup \ldots \sqcup \overline{T}_r$ with $\overline{T}_i \subseteq \overline{S}_i$, for all $i \in \{1, \ldots, r\}$. By construction using the algorithm of Hoppe and Tardos we know that

$$\overline{o}^{\theta_i}\left(\bigcup_{j=i}^{r} \overline{S}_j\right) - \overline{o}^{\theta_i}\left(\bigcup_{j=i+1}^{r} \overline{S}_j \cup \overline{S}_i \setminus \overline{T}_i\right) \leq \overline{b}(\overline{T}_i), \text{ for all } i \in \{1, \ldots, r\}.$$

Using the strong map property (see Lemma 4.2) this implies

$$\overline{o}^{\theta_1}\left(\bigcup_{j=i}^{r} \overline{S}_j\right) - \overline{o}^{\theta_1}\left(\bigcup_{j=i+1}^{r} \overline{S}_j \cup \overline{S}_i \setminus \overline{T}_i\right) \leq \overline{b}(\overline{T}_i).$$

Thus, we have

$$\sum_{k=1}^{r}\left(\overline{o}^{\theta_1}\left(\bigcup_{j=k}^{r}\overline{S}_j\right)-\overline{o}^{\theta_1}\left(\bigcup_{j=k+1}^{r}\overline{S}_j\cup\overline{S}_k\setminus\overline{T}_k\right)\right)\leq\overline{b}(S).$$

In order to show our claim it suffices to prove

$$\overline{o}^{\theta_1}(\overline{S}^+)-\overline{o}^{\theta_1}(S^+\setminus\overline{S})\leq\sum_{k=1}^{r}\left(\overline{o}^{\theta_1}\left(\bigcup_{j=k}^{r}\overline{S}_j\right)-\overline{o}^{\theta_1}\left(\bigcup_{j=k+1}^{r}\overline{S}_j\cup\overline{S}_k\setminus\overline{T}_k\right)\right)$$

To show this fact, we will prove by induction on l that the following inequality is fulfilled for all $l\in\{1,\dots,r\}$,

$$\sum_{k=1}^{l}\left(\overline{o}^{\theta_1}\left(\bigcup_{j=k}^{r}\overline{S}_j\right)-\overline{o}^{\theta_1}\left(\bigcup_{j=k+1}^{r}\overline{S}_j\cup\overline{S}_k\setminus\overline{T}_k\right)\right)\geq\overline{o}^{\theta_1}(\overline{S}^+)-\overline{o}^{\theta_1}(\overline{S}^+\setminus(\overline{T}_1\cup\ldots\cup\overline{T}_l)).$$

For $l=1$ this statement is trivially true. Assume the statement is true for $1\leq l<r$. We show that it also holds for $l+1$. We achieve this by using submodularity and the induction hypothesis,

$$\begin{aligned}
&\sum_{k=1}^{l+1}\left(\overline{o}^{\theta_1}\left(\bigcup_{j=k}^{r}\overline{S}_j\right)-\overline{o}^{\theta_1}\left(\bigcup_{j=k+1}^{r}\overline{S}_j\cup\overline{S}_k\setminus\overline{T}_k\right)\right)\\
&\overset{\text{I.H.}}{\geq}\ \overline{o}^{\theta_1}(\overline{S}^+)-\overline{o}^{\theta_1}(\overline{S}^+\setminus(\overline{T}_1\cup\ldots\cup\overline{T}_l))+\overline{o}^{\theta_1}\left(\bigcup_{j=l+1}^{r}\overline{S}_j\right)-\overline{o}^{\theta_r}\left(\bigcup_{j=l+2}^{r}\overline{S}_j\cup\overline{S}_{l+1}\setminus\overline{T}_{k+1}\right)\\
&\overset{\text{subm.}}{\geq}\ \overline{o}^{\theta_1}(\overline{S}^+)-\overline{o}^{\theta_1}(\overline{S}^+\setminus(\overline{T}_1\cup\overline{T}_2\cup\ldots\overline{T}_{l+1})).
\end{aligned}$$

Thus, we have shown that

$$\overline{o}^{\theta_1}(\overline{S}^+)-\overline{o}^{\theta_1}(\overline{S}^+\setminus S)\leq\overline{b}(S)\text{ for all }S\subseteq\overline{S}^+.$$

Also, equality holds for $S=\overline{S}_1$ because by construction we have $\overline{o}^{\theta}(\overline{S}^+)=o^{\theta}(S)$ and $\overline{o}^{\theta}(\overline{S}^+\setminus\overline{S}_1)=o^{\theta}(S^+\setminus S_1)$. This is due to the fact that the new sources in the first phase of HoppeTardos are attached by arcs with zero transit time and infinite capacity. This implies that in the first iteration of Algorithm 16 the time that is determined is in fact θ_1 and the set that is determined also needs to be $\overline{S}_1$. We can now prove inductively that in the subsequent iterations also the correct times and sets are determined. Assume the correct times and sets are determined until iteration i. We show that also in iteration $i+1$ the correct times and sets are determined. Thus, we now want to prove that

$$\overline{o}^{\theta_{i+1}}\left(\overline{S}^+\setminus\left(\bigcup_{k=1}^{i}\overline{S}_k\right)\right)-\overline{o}^{\theta_{i+1}}\left(\overline{S}^+\setminus\left(\bigcup_{j=1}^{i}\overline{S}_j\cup S\right)\right)\leq b(S)\text{ for all }S\subseteq\overline{S}^+\setminus\left(\bigcup_{k=1}^{i}\overline{S}_i\right),$$

with equality for $\overline{S}_{i+1}$. However, showing this fact works exactly as the proof of the induction base case from above. It remains to be shown that the computed patten $\overline{p}^*$ is the same as p^*, but this follows directly from the fact that

$$\overline{o}^{\theta}(\overline{S}_r\cup\ldots\overline{S}_i)=o^{\theta}(S_r\cup\ldots\cup S_i)$$

for all θ and $i\in\{1,\dots,r\}$ and $b(S_i)=\overline{b}(\overline{S}_i)$ for all $i\in\{1,\dots,r\}$, because of the new sources attached in the first phase of HoppeTardos by arcs with zero transit time and infinite capacity. □

Putting the lemmas from above together yields a proof of Theorem 5.26.

Proof of Theorem 5.26. The correctness follows immediately from the lemma above. □

5.3.4 Summary, Conclusions and Open Questions

In this section we presented two new algorithms to solve an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a single sink. Both algorithms only work on the original dynamic network $\mathcal{N}$ without requiring any form of expansions of $\mathcal{N}$. With these algorithms we thus present the first algorithms for solving the earliest arrival transshipment problem in polynomial space. This is a huge progress compared to the so far best known algorithm for the earliest arrival transshipment problem by Baumann and Skutella [BS09], which in the worst case requires an exponential expansion of $\mathcal{N}$. Algorithm 21 solves an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ by a convex combination of generalized lex-max flows over time and thus the returned transshipment usually is fractional. Algorithm 22 on the other hand computes and integral solutions. Compared to Algorithm 22, Algorithm 21 has a faster running time because it needs significantly less parametrized submodular function minimizations than the other algorithm. One Open Question is whether it is possible to efficiently compute an integral transshipment out of the transshipment computed by Algorithm 21.

5.4 Multiple Deadline Transshipments Over Time

In Section 3.2.1 we shortly introduced **multiple deadline flows** in dynamic networks with a single source and a single sink (or dynamic networks without any supplies and demands). Recall, that such flows have the property that they are maximal at multiple time horizons but not necessarily at all points in time simultaneously. Thus, such flows are a generalization of earliest arrival flows. For multiple deadline flows an algorithm with strongly polynomial worst case running time for computing them is known.
In this section, we consider the equivalent of multiple deadline flows in dynamic networks with given supplies and demands on the terminals. Given a dynamic network $\mathcal{N}$ with supplies and demands b on the terminals a **multiple deadline transshipment** f is a quickest transshipment solving $(\mathcal{N}, b)$ with the additional property that f is maximal at all points in time $t \in \mathcal{T}$, where $\mathcal{T} = \{\eta_1, \eta_2, \dots, \eta_k\}$ with $\eta_i \in [0, T)$ is a list of points in time given via the input. Here T is the minimal feasible time horizon of $(\mathcal{N}, b)$.

Definition 5.29 (Multiple Deadline Transshipments Over Time).
*Let $(\mathcal{N}, b)$ be a quickest transshipment problem with minimal feasible time horizon T, and $\mathcal{T} = \{\eta_1, \eta_2, \dots, \eta_k\}$ a given list of rational points in time from $[0, T)$. A **multiple deadline transshipment over time** is a quickest transshipment f solving $(\mathcal{N}, b)$ with the additional property that*

$$|f|_{\eta_i} = p^*(\eta_i) \text{ for all } i \in \{1, \dots, k\}.$$

Here p^* is the earliest arrival pattern corresponding to $(\mathcal{N}, b)_{\text{EAT}}$.

Similar to earliest arrival transshipments it is easy to see that in dynamic networks with multiple sinks multiple deadline transshipments over time do not always exist (the network in Figure 6.1 is an example for $\mathcal{T} = \{3, 4\}$). However, for problems in dynamic networks with only a single sink such flows do always exist. This follows immediately from the existence of earliest arrival transshipments in dynamic networks with only a single sink. Nevertheless, so far no polynomial space algorithm for computing such flows was known. With our results from this chapter we can now compute multiple deadline transshipments over time in polynomial space by just using our polynomial space algorithm Algorithm 21 for the earliest arrival transshipment problem. However, using our results from this chapter, we can also come up with an algorithm that computes multiple deadline transshipments over time in strongly polynomial time.

5.4.1 Computing Multiple Deadline Transshipments in Strongly Polynomial Time.

Let $(\mathcal{N}, b)$ be a quickest transshipment problem in a dynamic network with only a single sink and let $\mathcal{T} = \{\eta_1, \eta_2, \dots, \eta_k\}$ be rational times in $[0, T)$ where T is the minimal feasible time horizon of $(\mathcal{N}, b)$. Using Algorithm 16 we can compute the earliest arrival pattern corresponding to $(\mathcal{N}, b)_{\text{EAT}}$. Let $S_1 \sqcup S_2 \sqcup \ldots \sqcup S_r = S^+$ and $\theta_1 < \theta_2 < \ldots < \theta_r = T$ be the sets and times corresponding to the earliest arrival pattern p^* of $(\mathcal{N}, b)_{\text{EAT}}$. By Theorem 5.19 we know that we can solve the earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ by a convex combination of generalized lex-max flows over time with respect to $S_1, \dots, S_r$ and $\theta_1, \dots, \theta_r$ with pattern p^*. Also, a suitable convex combination can be computed in strongly polynomial running time during the pattern computation. That is, in strongly polynomial running time we can compute total orders $\prec_1, \dots, \prec_d$ that respect the given partition and convex coefficients such that the corresponding convex combination of generalized lex-max flows over time yields a solution to $(\mathcal{N}, b)_{\text{EAT}}$ if the generalized lex-max flows over time are computed with GenEALexMax. It is a simple corollary that a suitable multiple deadline transshipment over time can be obtained as a convex combination of the same generalized lex-max flows over time with respect to the given partition, the given times, and $\prec_1, \dots, \prec_d$, but now it suffices for them to satisfy p^* at the times in $\mathcal{T}$.

Corollary 5.30. *Let $(\mathcal{N}, b)$ be a quickest transshipment problem with minimal feasible time horizon T, and let $\mathcal{T} = \{\eta_1, \eta_2, \dots, \eta_k\}$ be rational points in time in $[0, T)$. A multiple deadline transshipment over time f with respect to the given parameters can be obtained as a convex combination of generalized lex-max flows over time such that each of them has value $p^*(\eta_i)$ at time η_i for all $i \in \{1, \dots, k\}$. We call generalized lex-max flows over time with this property* ***multiple deadline generalized lex-max flows over time.***

Proof. This follows directly from the fact that each earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ can be solved by a convex combination of generalized lex-max flows over time with pattern p^* (see Theorem 5.19). □

Thus, our main goal is to show how we can compute such **multiple deadline generalized lex-max flows over time** in strongly polynomial time. It turns out that we can essentially use our generalized lex-max flow over time algorithm (Algorithm 19) for this purpose:
At first we create one list of points in time consisting of the times in $\mathcal{T}$ and $\theta_1, \dots, \theta_r$, say $\gamma_1 < \gamma_2 < \ldots < \gamma_l = T$ (we remove times that appear more than once). Then we define

$$\hat{S}_i = S_i \text{ for all } i \in \{1, \dots, l\} \text{ with } \gamma_i \in (\theta_{i-1}, \theta_i].$$

The sets $\hat{S}_1, \hat{S}_2, \dots, \hat{S}_l$ are clearly not pairwise disjoint, however if we execute the generalized lex-max flow over time algorithm with respect to these sets, the times $\gamma_1 < \gamma_2 < \ldots < \gamma_l$ and any total order $\prec$ on S^+ respecting the initially given partition of S^+, yields a generalized lex-max flow over time with the required properties.

Lemma 5.31. *Let $(\mathcal{N}, b)$ be a quickest transshipment problem with minimal feasible time horizon T, and $\{\eta_1, \dots, \eta_l\}$ rational points in time in $[0, T)$. The multiple deadline generalized lex-max flows over time solving this multiple deadline transshipment problem can be computed with the method described above.*

Proof. That the computed flow over time fulfills the required properties follows completely analogue to the proof of Theorem 5.9. □

Thus, putting all these results together yields that we can compute multiple deadline transshipments over time in strongly polynomial time. Overall, the following theorem follows.

Theorem 5.32.

Multiple deadline transshipment problems can be solved in strongly polynomial time by the method described above.

As a small remark, note that we can also use our adaptation of the algorithm of Hoppe and Tardos to compute multiple deadline transshipment over time in strongly polynomial time. All we have to do is to do the reduction of an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ to a generalized lex-max flow over time problem we described in Section 5.3.3. From this reduction we get $\overline{S}_1, \dots, \overline{S}_r$ in a modified network $\overline{\mathcal{N}}$ together with a total order $\prec$ on the sources that respects the partition. Doing the same construction as above for this generalized lex-max flow over time problem achieves an integral multiple deadline transshipment over time.

5.4.2 An FPTAS without Requiring Time Expansion

As a simple consequence from our algorithm for computing multiple deadline transshipments over time, we obtain an FPTAS for the earliest arrival transshipment problem that does not require any time expansion.

Corollary 5.33. *Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem with minimal feasible time horizon T and $\varepsilon > 0$. Define $k = \log_{1+\varepsilon} T = \log(T)/\log(1+\varepsilon)$ and $\mathcal{T} = \{(1+\varepsilon)^1, \dots, (1+\varepsilon)^k = T\}$. A multiple deadline transshipment over time f with respect to $(\mathcal{N}, b)$ and $\mathcal{T}$ is an $(1+\varepsilon)$-time approximation for $(\mathcal{N}, b)_{\text{EAT}}$ when computed with our algorithm described above.*

Proof. We obtain a flow over time f with the property that

$$|f|_{(1+\varepsilon)^i} = p^*((1+\varepsilon)^i) \text{ for all } i \in \{1, \dots, k\}.$$

Thus for $\theta \in [(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$, we get

$$|f|_\theta \geq |f|_{(1+\varepsilon)^i} = p^*((1+\varepsilon)^i) \geq p^*(\theta/(1+\varepsilon)),$$

because the pattern p^* is monotonically increasing in θ [BS09]. Hence, f is an $(1+\varepsilon)$-time approximation. It remains to check the running time of this procedure. Overall, in order to obtain our approximation, we have to compute at most $|S^+| - r + 1$ generalized lex-max flows over time. Each of these generalized lex-max flows over time is computed with respect to at most $\log_{1+\varepsilon} T + r$ subsets and times, thus we need as many lex-max flow over time computations. Note that we have

$$\log_{1+\varepsilon} T = \frac{\log T}{\log(1+\varepsilon)} \leq \frac{\log T \cdot (1+\varepsilon)}{\varepsilon}.$$

Thus, the running time of our algorithm is a polynomial in the input size and $1/\varepsilon$ and hence the algorithm is an FPTAS. □

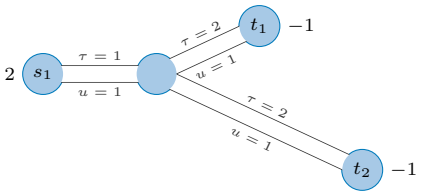
Thus, we obtain an FPTAS for the earliest arrival transshipment problem that does not require any form of time expansion.

6 Earliest Arrival Transshipments in Networks with Multiple Sinks

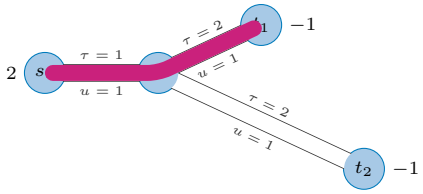
So far, a lot of effort has been put into the development of algorithms for computing *earliest arrival transshipments* in dynamic networks with only a single sink because in such networks earliest arrival transshipments do always exist.
Regarding earliest arrival transshipments in networks with **multiple sinks** not much is known aside from the fact that in such networks earliest arrival transshipments do not exist in general. In particular, there is no algorithm known for computing earliest arrival transshipments in case of existence (even an algorithm that relies on time expansion is not stated in the literature) and also the complexity of deciding whether an earliest arrival transshipment solving a given transshipment problem in a multiple sink network does exist is still unknown. In this chapter we make huge progress on answering these questions.
At first we derive the earliest arrival pattern corresponding to earliest arrival transshipment problems in dynamic networks with multiple sinks and *a single source* and for the special case of tight transshipment problems in general dynamic networks. Making use of the structure of the earliest arrival pattern we formulate the first PSPACE algorithm that decides whether a given earliest arrival transshipment problem has a solution and which computes the solution in case of existence. Again, we achieve this result for earliest arrival transshipment problems in dynamic networks with a single source, and for tight problems in general dynamic networks. At the end of this chapter we settle the complexity by showing that in multiple sink networks it is $\mathcal{NP}$-hard to decide whether an earliest arrival transshipment solving a given problem does exist.

Contents

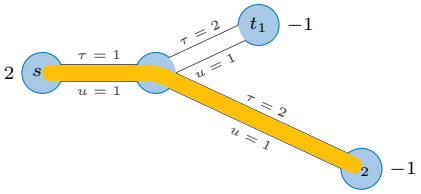
So far, we only considered earliest arrival transshipments in networks with a single sink as in this case earliest arrival transshipment do always exist. This is *not* the case for earliest arrival transshipments in networks with *multiple sinks*. See Figure 6.1 for an example of a small earliest arrival transshipment problem in a network with two sinks that does not have a solution. The depicted example was first published in [BS09]. So far, a lot of research has been put into the development of efficient algorithms for solving earliest arrival transshipment problems in networks with only a single sink. In contrast, surprisingly little is known about earliest arrival transshipments



(a) An earliest arrival transshipment problem in a dynamic network with a single source with supply 2 and two sinks, each with demand -1.



(b) At time 3 at most one flow unit can have arrived at the sinks. This is achieved by sending one flow unit into the magenta path at rate one during the time interval $[0, 1)$.



(c) At time 4 at most two flow unit can have arrived at the sinks. This is achieved by sending one flow unit into the yellow path during the time interval $[0, 1)$ and one flow unit into the magenta path during $[1, 2)$, both at rate one.

Figure 6.1: An example for an earliest arrival transshipment problem in a dynamic network with two sink that does not have a solution: clearly, we cannot send flow such that one flow unit has arrived at the sinks at time 1 and two flow units have arrived at time 2, i.e., we cannot send flow into the yellow *and* magenta path at rate one during$[0, 1)$.

in networks with multiple sinks. To change this, is the main objective of this chapter.
Recall, that the objective of an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with supplies/demands b is to compute a flow over time that sends flow from the sources to the sinks such that it meets all supplies and demands *and* it fulfills the property that *for every point in time simultaneously* as much flow as possible has reached the sinks. That an earliest arrival transshipment has the maximal possible value at each point in time makes it such a good candidate for evacuation planning. Due to the research focused on the efficient exact computation of earliest arrival transshipments in dynamic networks with only a single sink (see [Min73; Wil71; Bau07; SS17b]) it is possible to come up with good evacuation strategies relying on earliest arrival transshipments in this special class of networks. But how do we proceed if we encounter a situation where multiple sinks are inevitable? Of course, earliest arrival transshipments do not exist in general in this setting but it might as well be the case that they do exist in the special situation at hand. However, no efficient algorithm exists that checks whether a given earliest arrival transshipment $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with multiple sinks has a solution and even the complexity of this decision problem is – so far – unknown. Apart from that, also no efficient algorithm is known to solve $(\mathcal{N}, b)_{\text{EAT}}$ even if it is known that the earliest arrival transshipment problem has a solution.
The only progress that has been made in recent years regarding earliest arrival transshipments in networks with multiple sinks is due to Schmidt and Skutella [SS14] who characterize dynamic network with *all zero transit times* such that earliest arrival transshipment problems have a solution for all choices of capacities and supplies and demands. However, their results are non-constructive and do not lead to an efficient algorithm for computing earliest arrival transshipments in case of existence.
During this section we mainly focus on earliest arrival transshipment problems in dynamic network with multiple sinks *but only a single source*. For this setting we derive PSPACE algorithms that check whether a given earliest arrival transshipment problem has a solution and compute the solution in case of existence. Again, our algorithms necessarily have an exponential output size (see [DS15]) but compute it sequentially and thus require only polynomial space. We will also settle the complexity of the decision problem mentioned above. To achieve these results we use similar methods as in Chapters 4 and 5. In particular, similar to our algorithm for solving earliest arrival transshipment problems in networks with a single source, our algorithms developed in this section again strongly use the structure of the earliest arrival pattern, which we derive in the first part of this chapter.

The Earliest Arrival Pattern. Recall, that the **earliest arrival pattern** p^* corresponding to a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ is a function $p^*\colon [0, T) \to \mathbb{R}_{\geq 0}$ such that, for all $\theta \in [0, T)$, $p^*(\theta)$ is the maximal possible value of a flow over time in $\mathcal{N}$ with time horizon θ that does not violate the given supplies and demands. Here T is the minimal feasible time horizon of $(\mathcal{N}, b)_{\text{EAT}}$. For earliest arrival transshipment problems in dynamic networks with only a single sink Baumann and Skutella [Bau07] derived the structure of the corresponding earliest arrival pattern (see also Section 3.2.2). Section 6.1.1 focuses on understanding the structure of the earliest arrival pattern corresponding to earliest arrival transshipment problems in dynamic networks with multiple sinks. We will derive the structure of the earliest arrival pattern for earliest arrival transshipment problems in dynamic networks with a single source (but multiple sinks) and give a strongly polynomial time algorithm for computing the earliest arrival pattern (Section 6.1.2). It turns out that the required constructions needed to compute the required information about the earliest arrival pattern are essentially symmetric to the ones done in the algorithm of Baumann and Skutella. Additionally, we also derive the earliest arrival pattern for *tight* earliest arrival transshipment problems in dynamic network with *multiple sources and sinks.*

Computing Earliest Arrival Transshipments – The Tight Case. In Section 6.2 we concentrate on tight earliest arrival transshipment problems in dynamic networks with multiple sinks. Recall, that an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ is **tight** if we have $o^T(S^+) = b(S^+)$, where T is the minimal feasible time horizon of the given transshipment over time problem. The dynamic network shown in Figure 6.1 is also an example for a tight problem.
For the larger part of this section we only consider tight problems in dynamic networks with only a single source, but in the end we put all our result together to achieve a PSPACE algorithm that checks whether a tight earliest arrival transshipment problem in a *general dynamic network* has a solution and computes it in case of existence. The main ideas behind our algorithm are similar to the ones used in the previous two chapters: We will show that $(\mathcal{N}, b)_{\text{EAT}}$ has a solution if and only if the vector $-b$ lies inside the base polytope $\mathcal{B}(\gamma^T)$ of a suitably chosen submodular function, whose vertices correspond to a family of special flows over time. Our first task, which we concentrate on in Section 6.2.1 and Section 6.2.2, is thus to define a suitable submodular function and a suitable class of flows over time (called **lex-max earliest arrival flows**) and to derive PSPACE algorithms for evaluating the submodular function and for computing the newly defined flows over time.
In Section 6.2.3 we finally show that $(\mathcal{N}, b)_{\text{EAT}}$ has a solution if and only if $-b \in \mathcal{B}(\gamma^T)$, which allows us to check the existence of an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ by doing submodular function minimization. The algorithm for evaluating γ^T, which we develop in Section 6.2.2, thus yields a PSPACE algorithm to check the existence of an earliest arrival transshipment. In order to be able to compute an earliest arrival transshipment in case of existence, we again derive a correspondence between the vertices of $\mathcal{B}(\gamma^T)$ and the special class of flows over time that we defined before. This implies that an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ can be obtained as a convex combination of lex-max earliest arrival flows. A suitable convex combination can be computed during the submodular function minimization required to check the existence of an earliest arrival transshipment – provided an algorithm relying on the framework of Cunningham is used for the submodular function minimization. The lex-max earliest arrival flows occurring in the convex combination can be computed in polynomial space with our algorithm from Section 6.2.2.
Most of our results in Section 6.2 are tailored to tight earliest arrival transshipment problems in dynamic networks with only a single source. At the end of Section 6.2.3 we extend all of these results to tight earliest arrival transshipments in general dynamic networks. In particular, the earliest arrival transshipments that our algorithms compute have the nice structural property of being generalized temporally repeated.

Computing Earliest Arrival Transshipments – The General Case. In Section 6.3 we focus on solving general (non-tight) earliest arrival transshipment problems. All the results we achieve in this section are valid only for the special case of dynamic networks with a single source (and multiple sinks). The methods we use are similar to what we did in Section 5.3.1. Again, our goal is to obtain earliest arrival transshipments as a convex combination of special flows over time that correspond to the vertices of a suitable polytope. In Section 6.3.1 we generalize lex-max earliest arrival flows and define **generalized** lex-max earliest arrival flows and derive a PSPACE algorithm to compute such flows over time in case of existence. In Section 6.3.2 we define a polytope $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ that is strongly

connected with the structure of the earliest arrival pattern of the given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$. It will turn out to be a necessary condition for the existence of an earliest arrival transshipment that $-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$. However, this condition is **not** sufficient for the existence of an earliest arrival transshipment. This is due to the fact that, even if $-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$, the generalized lex-max earliest arrival flows that are shown to correspond to the vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ might not exist. Our main result is that $(\mathcal{N}, b)_{\text{EAT}}$ has a solution if and only if $-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$ *and* all generalized lex-max earliest arrival flows corresponding to the vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ do exist. We will present a polynomial space algorithm to check these conditions (again relying on multiple submodular function minimizations) and to compute a suitable convex combination of generalized lex-max earliest arrival flows in case of existence. Again, the earliest arrival transshipments computed by our algorithms are generalized temporally repeated.

Complexity. In this section we derive the complexity of the decision problem that asks whether a given earliest arrival problem $(\mathcal{N}, b)_{\text{EAT}}$ has a solution. The main result from this section is the proof of the fact that this problem is $\mathcal{NP}$-hard (see 6.4).

Approximation. In the final section of this chapter we show that for earliest arrival transshipment problems in dynamic networks with only a single source a 2-time approximation does always exist (see Section 6.5). Together with a lower bound of 2 shown in [Gro+12] this gives a tight result.

6.1 The Earliest Arrival Pattern

During this section we derive the earliest arrival pattern corresponding to an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with multiple sinks but only a *a single source*. It turns out that in this special setting the construction of the earliest arrival pattern is essentially symmetric to the construction of p^* in networks with multiple sources but only a single sink due to Baumann and Skutella [BS09].

6.1.1 The Structure of the Earliest Arrival Pattern

At first we will consider dynamic networks $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ with *multiple sources and multiple sinks* in order to achieve a slightly more general result. Let $b\colon S^+ \cup S^- \to \mathbb{Z}$ be a supply/demand function on the terminals of $\mathcal{N}$. Recall, that in a dynamic network without supplies and demands the earliest arrival pattern p^* is given by $o^\theta(S^+)$ for all $\theta \geq 0$ and a flow over time respecting this pattern can be achieved as a generalized temporally repeated flow computed by the successive shortest path algorithm from S^+ to S^- (see Observation 3.15 and Algorithm 8). It will later turn out that for at least some time the earliest arrival pattern for $(\mathcal{N}, b)_{\text{EAT}}$ is also equal to $o^\theta(S^+)$. Intuitively, we have $p^*(\theta) = o^\theta(S^+)$ until the supplies of a set of sources run empty or the demands of a set of sinks are full. Afterwards only the remaining non-empty sources can keep sending flow or the remaining non-empty sinks can keep receiving flow, what they will do with the highest possible rate until the next sources are empty or sinks are full, and so on. We start by showing the following lemma which is central for deriving the structure of the earliest arrival pattern.

Lemma 6.1. *Let $\mathcal{N} = (D = (V, A), u, \tau, S^+, S^-)$ be a dynamic network, b a supply/demand function on $S^+ \cup S^-$, and $\theta, q \geq 0$. We have $p^*(\theta) \geq q$ if and only if*

$$o^\theta(S \cup T) \geq q - b(S^+ \setminus S) + b(T) \text{ for all } S \subseteq S^+ \text{ and all } T \subseteq S^-. \tag{6.1}$$

Proof. For the proof we consider an extended dynamic network $\overline{\mathcal{N}} := (\overline{D} = (\overline{V}, \overline{A}), \overline{u}, \overline{\tau}, \overline{S}^+, \overline{S}^-)$ which is defined as follows (see also Figure 6.2): An additional source $\overline{s}$ and an additional sink $\overline{t}$ are added to the original dynamic network, i.e.,

$$\overline{V} := V \cup \{\overline{s}, \overline{t}\} \quad \overline{S}^+ := S^+ \cup \{\overline{s}\} \quad \overline{S}^- := S^- \cup \{\overline{t}\}.$$

The new source $\overline{s}$ is connected to every sink in S^- by an arc with zero transit time and infinite

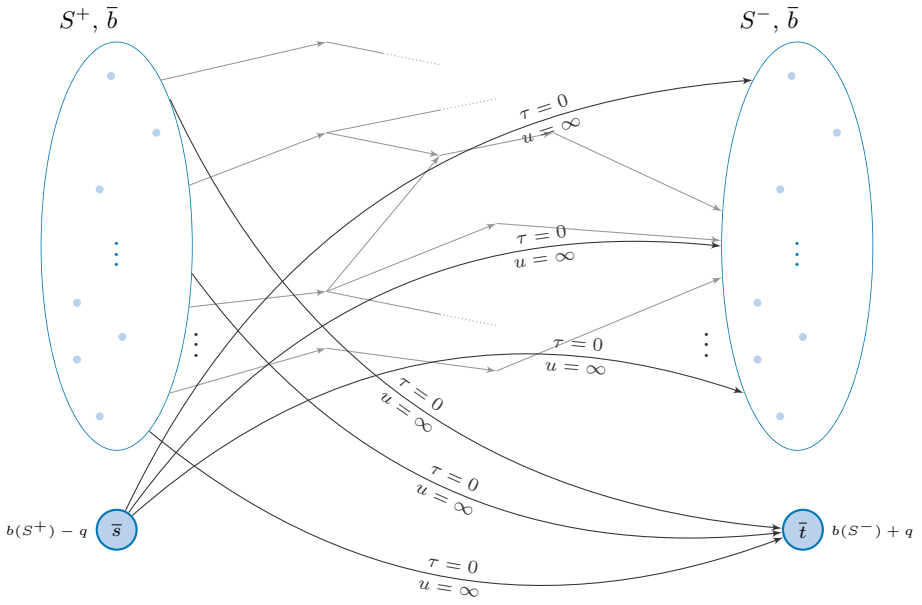


Figure 6.2: The dynamic network $\overline{\mathcal{N}}$

capacity, while $\overline{t}$ is connected to every source in S^+ by an arc with zero transit time and infinite capacity. Overall, we get for $\overline{A}$, $\overline{u}$ and $\overline{\tau}$ in our newly defined dynamic network,

$$\overline{A} \coloneqq A \cup \{(s, \overline{t}) \mid s \in S^+\} \cup \{(\overline{s}, t) \mid t \in S^-\},$$

with

$$\overline{\tau}(a) \coloneqq \begin{cases} \tau(a) \text{ if } a \in A, \\ \infty \text{ otherwise,} \end{cases}$$

and

$$\overline{u}(a) \coloneqq \begin{cases} \tau(a) \text{ if } a \in A, \\ 0 \text{ otherwise,} \end{cases}$$

for all $a \in \overline{A}$. We also extend the supplies and demands to the new set of terminals and define $\overline{b}\colon \overline{S}^+ \cup \overline{S}^- \to \mathbb{Z}$ by

$$\begin{aligned} \overline{b}(\overline{s}) &\coloneqq -b(S^-) - q = b(S^+) - q \\ \overline{b}(\overline{t}) &\coloneqq b(S^-) + q = -b(S^+) + q \\ \overline{b}(v) &\coloneqq b(v) \text{ for all } v \in S^+ \cup S^-. \end{aligned} \tag{6.2}$$

Finally, we define $\overline{o}^\theta$ to be the extension of o^θ to $\overline{\mathcal{N}}$: for each $X \subseteq \overline{S}^+ \cup \overline{S}^-$ and $\theta \geq 0$ we define $\overline{o}^\theta(X)$ to be the maximal amount of flow that can be sent from the sources in $\overline{S}^+ \cap X$ to the sinks in $\overline{S}^- \setminus X$ in the dynamic network $\overline{\mathcal{N}}$ until time θ.
Note that for all $S \subseteq S^+$ and $T \subseteq S^-$ the following relation between o^θ and $\overline{o}^\theta$ holds,

$$o^\theta(S \cup T) = \overline{o}^\theta(S \cup T \cup \overline{t}). \tag{6.3}$$

At first we prove the following statement:

We have $p^*(\theta) \geq q$ if and only if $(\overline{\mathcal{N}}, \overline{b}, \theta)$ is feasible.

To show this fact we assume at first that the transshipment over time problem $(\overline{\mathcal{N}}, \overline{b}, \theta)$ is feasible, i.e., there exists a flow over time $\overline{f}$ with time horizon θ in $\overline{\mathcal{N}}$ that fulfills all the supplies and demands given by $\overline{b}$. In such a flow over time $\overline{f}$ the source $\overline{s}$ (which is only connected to the sinks in S^-) has to send exactly $b(S^+) - q$ units of flow to the sinks in S^-. The remaining q demands of the sinks in S^- need to be fulfilled by the sources in S^+. Thus, in $\overline{f}$ the sources in S^+ send q flow units to the sinks in S^- and $b(S^+) - q$ flow units to the sink $\overline{t}$. We can now restrict $\overline{f}$ to a flow over time f in $\mathcal{N}$ by only considering the flow sent between S^+ and S^- in $\overline{f}$. Thus, we have by construction $|f|_\theta = q$, i.e., $p^*(\theta) \geq q$.
To show the converse direction, assume that $p^*(\theta) \geq q$. This implies that in the original dynamic network $\mathcal{N}$ there exists a flow over time f with time horizon θ and $|f|_\theta = q$ that does not violate any supplies and demands. We can now extend f to a flow over time $\overline{f}$ in $\overline{\mathcal{N}}$ that solves $(\overline{\mathcal{N}}, \overline{b}, \theta)$ by sending the $b(S^+) - q$ supplies that are unfulfilled in f towards the sink $\overline{t}$. The demands that are unfulfilled in f can be strictly fulfilled by sending $b(S^+) - q$ flow units from $\overline{s}$ towards S^-.
By the feasibility criterion of Klinz [Kli] (see Theorem 3.5) we know that the transshipment over time problem $(\overline{\mathcal{N}}, \overline{b}, \theta)$ is feasible if and only if

$$\overline{o}^\theta(X \cup Y) \geq \overline{b}(X \cup Y) \text{ for all } X \subseteq \overline{S}^+ \text{ and all } Y \subseteq \overline{S}^-. \tag{6.4}$$

In light of the observation that we have just shown, in order to prove the statement of the lemma, it suffices to show that (6.1) holds if and only if (6.4) holds.
We start by assuming that (6.4) is valid. Let $S \subseteq S^+$ and $T \subseteq S^-$. we have

$$\begin{aligned} o^\theta(S \cup T) &\overset{(6.3)}{=} \overline{o}^\theta(S \cup T \cup \{\overline{t}\}) \\ &\overset{(6.4)}{\geq} \overline{b}(S \cup T \cup \overline{t}) \\ &= \overline{b}(S) + \overline{b}(T) + \overline{b}(\overline{t}) \\ &\overset{(6.2)}{=} b(S) + b(T) - b(S^+) + q \\ &= q - b(S^+ \setminus S) + b(T). \end{aligned}$$

Thus (6.1) holds for all $S \subseteq S^+$ and all $T \subseteq S^-$.
For the converse direction assume that (6.1) holds. Let $X \subseteq S^+ \cup \{\overline{s}\}$ and $Y \subseteq S^- \cup \{\overline{t}\}$. We have to consider several cases. At first assume that $X = \varnothing$, then

$$0 = \overline{o}^\theta(\varnothing \cup Y) = \overline{o}^\theta(X \cup Y) \geq \overline{b}(Y) \text{ for all choices of } Y \subseteq S^- \cup \{\overline{t}\}.$$

We will thus assume in the following that $X \neq \varnothing$. Next, we consider the case $Y = \overline{S}^-$. We have

$$0 = \overline{o}^\theta(X \cup S^- \cup \{\overline{t}\}) = \overline{o}^\theta(X \cup Y) \geq \overline{b}(X) + \overline{b}(\overline{S}^-).$$

Thus, we will in the following assume $Y \subsetneq \overline{S}^-$. If $\overline{s} \in X$ or $\overline{t} \notin Y$, we have $\overline{o}^\theta(X \cup Y) = \infty$ and thus (6.4) is clearly fulfilled in these cases. The last case we need to consider is thus, $\overline{s} \notin X$, $\overline{t} \in Y$, $X \neq \varnothing$ and $Y \subsetneq \overline{S}^- \cup \{\overline{t}\}$. In this case we have

$$\begin{aligned} \overline{o}^\theta(X \cup Y) &= \overline{o}^\theta(X \cup Y \setminus \{\overline{t}\} \cup \{\overline{t}\}) \\ &\overset{(6.3)}{=} o^\theta(X \cup Y \setminus \{\overline{t}\}) \\ &\overset{(6.1)}{\geq} q - b(S^+ \setminus X) + b(Y \setminus \{\overline{t}\}) \\ &= q - b(S^+) + b(Y \setminus \{\overline{t}\}) + b(X) \\ &\overset{(6.2)}{=} \overline{b}(\overline{t}) + \overline{b}(Y \setminus \{\overline{t}\}) + \overline{b}(X) \\ &= \overline{b}(X) + \overline{b}(Y), \end{aligned}$$

which finishes the proof. □

The following corollary is a simple consequence of Lemma 6.1.

Corollary 6.2. *Let $\mathcal{N}$ be a dynamic network and b a supply/demand function on the terminals of $\mathcal{N}$. The earliest arrival pattern p^* for $(\mathcal{N}, b)_{\text{EAT}}$ is piecewise linear and given by*

$$p^*(\theta) = \min\{o^\theta(S \cup T) + b(S^+ \setminus S) - b(T) \mid S \subseteq S^+ \text{ and } T \subseteq S^-\}.$$

Proof. Let $S \subseteq S^+$ and $T \subseteq S^-$. By definition $o^\theta(S \cup T)$ is the maximal amount of flow that the sources in S can send towards the sinks in $S^- \setminus T$ until time θ, while $-b(T)$ is the maximal amount of flow that the sinks in T can receive until time θ, and $b(S^+ \setminus S)$ is the maximal amount of flow that the sources in $S^+ \setminus S$ can send until time θ in a flow over time respecting supplies and demands. This implies

$$p^*(\theta) \leq \min\{o^\theta(S \cup T) - b(T) + b(S^+ \setminus S) \mid S \subseteq S^+ \text{ and } T \subseteq S^-\}.$$

It is a consequence of Lemma 6.1 that we also have

$$p^*(\theta) \overset{\text{Lem. 6.1}}{\geq} \min\{o^\theta(S \cup T) - b(T) + b(S^+ \setminus S) \mid S \subseteq S^+ \text{ and } T \subseteq S^-\}.$$

□

The next important ingredient to derive the structure of the earliest arrival pattern p^* is the following lemma which shows that the parametric submodular function o^θ fulfills a property similar to the strong map property.

Lemma 6.3. *Let $\mathcal{N}$ be a dynamic network, $T \subseteq T' \subseteq S^-$, $S' \subseteq S \subseteq S^+$ and $0 \leq \theta' \leq \theta$. Then*

$$o^{\theta'}(S \cup T) - o^{\theta'}(S' \cup T') \leq o^\theta(S \cup T) - o^\theta(S' \cup T').$$

Proof. Let $R \subseteq S^+$ be a fixed subset of sources. We have shown in Lemma 4.4 that we have

$$o^{\theta'}(R \cup T) - o^{\theta'}(R \cup T') \overset{\text{Lem. 4.4}}{\leq} o^\theta(R \cup T) - o^\theta(R \cup T').$$

Symmetrically, Baumann and Skutella [BS09] showed that for a fixed $U \subseteq S^-$ we have

$$o^{\theta'}(S \cup U) - o^{\theta'}(S' \cup U) \overset{\text{Lem. 4.2}}{\leq} o^\theta(S \cup U) - o^\theta(S' \cup U).$$

Thus, for $R = S$ and $U = T'$ we get

$$o^{\theta'}(S \cup T) - o^{\theta'}(S \cup T') \leq o^\theta(S \cup T) - o^\theta(S \cup T')$$

and

$$o^{\theta'}(S \cup T') - o^{\theta'}(S' \cup T') \leq o^\theta(S \cup T') - o^\theta(S' \cup T').$$

Adding up both inequalities yields the statement of the lemma. □

Corollary 6.4. *Let $\mathcal{N}$ be a dynamic network, b a supply/demand function on the terminals of $\mathcal{N}$ and $\theta_1 = \max\{\theta \mid p^*(\theta) = o^\theta(S^+)\}$. Then*

$$p^*(\theta) = o^\theta(S^+) \text{ for all } 0 \leq \theta \leq \theta_1.$$

Proof. Aiming for a contradiction, assume that $p(\theta) < o^\theta(S^+)$ for some $0 \leq \theta < \theta_1$. By Lemma 6.1 there exist $S \subseteq S^+$ and $T \subseteq S^-$ with

$$o^\theta(S \cup T) \overset{\text{Lem. 6.1}}{<} o^\theta(S^+) + b(T) - b(S^+ \setminus S). \tag{6.5}$$

Together with Lemma 6.3 this implies

$$\begin{aligned} o^{\theta_1}(S \cup T) - o^{\theta_1}(S^+) &\overset{\text{Lem. 6.3}}{\leq} o^\theta(S \cup T) - o^\theta(S^+) \\ &\overset{(6.5)}{<} b(T) - b(S^+ \setminus S), \end{aligned}$$

and thus, $p^*(\theta_1) < o^{\theta_1}(S^+)$ according to Lemma 6.1, contradicting the definition of θ_1. □

With Corollary 6.4 we have already deduced the earliest arrival pattern for *tight* earliest arrival transshipment problems $(\mathcal{N}, b)_{\text{EAT}}$, i.e., for problems with $o^T(S^+) = b(S^+)$ where T is the minimal feasible time horizon for $(\mathcal{N}, b)_{\text{EAT}}$.

Corollary 6.5 (Earliest Arrival Pattern for Tight Problems). *Let $\mathcal{N}$ be a dynamic network and b a supply/demand function on the terminals of $\mathcal{N}$. If $(\mathcal{N}, b)_{\text{EAT}}$ is a tight earliest arrival transshipment problem, i.e., $o^T(S^+) = b(S^+)$ for the minimal feasible time horizon T of $(\mathcal{N}, b)_{\text{EAT}}$, then the earliest arrival pattern p^* for $(\mathcal{N}, b)_{\text{EAT}}$ is given by*

$$p^*(\theta) = o^\theta(S^+) \text{ for all } \theta \in [0, T).$$

Proof. The statement of the corollary follows immediately from Corollary 6.4. □

Lemma 6.6. *Let $\mathcal{N}$ be a dynamic network, b a supply/demand function on the terminals of $\mathcal{N}$ and $\theta_1 = \max\{\theta \mid p^*(\theta) = o^\theta(S^+)\}$. Then there exists a subsets of sinks $T_1 \subseteq S^-$ and a subset of sources $T_1 \subseteq S^+$ with $S_1 \cup T_1 \neq S^+$ such that*

$$o^{\theta_1}(S^+) = o^{\theta_1}(S_1 \cup T_1) - b(T_1) + b(S^+ \setminus S_1).$$

Proof. Assume that

$$o^{\theta_1}(S^+) < o^{\theta_1}(S \cup T) - b(T) + b(S^+ \setminus S) \text{ for all } T \subseteq S^- \text{ and } S \subseteq S^+ \text{ with } S_1 \cup T_1 \neq S^+.$$

Because the function $o^\theta(B)$ is continuous in θ for a fixed subset of terminals B, this implies that there exists an $\varepsilon > 0$ such that

$$o^{\theta_1+\varepsilon}(S^+) \leq o^{\theta_1+\varepsilon}(S \cup T) - b(T) + b(S^+ \setminus S) \text{ for all } T \subseteq S^- \text{ and } S \subseteq S^+.$$

By Lemma 6.1 this implies $p^*(\theta_1 + \varepsilon) \geq o^{\theta_1+\varepsilon}(S^+)$. □

Lemma 6.6 has a nice intuitive interpretation. Until time θ_1 the sources in S^+ have sent an overall amount of flow of value $o^{\theta_1}(S^+)$ towards the sinks in S^-. The lemma states that if the sources in $S^+ \setminus S_1$ send as little flow as possible and the sinks in T_1 receive as little flow as possible, then the sources in $S^+ \setminus S_1$ have sent their summed up supplies, while the sinks in T_1 have received their summed up demands. This implies that in an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ the sources in $S^+ \setminus S_1$ have to run empty, whereas the sinks in T_1 have to run full until time θ_1.
Assume now that $\mathcal{N}$ is a dynamic network with only a single source $s \in S^+$, and assume that T_1 is the subset of sinks that has to run full at time θ_1 in an earliest arrival transshipment according to Lemma 6.6. We thus know that after time θ_1 only the sinks in $S^- \setminus T_1$ may receive flow. It will turn out that in order to fulfill the earliest arrival property the sinks in $S^- \setminus T_1$ have to receive flow with the highest possible rate $o^\theta(\{s\} \cup T_1)$ until the next set of sinks runs full, and so on.
This implies that in this case the earliest arrival pattern p^* is composed of several s-t earliest arrival patterns in extended networks with an additional super-sink t that is connected to the subset of sinks that has not yet run full. This result is summarized in the following theorem.

Theorem 6.7 (The earliest arrival pattern).

Let $\mathcal{N}$ be a dynamic network with multiple sinks S^- and only a single source s, and b a supply/demand function on the terminals of $\mathcal{N}$. Further let $\theta_1 = \max\{\theta \mid p^*(\theta) = o^\theta(\{s\})\}$ and $\varnothing \subsetneq T_1 \subsetneq S^-$ such that

$$o^{\theta_1}(\{s\} \cup T_1) = o^{\theta_1}(\{s\}) + b(T_1). \tag{6.6}$$

Denote by p_1^* the earliest arrival pattern of the modified dynamic network with sink set $S^- \setminus T_1$. Then,

$$p^*(\theta) = \begin{cases} o^\theta(\{s\}) & \text{if } \theta < \theta_1 \\ p_1^*(\theta) - b(T_1) & \text{if } \theta \geq \theta_1. \end{cases}$$

Proof. We already showed in Corollary 6.4 that $p^*(\theta) = o^\theta(\{s\})$ if $\theta \leq \theta_1$. It remains to be shown that $p^*(\theta) = p_1^*(\theta) - b(T_1)$ for all $\theta \geq \theta_1$. That $\leq$ holds is immediate. We use Lemma 6.1 to show $\geq$. For an arbitrary $T \subseteq S^-$ and $\theta \geq \theta_1$ we get

$$\begin{aligned} o^\theta(\{s\} \cup T) &\overset{\text{subm.}}{\geq} o^\theta(\{s\} \cup T \cap T_1) + o^\theta(\{s\} \cup T_1 \cup T) - o^\theta(\{s\} \cup T_1) \\ &\overset{\text{Lem. 6.3}}{\geq} o^{\theta_1}(\{s\} \cup T \cap T_1) - o^{\theta_1}(\{s\} \cup T_1) + o^\theta(\{s\} \cup T_1 \cup T) \\ &\overset{(6.6)}{=} o^{\theta_1}(\{s\} \cup T \cap T_1) + o^\theta(\{s\} \cup T_1 \cup T) - (o^{\theta_1}(\{s\}) + b(T_1)) \\ &\overset{\text{Cor. 6.2}}{\geq} o^{\theta_1}(\{s\}) + b(T \cap T_1) + (p_1^*(\theta) + b(T \setminus T_1)) - (o^{\theta_1}(\{s\}) + b(T_1)) \\ &= p_1^*(\theta) + b(T \setminus T_1) + b(T \cap T_1) - b(T_1) \\ &= p_1^*(\theta) + b(T) - b(T_1), \end{aligned}$$

which implies $p^*(\theta) = p_1^*(\theta) - b(T_1)$ for all $\theta \geq \theta_1$ by Lemma 6.1. □

6.1.2 Computing the Earliest Arrival Pattern

By Theorem 6.7 we have reduced the computation of the earliest arrival pattern p^* to the computation of θ_1 and T_1 and the pattern p_1^* of the reduced instance on the sink set $S^- \setminus T_1$. It turns out that θ_1 and T_1 can be determined in strongly polynomial time using parametric submodular function minimization.

Lemma 6.8. *Let $\mathcal{N}$ be a dynamic network with only a single source and b a supply/demand function on the terminals of $\mathcal{N}$. Then the time θ_1 with $\theta_1 = \max\{\theta \mid p^*(\theta) = o^\theta(\{s\})\}$ and a subset $T_1 \subsetneq S^-$ with $o^{\theta_1}(\{s\} \cup T_1) = o^{\theta_1}(\{s\}) + b(T_1)$ can be computed by solving a parametric submodular function minimization problem.*

Proof. We define a parametric submodular function g^θ on S^- by

$$g^\theta(T) = o^\theta(\{s\} \cup T) - b(T) - o^\theta(\{s\}) \text{ for all } T \subseteq S^-.$$

The value θ_1 is defined to be $\theta_1 = \max\{\theta \mid p^*(\theta) = o^\theta(\{s\})\}$. By Lemma 6.4, we have $p^*(\theta) = o^\theta(\{s\})$ for $\theta \leq \theta_1$. Thus, for all $\theta \leq \theta_1$ we have by Lemma 6.1

$$o^\theta(\{s\} \cup T) \geq o^\theta(\{s\}) + b(T) \text{ for all } T \subseteq S^- \Rightarrow g^\theta(T) \geq 0 \text{ for all } T \subseteq S^-.$$

The value θ_1 is thus the maximal value with $g^\theta(T) \geq 0$ for all $T \subseteq S^-$ and T_1 is a minimizer of g^{θ_1}. This is what is computed in Algorithm 23 while computing the earliest arrival pattern. □

Applying Theorem 6.7 recursively leads to Algorithm 23 for the computation of the earliest arrival pattern of an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a single sink.

Algorithm 23: Computation of the earliest arrival pattern in dynamic networks with a single source

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^+, S-)$ with a supply/demand function b

Output : Sets $T_1, T_2, \dots, T_r \subseteq S^-$ such that $\bigsqcup_{i=1}^{r} T_i = S^-$ and times $0 < \theta_1 < \theta_2 < \dots < \theta_r$ corresponding to the earliest arrival pattern of the given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$

$i \leftarrow 0$, $T_i \leftarrow \varnothing$, $\theta_i \leftarrow 0$

while $\bigsqcup_{j=1} T_j \neq S^-$ **do**

  Compute the maximal value $\theta_{i+1} \geq 0$ such that

$$o^{\theta_{i+1}}\left(\{s\} \cup \bigcup_{j=1}^{i} T_i \cup T\right) \geq o^{\theta_{i+1}}\left(\{s\} \cup \bigcup_{j=1}^{i} T_i\right) + b(T) \text{ for all } T \subseteq S^- \setminus \left(\bigcup_{j=1}^{i} T_i\right).$$

  Compute the inclusion-wise maximal set $T_{i+1} \subsetneq S^- \setminus \left(\bigcup_{j=1}^{i} T_i\right)$ with

$$o^{\theta_{i+1}}\left(\{s\} \cup \bigcup_{j=1}^{i+1} T_i\right) = o^{\theta_{i+1}}\left(\{s\} \cup \bigcup_{j=1}^{i} T_i\right) + b(T_{i+1}).$$

  $i \leftarrow i + 1$

end

return $T_1, T_2, \dots, T_r$ *and* $\theta_1, \theta_2, \dots, \theta_r$

Theorem 6.9.

Let $\mathcal{N}$ be a dynamic network with a single source s and b a supply/demand function on the terminals of $\mathcal{N}$. The sets $T_1 \sqcup T_2 \sqcup \dots \sqcup T_r = S^-$ and times $0 < \theta_1 < \theta_2 < \dots < \theta_r$ as returned by Algorithm 23 characterize the pattern p^* correctly, that is θ_r is the minimal feasible time horizon and for all $j \in \{0, \dots, r-1\}$ we have,

$$p^*(\theta) = o^{\theta}\left(\{s\} \cup \bigcup_{i=1}^{j} T_i\right) - b\left(\bigcup_{i=1}^{j} T_i\right) \text{ for all } \theta \in [\theta_j, \theta_{j+1}).$$

Proof. With Theorem 6.7 and Lemma 6.8 the only thing that remains to be shown is to prove that $\theta_i > \theta_{i+1}$ for the returned times for all $i \in \{1, \dots, r-1\}$.
We show this by induction. By definition we have $\theta_1 > 0$ as the sinks in T_1 have non-zero demands and hence their whole demand cannot be fulfilled at time $\theta_0 := 0$. For $1 \geq i < r$ we assume by contradiction that $\theta_{i+1} \leq \theta_i$. This yields

$$\begin{aligned} o^{\theta_i}\left(\{s\} \cup \bigcup_{j=1}^{i+1} T_j\right) &\overset{\text{Lem. 6.3}}{\leq} o^{\theta_i}\left(\{s\} \cup \bigcup_{j=1}^{i} T_i\right) + o^{\theta_{i+1}}\left(\{s\} \cup \bigcup_{j=1}^{i+1} T_j\right) - o^{\theta_{i+1}}\left(\{s\} \cup \bigcup_{j=1}^{i} T_i\right) \\ &= o^{\theta_i}\left(\{s\} \cup \bigcup_{j=1}^{i} T_j\right) + b(T_{i+1}) \\ &= o^{\theta_i}\left(\{s\} \cup \bigcup_{j=1}^{i-1} T_j\right) + b(T_i) + b(T_{i+1}) \end{aligned}$$

contradicting the maximal choice of $T_i \subsetneq S^- \setminus (\bigcup_{j=1}^{i-1} T_i)$ during the Algorithm 23. □

Note that in order to really compute the whole earliest arrival pattern, we would again need to compute the breakpoints of the functions $\theta \mapsto o^{\theta}\left(\{s\} \cup \bigcup_{j=1}^{i} T_i\right)$ for all $\theta \in [\theta_i, \theta_{i+1})$ in each iteration of Algorithm 23. However, we again only need the sets $T_1, \dots, T_r$ and times $\theta_1, \dots, \theta_r$ as returned by the algorithm. Also note that during the execution of Algorithm 23, r parametric submodular

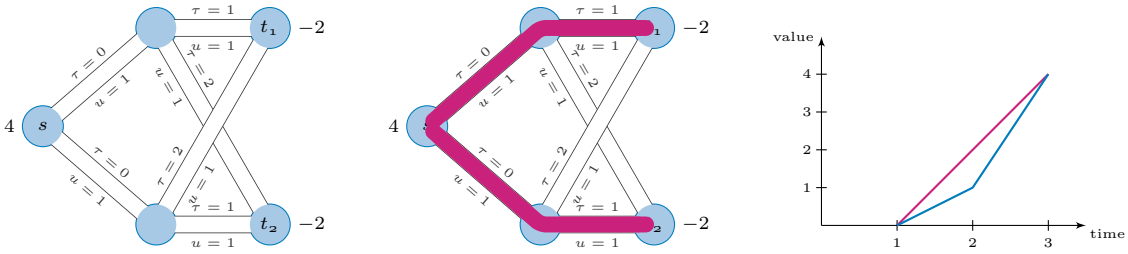
function minimizations are required. Using the same approach as in Section 5.1, we can again exploit the strong map property and get rid of Megiddo's parametric search framework when computing p^*. More precisely, we are able to do each of these parametric submodular function minimizations in the same asymptotic running time as the algorithm of Orlin if we use the algorithm of Nagano [Nag07], or in $|S^-|$ times the running time of the algorithm of Lee et al.

6.1.3 Summary, Conclusions and Open Questions.

In this section we derived the structure of the earliest arrival pattern of an earliest arrival transshipment problem in dynamic networks with a single source and for the special case of a tight earliest arrival transshipment problem in general dynamic networks. Moreover, we presented an efficient algorithm to compute this pattern. The obvious open question here is to derive the structure of the earliest arrival pattern for earliest arrival transshipment problems in general dynamic networks and to give an efficient algorithm to compute this pattern.

6.2 The Tight Case

We have shown in Section 5.3.1 that earliest arrival transshipment problems $(\mathcal{N}, b)_{\text{EAT}}$ in dynamic networks with *only a single sink* can be solved by a convex combination of generalized lex-max flows over time with a suitable arrival pattern. In particular, an earliest arrival transshipment solving a *tight* problem can be obtained as a convex combination of lex-max flows over time (see Theorem 5.19). In networks with multiple sinks but only a single source an analogue statement is *not* true anymore. See Figure 6.3 for an example of an earliest arrival transshipment problem in a dynamic network with only a single source and two sinks that has a solution which cannot be obtained as a convex combination of lex-max flows over time. As our structural main result we will show that – in case of



(a) A tight earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ with minimal feasible time horizon $T = 3$

(b) An earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ sends flow into the indicated paths during $[0, 2)$ at rate one.

(c) The earliest arrival pattern (magenta) and the pattern of the two tight lex-max flows over time in $\mathcal{N}$ with time horizon $T = 3$ (blue). Clearly, a lex-max flow over time can never satisfy the earliest arrival pattern.

Figure 6.3: An example of a tight earliest arrival transshipment problem that cannot be solved by a convex combination of lex-max flows over time since the lex-max flows over time never satisfy the earliest arrival pattern.

existence – earliest arrival transshipments can always be obtained as convex combinations of certain other flows over time, which we call **lex-max earliest arrival flows**. Algorithmic-wise, we present a PSPACE algorithm that checks whether a given tight earliest arrival transshipment problem has a solution and, in case of existence, computes a solution as a convex combination of lex-max earliest arrival flows.

More precisely, we will show that a given *tight* earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a *single source* s has a solution if and only if $-b \in \mathcal{B}(\gamma^T)$, and we present a polynomial space algorithm to check this condition and to compute a solution of $(\mathcal{N}, b)_{\text{EAT}}$ in case of existence. Here γ^T is a suitably defined submodular function and T the minimal feasible time horizon of $(\mathcal{N}, b)_{\text{EAT}}$. The structural result follows from the fact that the vertices of $\mathcal{B}(\gamma^T)$ correspond to lex-max earliest arrival flows and it will turn out that a suitable convex

combination can be computed during the submodular function minimization required to check whether $-b \in \mathcal{B}(\gamma^T)$. In order to achieve these results, several steps are required of which we give a short overview below. In particular, we use the fact that we know by Corollary 6.5 that the earliest arrival pattern of a tight earliest arrival transshipment problem with minimal feasible time horizon T is given by $\theta \mapsto o^\theta(\{s\})$, for $\theta \in [0, T)$. For the main part of this section we only consider tight earliest arrival transshipment problems in dynamic networks with only a single sink, only in the final part will we generalize all our results to general networks. In order to achieve our structural and algorithmic main results, the following steps are required. Note the similarity to the techniques used in the previous chapters:

1. Define the set function γ^θ, show that it is submodular (Section 6.2.1) for every $\theta \geq 0$, and derive a PSPACE algorithm to evaluate this function (Section 6.2.2).
2. Define a special class of flows over time, called **lex-max earliest arrival flows**, and derive a PSPACE algorithm to compute them (Section 6.2.2).
3. Show that a tight problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a single source has a solution if and only if $-b \in \mathcal{B}(\gamma^T)$ where T is the minimal feasible time horizon for $(\mathcal{N}, b)_{\text{EAT}}$. With our PSPACE algorithm for evaluating γ^T we can thus check this condition in PSPACE (Section 6.2.3).
4. Show that the vertices of $\mathcal{B}(\gamma^T)$ correspond to lex-max earliest arrival flows and that a suitable convex combination of lex-max earliest arrival flows can be computed during the submodular function minimization required to check whether $(\mathcal{N}, b)_{\text{EAT}}$ has a solution. With our algorithm for computing lex-max earliest arrival flows we can then compute a flow over time solving $(\mathcal{N}, b)_{\text{EAT}}$ in PSPACE (Section 6.2.3).
5. Extend all our results to tight earliest arrival transshipment problems in general networks (Section 6.3.1).

Until the last part of this section, $\mathcal{N} = (D, u, \tau, s, S^-)$ denotes a dynamic network with multiple sinks S^- but only *a single source* s. We also assume that we are given a supply/demand function b on the terminals of $\mathcal{N}$ and that $(\mathcal{N}, b)_{\text{EAT}}$ is a **tight** earliest arrival transshipment problem. That is, we have $o^T(\{s\}) = b(\{s\}) = -b(S^-)$ where T is the minimal feasible time horizon of $(\mathcal{N}, b)_{\text{EAT}}$. We showed in Corollary 6.4 that in this case the earliest arrival pattern p^* of $(\mathcal{N}, b)_{\text{EAT}}$ is given by

$$p^*(\theta) = o^\theta(\{s\}) \text{ for all } \theta \in [0, T).$$

Since the dynamic networks we consider throughout this section are single source networks, we will in the following assume that b is a vector in $\mathbb{Z}^{S^-}_{<0}$, while the supply of the source s is implicitly given by $-b(S^-)$.
For a given time horizon T we would like to be able to characterize the set of all demand vectors that lead to a tight earliest arrival transshipment problem in $\mathcal{N}$ with time horizon T that has a solution. Thus, we define the set $\mathcal{S}^{\text{EAT}}(\mathcal{N}, T)$ as follows:

$$\mathcal{S}^{\text{EAT}}(\mathcal{N}, T) \coloneqq \{x \in \mathbb{Z}^{S^-}_{\geq 0} \mid (\mathcal{N}, -x)_{\text{EAT}} \text{ is tight with minimal time horizon } T \textit{ and} \text{ has a solution}\}.$$

In order to check whether our tight earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ has a solution, it thus suffices to check whether $-b \in \mathcal{S}^{\text{EAT}}(\mathcal{N}, T)$. Recall, that the minimal feasible time horizon T can be computed in strongly polynomial time using the methods described in Section 4.1. It will be a main result of this section to show that $\mathcal{S}^{\text{EAT}}(\mathcal{N}, T)$ is the base polytope of a submodular function that we can evaluate in polynomial space. However, in order to achieve this result, we at first need to define a suitable submodular function.

6.2.1 Defining a Submodular Function

In this section we concentrate on the first step from our outline presented before: we define a suitably chosen parametric submodular function γ^θ for $\theta \geq 0$ such that we are able to show that $\mathcal{S}^{\text{EAT}}(\mathcal{N}, T) = \mathcal{B}(\gamma^T)$, and a special class of flows over time, called **lex-max earliest arrival flows**,

that are connected to the vertices of $\mathcal{B}(\gamma^T)$. We start by defining the submodular function γ^θ for all $\theta \geq 0$.

Definition 6.10.
Given a dynamic network $\mathcal{N}$ with only a single source s and a time horizon $\theta \geq 0$, we define the set function $\gamma^\theta \colon 2^{S^-} \to \mathbb{R}$ as follows,

$$X \mapsto \begin{array}{l} \text{maximum amount of flow that can arrive at the sinks in } X \text{ until time } \theta \\ \text{in a flow over time with time horizon } \theta \text{ in and } o^{\theta'}(\{s\}) = |f|_{\theta'} \text{ for all } \theta' \leq \theta. \end{array}$$

A flow over time f with time horizon T and pattern $\theta' \mapsto o^{\theta'}(\{s\})$ for all $\theta' \in [0, T)$ that also fulfills

$$-\operatorname{net}_f(X, \theta) = \gamma^\theta(X)$$

for some $\theta \leq T$ and $X \subseteq S^-$ is said to ***satisfy*** *$\gamma^\theta(X)$.*

On the first glance the function γ^T seems to be quite similar to o^T. However, it can easily be seen that both functions are *not* equal. For example, for the dynamic network depicted in Figure 6.3 we obtain $o^3(\{s, t_2\}) = 3$ (recall that $o^3(\{s, t_2\})$ is the maximal amount of flow that can be sent into t_1 until time 3) while $\gamma^3(\{t_1\}) = 2$. What both of these functions have in common is that they are completely independent from potentially given demands on the sinks. Also note that in the definition of γ^θ the pattern $\theta' \mapsto o^{\theta'}(\{s\})$ of tight earliest arrival transshipment problems pops up. Our first main result from this section is to show that the function γ^θ is in fact submodular for each time horizon $\theta \geq 0$.

Theorem 6.11 (Submodularity of γ^θ).
Let $\mathcal{N}$ be a dynamic network with only a single source s. The corresponding set function γ^θ is submodular for every $\theta \geq 0$.

The proof of Theorem 6.11 is quite involved and relies on several lemmas, which we present during the following paragraph.

Preliminaries for the Proof of Theorem 6.11. The proof of Theorem 6.11 strongly relies on a connection between flows over time in $\mathcal{N}$ satisfying $\gamma^\theta(X)$ for some $\theta \geq 0$ and some $X \subseteq S^-$, and certain static lex-max flows in the time-expanded network. Assume for the moment that $\theta \geq 0$ is integral. We consider the time-expanded network $\mathcal{N}^\theta$ corresponding to our given dynamic network $\mathcal{N}$ with only a single source s. For our purpose we have to slightly alter the construction of the time-expanded network that was presented in Section 2.5.2: In each time layer $\theta' \in \{1, \dots, \theta\}$ we attach a super-sink $t_X^{\theta'}$ to nodes $t^{\theta'}$ with $t \in X$ by an arc with infinite capacity. Similarly, we attach a super-sink $t_{S^- \setminus X}^{\theta'}$ to all nodes $t^{\theta'}$ with $t \in S^- \setminus X$. In the end an overall super-sink $t^{\theta'}$ is connected to $t_X^{\theta'}$ and $t_{S^- \setminus X}^{\theta'}$. We denote the resulting network by $\overline{\mathcal{N}}^\theta$. In the following we will consider $\{t_X^i, t_{S^- \setminus X}^i \mid \text{for all } t \in S^-, i \in \{1, \dots, \theta\}\}$ or $\{t^i \mid \text{for all } t \in S^-, i \in \{1, \dots, \theta\}\}$ as sets of sinks of the considered time-expanded network. The single source is given by the overall super-source s^*. This construction (for one time layer) is illustrated in Figure 6.4. Clearly, this construction can easily be extended to arbitrary rational time horizons. Assume now that $\theta = p/q$ for $p, q \in \mathbb{Z}_{>0}$ is a rational time horizon. Recall, that instead of considering time layers for all integral points in time, we can now define time layers for all $\theta \in \mathcal{I} \coloneqq \{1/q, 2/q, \dots, p/q\}$. The rest of the definition of $\overline{\mathcal{N}}^\theta$ carries over one-to-one to this setting.

Lemma 6.12. *A flow over time f in a dynamic network $\mathcal{N}$ with only a single source s with time horizon T satisfying $\gamma^T(X)$ for some $X \subseteq S^-$ and some rational time horizon $T \geq 0$ fulfills the following properties:*

1. *We have $\operatorname{net}_f(X, \theta) = -\gamma^\theta(X)$ for all $\theta \leq T$.*

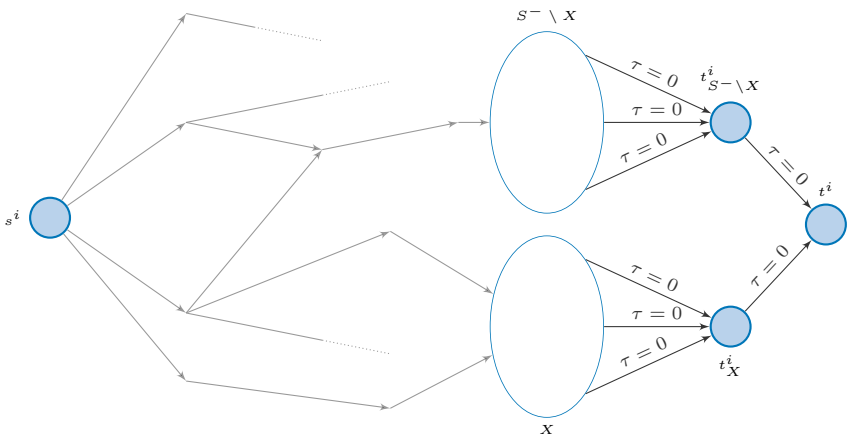


Figure 6.4: An illustration of the time layer $i \in \{1, \dots, r\}$ of $\overline{\mathcal{N}}^\theta$ as described above.

2. *The flow over time f induces a static lex-max flow in $\overline{\mathcal{N}}^T$ with respect to the total order*

$$s^* \prec t^T_{S^-\setminus X} \prec t^T_X \prec t^{T-1}_{S^-\setminus X} \prec t^{T-1}_X \prec \dots \prec t^1_{S^-\setminus X} \prec t^1_X$$

if T is an integral time horizon. For a rational time horizon $T = p/q$ with $p, q \in \mathbb{Z}_{>0}$, the order on the terminals is defined similarly according to the considered time layers corresponding to $\mathcal{I} = \{1/q, 2/q, \dots, p/q\}$.

Proof. Assume at first that T is integral and that f is a flow over time in $\mathcal{N}$ with time horizon T that satisfies $\gamma^T(X)$ for $X \subseteq S^-$. We start by showing that the second statement of the lemma holds. Let x_f be the static flow in $\overline{\mathcal{N}}^T$ induced by f according to Lemma 2.20. As the flow over time f satisfies $\gamma^T(X)$, it has pattern $\theta \mapsto o^\theta(\{s\})$ for $\theta \in [0, T)$ by definition. That is, f has the maximal possible value for each time horizon $0 \leq \theta \leq T$. By Lemma 2.20 (and also by the results of Minieka [Min73] and Megiddo [Meg79]) we know that the static flow x_f is a lex-max flow in $\overline{\mathcal{N}}^T$ with respect to the order $\prec'$ given by $t^T \prec' t^{T-1} \prec' \dots \prec' t^1$and hence Lemma 2.19 yields,

$$-\operatorname{net}_{x_f}(\{t^1, t^2, \dots, t^i\}) \overset{\text{Lem.2.19}}{=} \max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, t^2, \dots, t^i\}) \text{ for all } i \in \{1, \dots, T\}. \tag{6.7}$$

Satisfying $\gamma^T(X)$ also means that f sends as much flow towards the sinks in X until time T as is possible in a flow over time with pattern $\theta \mapsto o^\theta(\{s\})$, for $\theta \in [0, T)$. For the static flow x_f this translates to the fact that x_f sends as much flow towards the sinks $\{t^1_X, t^2_X, \dots, t^T_X\}$ as is possible in a static lex-max flow in $\overline{\mathcal{N}}^T$ with respect to the order $\prec'$ on $\{t^1, t^2, \dots, t^T\}$. Note that for all $i \in \{1, \dots, t\}$ the static flow x_f fulfills

$$-\operatorname{net}_{x_f}(t^i_X) \leq \max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, t^2, \dots, t^{i-1}, t^i_X\}) - \max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, t^2, \dots, t^{i-1}\}), \tag{6.8}$$

because assuming $>$ in (6.8) implies

$$\begin{aligned} -\operatorname{net}_{x_f}(t^i_X) + \max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, \dots, t^{i-1}\}) &\overset{(6.7)}{=} -\operatorname{net}_{x_f}(\{t^1, t^2, \dots, t^{i-1}, t^i_X\}) \\ &> \max_{\overline{\mathcal{N}}^T}(\{s^*\}, \{t^1, t^2, \dots, t^{i-1}, t^i_X\}), \end{aligned}$$

which is a contradiction. We also know that a lex-max flow with respect to the order

$$s^* \prec t^T_{S^-\setminus X} \prec t^T_X \prec t^{T-1}_{S^-\setminus X} \prec t^{T-1}_X \prec \dots \prec t^1_{S^-\setminus X} \prec t^1_X$$

fulfills all inequalities in (6.8) with equality (see Lemma 2.19). That is, a static lex-max flow in $\overline{\mathcal{N}}^T$ with respect to the order $\prec'$ that simultaneously sends as much flow as possible in such a lex-max flow to the sinks in $\{t^1_X, t^2_X, \dots, t^T_X\}$ has to be a lex-max flow with respect to the order $\prec$. This

shows the second statement of the lemma for integral T. We proceed to show the first statement. Aiming for a contradiction, assume that f does not satisfy $\gamma^\theta(X)$ for some integral $\theta \in \{1, 2, \dots, T-1\}$. For the static flow x_f this implies that the amount of flow sent towards $\{t_X^i \mid i \in \{1, 2, \dots, \theta\}\}$ is not the maximal amount of flow that can be sent towards these sinks in a static lex-max flow with respect to the order $\prec'$ in $\overline{\mathcal{N}}^T$. In particular, this implies that at least one of the inequalities in (6.8) is not fulfilled with equality implying that x_f is not a lex-max flow with respect to the order $\prec$, contradiction. Hence, the flow over time f satisfies $\gamma^\theta(X)$ for all $\theta \in \{1, 2, \dots, T\}$. To show the statement for all rational times $\theta \in [0, T)$, we use a finer discretization of time. Let $\theta = \overline{p}/\overline{q}$ with $\overline{p}, \overline{q} \in \mathbb{Z}_{>0}$. We now discretize time as follows,

$$\overline{\mathcal{I}} \coloneqq \{1/\overline{q}, 2/\overline{q}, \dots, \overline{p}/\overline{q}, \dots, T\},$$

and also assume that our time-expanded network is built with respect to this discretization. The same argument as above yields that f satisfies $\gamma^T(X)$ for all $\theta \in \overline{\mathcal{I}}$ showing that $\gamma^\theta(X)$ is satisfied for all rational times θ.
Next assume that there is an irrational time $\overline{\theta}$ with $-\operatorname{net}_f(X, \overline{\theta}) < \gamma^{\overline{\theta}}(X)$. Let $\overline{f}$ be a flow over time in $\mathcal{N}$ with time horizon $\overline{\theta}$, pattern $\theta \mapsto o^\theta(\{s\})$ and $-\operatorname{net}_{\overline{f}}(X, \overline{\theta}) = \gamma^{\overline{\theta}}(X)$. Such a flow over time exists according to the definition of the function $\gamma^{\overline{\theta}}$. By definition of a flow over time f is a continuous function in θ, which implies that $\operatorname{net}_f(X, \theta)$ is continuous in θ. The same holds for the flow over time $\overline{f}$. We have

$$-\operatorname{net}_f(X, \overline{\theta}) < -\operatorname{net}_{\overline{f}}(X, \overline{\theta}).$$

Because of the continuity, there is an $\varepsilon > 0$ with

$$-\operatorname{net}_f(X, \theta') < -\operatorname{net}_{\overline{f}}(X, \theta') \leq \gamma^{\theta'}(X) \text{ for all } \theta' \in [\overline{\theta} - \varepsilon, \overline{\theta}].$$

This contradicts the fact that $-\operatorname{net}_f(x, \theta) = \gamma^\theta(X)$ for each rational θ since the rational numbers are a dense set in the set of real numbers. We have now shown all our statements for an integral time horizon T. To show the lemma for rational time horizons T we again just use a suitable discretization of time and construct the time-expanded network with respect to this discretization. The rest of the proof works completely similarly. □

See Figure 6.5 for an illustration of Lemma 6.12. The following corollary follows directly from the (proof of the) previous lemma.

Corollary 6.13. *Let $\mathcal{N}$ be a dynamic network with only a single source s, $T \geq 0$ a rational time horizon and $X \subseteq S^-$. Further, $\overline{\mathcal{N}}^T$ denotes the time-expanded network constructed as above. We have*

$$\gamma^T(X) = \sum_{i=1}^{T} \max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, \dots, t^{i-1}, t_X^i\}) - \max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, \dots, t^{i-1}\}),$$

if T is integral. For a rational T we again need to choose a suitably discretized time-expanded network to obtain the same result with respect to the chosen discretization.

It turns out that also the other direction of Lemma 6.12 holds: A suitable lex-max flow in the time-expanded network induces a lex-max earliest arrival flow.

Lemma 6.14. *Let $\mathcal{N}$ be a dynamic network and $T = p/q$ with $p, q \in \mathbb{Z}_{>0}$ a rational time horizon. If x is a static lex-max flow in $\overline{\mathcal{N}}^T$ with respect to the total order*

$$s^* \prec t_{S^- \setminus X}^{p/q} \prec t_X^{p/q} \prec t_{S^- \setminus X}^{(p-1)/q} \prec t_X^{(p-1)/q} \prec \dots \prec t_{S^- \setminus X}^{1/q} \prec t_X^{1/q},$$

it induces a flow over time f in $\mathcal{N}$ with time horizon T satisfying $\gamma^\theta(X)$ for each $\theta \in [0, T)$.

Proof. For the simplification of notation we only consider integral time horizons in our proof. Everything works completely similar for rational time horizons. Let f be the flow over time induced

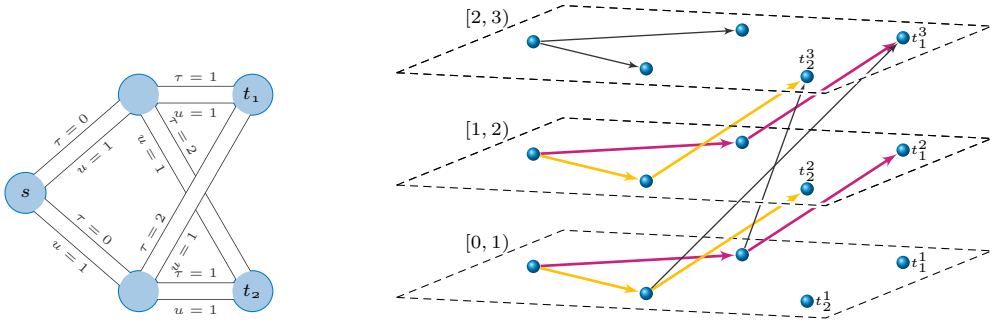


Figure 6.5: The time-expanded network corresponding to the depicted dynamic network. The visualized flow is the static lex-max flow according to Lemma 6.12 induced by a flow over time that satisfies $\gamma^3(\{t_1\})$. It can be seen that this static flow is in fact a lex-max flow with respect to the order given in Lemma 6.12. On the other hand does the depicted static flow induce a flow over time that satisfies $\gamma^3(\{t_1\})$ (see Lemma 6.14). Note that we did not visualize the overall super-source of the time-expanded network.

by the static lex-max flow x according to Lemma 2.20. By construction and Corollary 6.13, we obtain for our flow over time f,

$$\begin{aligned} -\operatorname{net}_f(X,\theta) &= \sum_{i=1}^{\theta} \operatorname{net}_{x_f}(t_X^i) \\ &\overset{\text{Lem. 2.20}}{=} \sum_{i=1}^{\theta} (\max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, \dots, t^{\theta-1}, t_X^{i-1}\}) - \max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, \dots, t^{i-1}\})) \\ &\overset{\text{Cor. 6.13}}{=} \gamma^\theta(X) \text{ for all } \theta \in \{0, 1, \dots, T\}. \end{aligned}$$

By definition of f it also holds that $|f|_\theta = o^\theta(\{s\})$ for all $\theta \in \{0, 1, \dots, T\}$. Corollary 6.2 implies that the function $\theta \mapsto o^\theta(\{s\})$ is piecewise linear and since all transit times are integral, breakpoints of this function only occur at integral points in time. However, by our construction of f from x the function $\theta \mapsto |f|_\theta$ is also a piecewise linear function with breakpoints only occurring at integral points in time. Thus, we have $|f|_\theta = o^\theta(\{s\})$ for all $\theta \leq T$ and hence by our arguing above the flow over time f satisfies $\gamma^\theta(X)$ for each integral $\theta \in \{1, 2, \dots, T\}$. By the first statement of Lemma 6.12, $\gamma^\theta(X)$ is satisfied by f for all $\theta \in [0, T)$. □

Again, see Figure 6.5 for an illustration of this lemma. The following corollaries are consequences of Lemmas 6.12 and 6.14.

Corollary 6.15. *A flow over time f in $\mathcal{N}$ with time horizon T and pattern $\theta \mapsto o^\theta(\{s\})$ for $\theta \in [0, T)$ such that*

$$-\operatorname{net}_f(X, \theta') = o^{\theta'}(\{s\} \cup S^- \setminus X)$$

for some rational $\theta' < T$ and $X \subseteq S^-$ fulfills

$$-\operatorname{net}_f(X, \theta'') = o^{\theta''}(\{s\} \cup S^- \setminus X) \text{ for all } \theta'' \in [0, \theta').$$

Proof. Assume at first that θ' is integral. Let x_f be the static flow in $\overline{\mathcal{N}}^{\theta'}$ induced by the flow over time f. Lemma 6.12 implies (note that here we have $\gamma^{\theta'}(X) = o^T(\{s\} \cup S^- \setminus X)$) that x_f is a static lex-max flow in $\overline{\mathcal{N}}^{\theta'}$ with respect to the total order

$$s^* \prec t_{S^- \setminus X}^{\theta'} \prec t_X^{\theta'} \prec t_{S^- \setminus X}^{\theta'-1} \prec t_X^{\theta'-1} \prec \dots \prec t_{S^- \setminus X}^1 \prec t_X^1.$$

This implies by Lemma 2.19

$$\begin{aligned} -\operatorname{net}_{x_f}(t_X^{\theta'}) &\overset{\text{Lem. 2.19}}{=} \max_{\overline{\mathcal{N}}^{\theta'}}(s^*, \{t^1, \dots, t^{\theta'-1}, t_X^{\theta'}\}) - \max_{\overline{\mathcal{N}}^{\theta'}}(s^*, \{t^1, \dots, t^{\theta'-1}\}) \\ &= \max_{\overline{\mathcal{N}}^{\theta'}}(s^*, \{t^1, \dots, t^{\theta'-1}, t_X^{\theta'}\}) - o^{\theta'-1}(\{s\}). \end{aligned} \tag{6.9}$$

Let y be a static lex-max flow in $\overline{\mathcal{N}}^{\theta'}$ with respect to the total order $\prec'$ given by

$$s^* \prec' t_X^{\theta'} \prec' t_{S^- \setminus X}^{\theta'-1} \prec' t_{S^- \setminus X}^{\theta'-2} \prec' \dots \prec' t_{S^- \setminus X}^{1} \prec' t_X^{\theta'-1} \prec' t_X^{\theta'-2} \prec' \dots \prec' t_X^1.$$

We have

$$\begin{aligned} -\operatorname{net}_y(t_X^{\theta'}) &\overset{\text{Lem. 2.19}}{=} \max_{\overline{\mathcal{N}}^{\theta'}}(s^*, t^1, t^2, \dots, t^{\theta'-1}, t_X^{\theta'}) - o^{\theta'-1}(\{s\}) \\ &\overset{(6.9)}{=} -\operatorname{net}_{x_f}(t_X^{\theta'}). \end{aligned} \tag{6.10}$$

On the other hand, it also holds that

$$-\operatorname{net}_y(\{t_X^1, t_X^2, \dots, t_X^{\theta'-1}\}) = o^{\theta'-1}(\{s\} \cup S^- \setminus X), \tag{6.11}$$

which implies

$$\begin{aligned} -\operatorname{net}_y(t_X^{\theta'}) &= -\operatorname{net}_y(t_X^1, t_X^2, \dots, t_X^{\theta'}) + \operatorname{net}_y(t_X^1, t_X^2, \dots, t_X^{\theta'-1}) \\ &\overset{(6.11)}{\leq} o^{\theta'}(\{s\} \cup S^- \setminus X) - o^{\theta'-1}(\{s\} \cup S^- \setminus X). \end{aligned}$$

Combining this with (6.10) yields

$$-\operatorname{net}_{x_f}(t_X^{\theta'}) \overset{(6.10)}{=} -\operatorname{net}_y(t_X^{\theta'}) \overset{(6.11)}{\leq} o^{\theta'}(\{s\} \cup S^- \setminus X) - o^{\theta'-1}(\{s\} \cup S^- \setminus X). \tag{6.12}$$

By assumption, we have that $-\operatorname{net}_{x_f}(\{t_X^1, \dots, t_X^{\theta'}\}) = o^{\theta'}(\{s\} \cup S^- \setminus X)$. This implies

$$\begin{aligned} -\operatorname{net}_{x_f}(\{t_X^1, \dots, t_X^{\theta'-1}\}) &= -\operatorname{net}_{x_f}(\{t_X^1, \dots, t_X^{\theta'}\}) + \operatorname{net}_{x_f}(t_X^{\theta'}) \\ &= o^{\theta'}(\{s\} \cup S^- \setminus X) + \operatorname{net}_{x_f}(t_X^{\theta'}) \\ &\overset{(6.12)}{\geq} o^{\theta'-1}(\{s\} \cup S^- \setminus X), \end{aligned}$$

i.e., $-\operatorname{net}_f(X, \theta' - 1) = o^{\theta'-1}(\{s\} \cup S^- \setminus X)$. Proceeding inductively yields that $-\operatorname{net}_f(X, \theta'') = o^{\theta''}(\{s\} \cup S^- \setminus X)$ for each integral $\theta'' \leq \theta'$. For non-integral but rational $\theta' \geq 0$ we can again show the statement of the corollary by using a suitable finer discretization of time. □

Corollary 6.16. *For a fixed $X \subseteq S^-$ the function $\theta \mapsto \gamma^\theta(X)$ is continuous in θ.*

We are finally ready to prove Theorem 6.11.

Proof of Theorem 6.11. By Corollary 6.13 we get for an integral time horizon $T \geq 0$,

$$\gamma^T(X) = \sum_{i=1}^{T} \max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, \dots, t^i, t_X^i\}) - \max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, \dots, t^i\}).$$

Note that $\max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, \dots, t^i\})$ is independent of X and thus, in order to show submodularity it suffices to show that $g^\theta \colon 2^{S^-} \to \mathbb{R}$ with

$$g^\theta(X) \coloneqq \max_{\overline{\mathcal{N}}^T}(s^*, \{t^1, \dots, t^i, t_X^i\}) \text{ for } X \subseteq S^-,$$

is submodular on S^- for all $\theta \in \{1, \dots, T\}$. To show this fact, fix some integral θ, and $A \subseteq B \subseteq S^-$ and $v \in S^- \setminus B$. We now again redefine the time-expanded network $\overline{\mathcal{N}}^\theta$. In each time layer i we now attach a super-sink t_v^i to v^i in layer i, a super-sink t_A^i to the nodes corresponding to the sinks in A in layer i, a super-sink $t_{B \setminus A}^i$ to the nodes corresponding to the sinks in $B \setminus A$ in layer i and a

super-sink $t^i_{S^- \setminus (B \cup \{v\})}$ to the nodes corresponding to the remaining sinks in layer i.
Then, $g^\theta(A \cup \{v\}) - g^\theta(A)$ is exactly the amount of flow that reaches t^θ_v in a lex-max flow in $\overline{\mathcal{N}}^\theta$ with respect to the order $\prec$ given by

$$s^* \prec t^\theta_{S^- \setminus (B \cup \{v\})} \prec t^\theta_{B \setminus A} \prec t^\theta_v \prec t^\theta_A \prec \dots t^{\theta-1} \prec \dots \prec t^1.$$

Similarly, $g^\theta(B \cup \{v\}) - g^\theta(B)$ is exactly the amount of flow that reaches t^θ_v in a static lex-max flow in $\overline{\mathcal{N}}^\theta$ with respect to

$$s^* \prec t^\theta_{S^- \setminus (B \cup \{v\})} \prec t^\theta_v \prec t^\theta_{B \setminus A} \prec t^\theta_A \prec \dots t^{\theta-1} \prec \dots \prec t^1.$$

Since the sink t^θ_v has a lower priority in the first order, we obtain

$$g^\theta(B \cup \{v\}) - g^\theta(B) \leq g^\theta(A \cup \{v\}) - g^\theta(A),$$

and thus submodularity. It remains to show submodularity for non-integral time horizons T. The submodularity for arbitrary rational time horizons follows again by using a finer discretization of time. Let $\theta' > 0$ an irrational time horizon with

$$\gamma^{\theta'}(B \cup \{v\}) - \gamma^{\theta'}(B) > \gamma^{\theta'}(A \cup \{v\}) - \gamma^{\theta'}(A)$$

for some $A \subseteq B \subseteq S$ and $v \in S^- \setminus B$. By the continuity of $\theta \mapsto \gamma^\theta(X)$ for $X \subseteq S^-$ there is an $\varepsilon > 0$ such that

$$\gamma^\theta(B \cup \{v\}) - \gamma^\theta(B) > \gamma^\theta(A \cup \{v\}) - \gamma^\theta(A) \text{ for all } \theta \in (\theta' - \varepsilon, \theta' + \varepsilon), \tag{6.13}$$

contradicting the submodularity of γ^T for all rational time horizons T. □

As we now know that γ^T is a submodular function, it makes sense to consider the base polytope $\mathcal{B}(\gamma^T)$ of γ^T. Recall, that it is our main goal to devise an efficient method to check whether $x \in \mathcal{S}^{\text{EAT}}(\mathcal{N}, T)$ for some $x \in \mathbb{Z}^{S^-}_{\geq 0}$. We can now make a first observation regarding the set $\mathcal{S}^{\text{EAT}}(\mathcal{N}, T)$, namely that we have

$$\mathcal{S}^{\text{EAT}}(\mathcal{N}, T) \subseteq \mathcal{B}(\gamma^T).$$

To see this, recall that

$$\mathcal{B}(\gamma^T) = \{x \in \mathbb{R}^{S^-} \mid x(X) \leq \gamma^T(X) \text{ for all } X \subseteq S^- \text{ and } x(S^-) = \gamma^T(S^-) = -b(S^-)\}.$$

The existence of $x \in \mathcal{S}^{\text{EAT}}(\mathcal{N}, T)$ with $x(X) > \gamma^T(X)$ for some $X \subseteq S^-$ contradicts the existence of an earliest arrival transshipment solving $(\mathcal{N}, -x)_{\text{EAT}}$ by the definition of γ^T and the fact that the pattern of a tight problem $(\mathcal{N}, -x)_{\text{EAT}}$ is given by $\theta \mapsto o^\theta(\{s\})$ for $\theta \in [0, T)]$ (Corollary 6.5). To show the other inclusion, we again prove that the vertices of $\mathcal{B}(\gamma^T)$ correspond to a certain class of flows over time in $\mathcal{N}$ with time horizon T and pattern $\theta \mapsto o^\theta(\{s\})$, which we call **lex-max earliest arrival flows**, because they fulfill the earliest arrival pattern corresponding to a tight earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$.

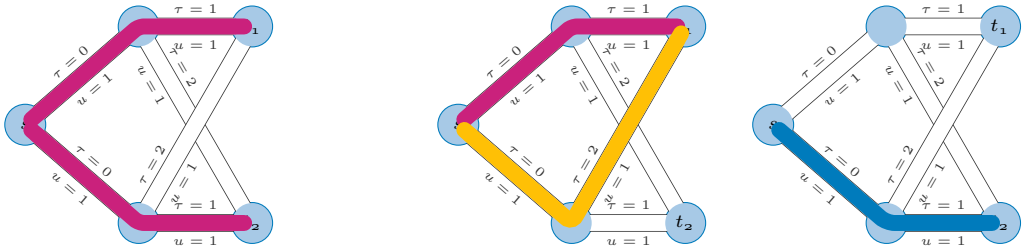
6.2.2 Lexicographically Maximum Earliest Arrival Flows

In this section we focus on the second step of the outline from the beginning of this section. Lex-max earliest arrival flows are quite similar to lex-max flows over time: they are flows over time corresponding to a given order on the sinks S^- and the goal is to send the flow in such a way that the flow arriving at the sinks is maximized with respect to the given order. In contrast to lex-max flows over time, we now add the additional constraint that the flow over time is supposed to have pattern $\theta \mapsto o^\theta(\{s\})$. This leads to the following definition:

Definition 6.17 (Lex-max earliest arrival flows).
Let $\mathcal{N}$ be a dynamic network with a single source s and multiple sinks S^-, let $T \geq 0$ be a time horizon, and $\prec$ a total order on the set of sinks. We call a flow over time f with time horizon T a ***lex-max earliest arrival flow*** *with respect to $\prec$ if f fulfills the following conditions:*

- *The flow over time f has pattern $\theta \mapsto o^\theta(\{s\})$ for $\theta \in [0, T)$.*
- *The amount of flow sent into the sinks is maximized in decreasing order $\prec$ while respecting the pattern.*

See Figure 6.6 for an example of a lex-max earliest arrival flow which also shows the difference between lex-max flows over time and lex-max earliest arrival flows. Note that in dynamic networks



(a) A lex-max earliest arrival flow with respect to $t_2 \prec t_1$ in this dynamic network (see also Figure 6.3) sends flow along the indicated magenta paths at rate one.

(b) A lex-max flow over time with respect to $t_2 \prec t_1$ in this dynamic network (see also Figure 6.3) sends flow into the indicated magenta path during $[0, 2)$, into the yellow path during $[0, 1)$ and into the blue path during $[1, 2)$ at rate one.

Figure 6.6: An example for the fact that lex-max flows over time and earliest arrival lex-max flows with respect to the same order are in general not equal.

with a single sink and multiple sources we can give a similar definition of lex-max earliest arrival flows. However, in this case lex-max earliest arrival flows are always a special case of lex-max flows over time, which ultimately leads to the fact that tight earliest arrival transshipment problems in dynamic networks with only a single sink can be solved by a convex combination of lex-max flows over time that satisfy the earliest arrival pattern. Intuitively, the fact that in dynamic networks with a single source but multiple sinks a lex-max earliest arrival flow is in general *not* a lex-max flow over time is the reason that not each tight earliest arrival transshipment problem in such a network has a solution. Before we present a polynomial space algorithm to compute arbitrary lex-max earliest arrival flows, we first deduce some properties of such flows over time. By the definition of γ^T (Definition 6.10) and Definition 6.17, a flow over time f with time horizon T, pattern $\theta \mapsto o^\theta(\{s\})$ for $\theta \in [0, T)$, and

$$-\operatorname{net}_f(\{t' \in S^- \mid t \preceq t'\}, T) = \gamma^T(\{t' \in S^- \mid t \preceq t'\})$$

for all $t \in S^-$ is a lex-max earliest arrival flow with respect to $\prec$. We will argue in the following that a flow over time f with these properties does always exist.

Lemma 6.18. *Let $\mathcal{N}$ be a dynamic network with only a single source s and sinks $S^- = \{t_1, t_2, \dots, t_k\}$, $T = p/q \geq 0$ a rational time horizon with $p \geq 0$ and $p > 0$, and $\prec$ a total order on the sinks in S^- such that $t_1 \prec t_2 \prec \dots \prec t_k$. A static lex-max flow in $\mathcal{N}^T$ with respect to the total order $\prec'$ given by*

$$s^* \prec' t_1^{p/q} \prec' \dots \prec' t_k^{p/q} \prec' t_1^{(p-1)/q} \prec' \dots \prec' t_k^{(p-1)/q} \prec' \dots \prec' t_1^{1/q} \prec' \dots \prec' t_k^{1/q}$$

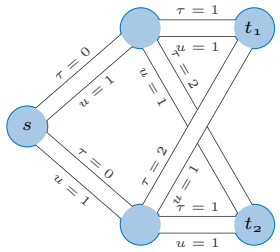
induces a flow over time f with time horizon T in $\mathcal{N}$ with

$$-\operatorname{net}_f(\{t' \in S^- \mid t \preceq t'\}) = \gamma^T(\{t' \in S^- \mid t \preceq t'\})$$

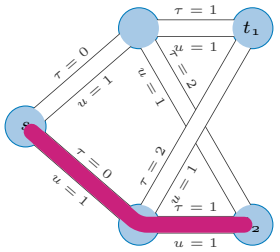
for all $t \in S^-$. In particular, the flow over time f is a lex-max earliest arrival flow with respect to $\prec$.

Proof. Denote by x the static lex-max flow in $\mathcal{N}^T$ with respect to the total order $\prec'$. Lemma 6.14 immediately implies that the induced flow over time f has the required properties. □

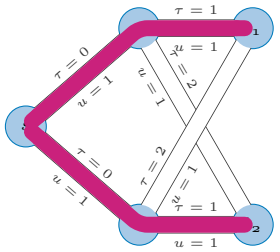
Lemma 6.18 also results in an algorithm for computing lex-max earliest arrival flows. However, this algorithm relies on computing static lex-max flows in the time-expanded network and is thus not applicable in praxis as it requires a pseudo-polynomial amount of space in the worst case. The final result of this section will be a polynomial space algorithm for computing lex-max earliest arrival flows that does not require any expansion of the original dynamic network. The intuition behind our algorithm for computing lex-max earliest arrival flows (see Algorithm 24) is pretty straightforward. When computing a lex-max earliest arrival flow with respect to some given order $\prec$ and a time horizon T, the first objective is of course to compute a flow over time f with pattern $\theta \mapsto o^\theta(\{s\})$ for $\theta \in [0, T)$. It was already shown by Wilkinson [Wil71] that the successive shortest path algorithm can be used to compute such a flow over time (see also Section 3.2). This is why the base of our algorithm is the successive shortest path algorithm. However, as we want the flow arriving at the sinks to respect the order $\prec$, our implementation of the successive shortest path algorithm has the additional feature that in each iteration the shortest path is chosen with respect to the order $\prec$. That is, if in iteration i there is a current shortest path leading towards t_k, we choose this path. If there is no shortest path leading towards t_k, it is checked whether there is a shortest path towards t_{k-1} and in case of existence this path is chosen, and so on. Here, we assume that $S^- = \{t_1, \dots, t_k\}$ with $t_1 \prec \dots t_k$. See Figure 6.7 and Figure 6.8 for examples showing the execution of Algorithm 24.



(a) A dynamic network $\mathcal{N}$ in which the sinks are ordered by $\prec$ as $t_1 \prec t_2$.



(b) In iteration 1 of Algorithm 24, there are two shortest paths from s to $\{t_1, t_2\}$. Since $t_1 \prec t_2$, the shortest path towards t_2 is chosen.



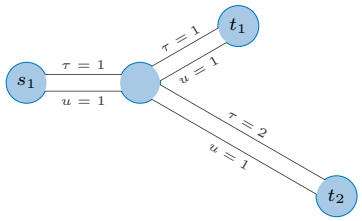
(c) In iteration 2 of Algorithm 24, there is only one shortest path from s to $\{t_1, t_2\}$ which leads towards t_1. In this iteration the difference to usual lex-max flows can be seen: In a lex-max flow over time with respect to $\prec$ flow would be sent along the (longer) path towards t_2. This however, would violate the earliest arrival property.

Figure 6.7: An example for the execution of Algorithm 24

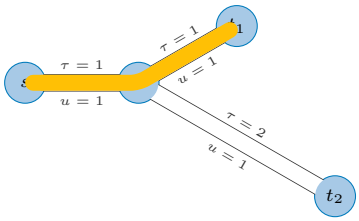
It is immediate that the flow over time f returned by Algorithm 24 is a feasible flow over time with time horizon T and pattern $\theta \mapsto o^\theta(\{s\})$ for all $\theta \in [0, T)$. It is, however, not inherently clear that f is a lex-max earliest arrival flow.

Theorem 6.19 (Correctness of Algorithm 24).
Let $\mathcal{N}$ be a dynamic network with only a single source, $T \geq 0$ a time horizon, and $\prec$ a total order on the set of sinks S^-. The flow over time f returned by Algorithm 24 with respect to these parameters is a lex-max earliest arrival flow with respect to T and $\prec$.

Proof. By construction f has time horizon T and the required pattern. It remains to prove that the flow over time f is in fact a lex-max earliest arrival flow. For simplicity assume that T is integral. The proof for a rational time horizon works completely similar. Again assume that $S^- = \{t_1, t_2, \dots, t_k\}$



(a) A dynamic network $\mathcal{N}$ in which the sinks are ordered by $t_1 \prec t_2$.



(b) In the first iteration of Algorithm 24 there is only one shortest path from s to $\{t_1, t_2\}$ which leads towards t_1. Thus, even if t_1 is ordered before t_2 this path is chosen in this iteration in order to maintain the earliest arrival property. After this first iteration, the algorithm terminates.

Figure 6.8: An example for the execution of Algorithm 24

Algorithm 24: Algorithm for computing lex-max earliest arrival flows

---

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, s, S^-)$, a time horizon $T \geq 0$, and a total order $\prec$ on S^-

Output : A lex-max earliest arrival flow f with respect to T and $\prec$

$x_P \leftarrow 0$ for all $P \in \overleftrightarrow{\mathcal{P}}$

$x \leftarrow$ static s-S^- flow with generalized path decomposition $(x_P)_{P \in \overleftrightarrow{\mathcal{P}}}$

while $d(\mathcal{N}_x, s, S^-) < T$ **do**

  $l \leftarrow d(\mathcal{N}_x, s, S^-)$

  for $i = k, k-1, \ldots, 1$ **do**

    while $d(\mathcal{N}_x, s, t_i) = l$ **do**

      $P \leftarrow$ shortest s-t_i path in $\mathcal{N}_x$

      $\gamma \leftarrow \min\{\tau(a) \mid a \in P\}$

      augment x along P by γ

    end

  end

end

$f \leftarrow$ generalized temporally repeated flow with time horizon T corresponding to $(x_P)_{P \in \overleftrightarrow{\mathcal{P}}}$

---

with $t_1 \prec t_2 \prec \ldots \prec t_k$. Denote by x_f the static flow in $\mathcal{N}^T$ induced by f. Our goal is to show that x_f is a static lex-max flow in $\mathcal{N}^T$ with respect to the order

$$s^* \prec' t_1^T \prec' t_2^T \prec' \ldots t_k^T \prec' t_1^{T-1} \prec' t_2^{T-1} \prec' \ldots \prec' t_k^{T-1} \prec' \ldots \prec' t_1^1 \prec' \ldots \prec' t_k^1.$$

Essentially, for all $l \in \{1, 2, \ldots, T-1\}$ the algorithm iterates over all the sinks and augments the flow towards these sinks along shortest paths of length l within the given order. For each $l \in \{1, \ldots, T-1\}$ and each $i \in \{k, k-1, \ldots, 1\}$ we construct a slight adaption $\mathcal{N}_{l,i}$ of the dynamic network $\mathcal{N}$, as follows:

- We add a super-sink t to the network that we attach to each of the terminals in $\{t_i, t_{i+1}, \ldots, t_k\}$ by arcs with zero transit time and infinite capacity. The sinks in $\{t_1, \ldots, t_{i-1}\}$ are attached to the sink t by arcs with transit time 1 and infinite capacity.
- We add an arc (t, s) with infinite capacity and transit time $-l$ to the network.

We denote by $x_{l,i}$ the static flow obtained after iteration i for length l. Clearly, the static flow $x_{l,i}$ can also be regarded as a static flow in $\mathcal{N}_{l,i}$. Our first observation is that this flow is a minimum-cost circulation in $\mathcal{N}_{l,i}$. This is true because we always augment along shortest paths, and the fact that we might not yet have augmented along paths of length l towards $t_1, \ldots, t_{i-1}$ is compensated by connecting those sinks with t by arcs with transit time 1. Thus, by Fact 2.21 the static flow $x_{l,i}$ induces a maximum flow over time, $f_{l,i}$ in $\mathcal{N}_{l,i}$ with time horizon l. Regarded as a flow over time in $\mathcal{N}$ $f_{l,i}$ has the property that flow arrives at the sinks $t_1, \ldots, t_{i-1}$ only until time $l-1$. That is, in

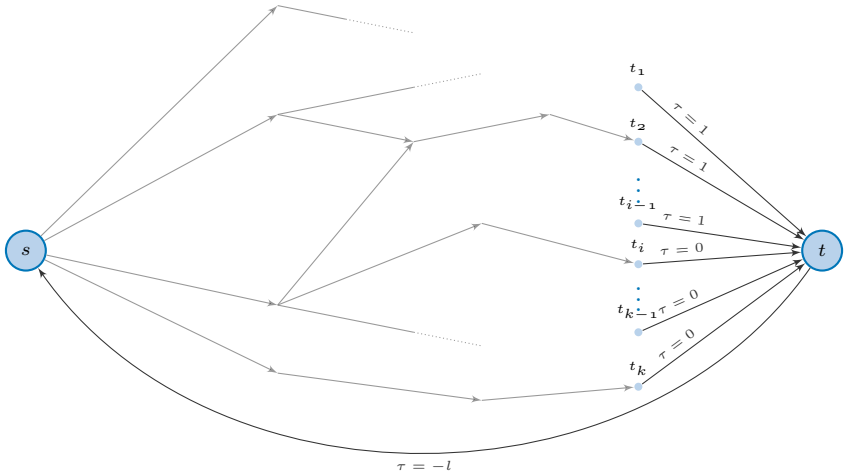


Figure 6.9: The dynamic network $\mathcal{N}_{l,i}$

$\mathcal{N}$ the flow over time $f_{l,i}$ is a flow over time with maximal value with the property that the sinks $t_1, \dots, t_{i-1}$ receive flow until time $l-1$, while the sinks $t_k, \dots, t_i$ receive flow until time l. Since in the course of the algorithm the amount of flow that arrives at each of the sinks $t_i, \dots, t_k$ until time l or at each of the sinks $t_1, \dots, t_{i-1}$ until time $l-1$ is not changed, we thus obtain

$$-\operatorname{net}_{x_f}(\{t_i^l, t_{i-1}^l, \dots, t_1^l, \dots, t_k^l, \dots, t_1^1\}) = \max_{\mathcal{N}^T}(s^*, \{t_i^l, t_{i-1}^l, \dots, t_1^l, \dots, t_k^l, \dots, t_1^1\}).$$

Thus, x_f is a static lex-max flow in $\mathcal{N}^T$ with respect to the order $\prec'$. According to Lemma 6.18 the flow over f' induced by x_f is a lex-max earliest arrival flow. All in all we have for all $t \in S^-$,

$$\begin{aligned} -\operatorname{net}_f(\{t' \in S^- \mid t' \preceq t\}, T) &= \sum_{\theta=1}^{T} -\operatorname{net}_{x_f}(\{t'^\theta \mid t' \in S^- \text{ s.t. } t' \preceq t\}) \\ &= -\operatorname{net}_{f'}(\{t' \in S^- \mid t' \preceq t\}, T) \\ &= \gamma^T(\{t' \in S^- \mid t' \preceq t\}). \end{aligned}$$

That is, according to Lemma 6.18, the flow over time f is a lex-max earliest arrival flow with time horizon T with respect to the order $\prec$. □

Another feature of Algorithm 24 is that it can immediately be used to evaluate the submodular function γ^θ at arbitrary sets $X \subseteq S^-$ and for all $\theta \geq 0$.

Corollary 6.20. *Using Algorithm 24, the submodular function γ^θ can be evaluated in PSPACE for all $\theta \geq 0$.*

Proof. Fix $\theta \geq 0$ and assume that we want to evaluate the submodular function γ^θ at $X \subseteq S^-$. We can just choose any order $\prec$ on S^- with the property that for all $t \in X$ and $t' \in S^- \setminus X$ we have $t' \prec t$. Denote by f the generalized lex-max earliest arrival flow with respect to $\prec$ and time horizon θ. It has the property that

$$-\operatorname{net}_f(X, \theta) = \gamma^\theta(X).$$

Thus, we can, during the course of Algorithm 24, sequentially compute the amount of flow sent towards X in order to evaluate γ^θ at X. □

Overall this finishes the first two steps towards our main result from the outline at the beginning of the section.

6.2.3 An Existence Criterion for Tight Earliest Arrival Transshipments and a PSPACE Algorithm

Using all the preliminaries we derived in the previous section, we are finally able to state our main result: an existence criterion and a structural result regarding tight earliest arrival transshipments in dynamic networks with only a single source.

Theorem 6.21 (Structure of Earliest Arrival Transshipments).
Let $\mathcal{N} = (D = (V, A), u, \tau, s, S^-)$ be a dynamic network with only a single source s and b a demand vector on the set of sinks S^- of $\mathcal{N}$ such that $(\mathcal{N}, b)_{\text{EAT}}$ is a tight earliest arrival transshipment problem with minimal feasible time horizon $T \geq 0$.
Then, we have $\mathcal{S}^{\text{EAT}}(\mathcal{N}, T) = \mathcal{B}(\gamma^T)$. In particular, an earliest arrival transshipment f solving $(\mathcal{N}, b)_{\text{EAT}}$ exists if and only if $-b \in \mathcal{B}(\gamma^T)$ and in case of existence such an earliest arrival transshipment can be obtained as a convex combination of $d \leq |S^-|$ many lex-max earliest arrival flows with time horizon T. That is, there are total orders $\prec_1, \dots, \prec_d$ on S^- and convex coefficients $\lambda_1, \dots, \lambda_d \geq 0$ such that

$$\lambda_1 f_{\prec_1} + \dots + \lambda_d f_{\prec_d}$$

is a flow over time solving $(\mathcal{N}, b)_{\text{EAT}}$. Here, $f_{\prec_i}$ is a lex-max earliest arrival flow with respect to T and $\prec_i$ for all $i \in \{1, \dots, d\}$.

To prove this theorem, we only need one preliminary lemma in which we derive a connection between the vertices of $\mathcal{B}(\gamma^T)$ and lex-max earliest arrival flows with time horizon T.

Lemma 6.22. *Let $\mathcal{N}$ be a dynamic network with only a single source s, $T \geq 0$ a time horizon and $\prec$ a total order on S^-. If $f_\prec$ is a lex-max earliest earliest arrival flow with respect to T and $\prec$ and $v^\prec$ is the vertex of $\mathcal{B}(\gamma^T)$ corresponding to $\prec$, we have*

$$-x_{f_\prec}(t) = v^\prec(t) \text{ for all } t \in S^-.$$

In particular,

$$\mathcal{B}(\gamma^T) = \operatorname{conv}(\{-x_{f_\prec} \mid \prec \text{ is a total order on } S^-\}).$$

Proof. Theorem 2.3 implies that each vertex of $\mathcal{B}(\gamma^T)$ corresponds to one or multiple total orders $\prec$ on S^-. Conversely, each such total order $\prec$ corresponds to a unique vertex $v^\prec$ of $\mathcal{B}(\gamma^T)$, which can be computed using the Greedy Algorithm 1. This implies

$$\mathcal{B}(o^T) = \operatorname{conv}(\{v^\prec \mid \prec \text{ is a total order on } S^-\}).$$

Moreover, each total order $\prec$ on S^- induces a lex-max earliest arrival flow with time horizon T and vice versa. Let $\prec$ be a fixed total order on S^- and assume that $f_\prec$ is the corresponding lex-max earliest arrival flow with time horizon T and $v^\prec$ is the vertex of $\mathcal{B}(\gamma^T)$ defined by $\prec$. Using Lemma 6.18, we get for all $t \in S^-$

$$\begin{aligned} x_{f_\prec}(t) &= \operatorname{net}_{f_\prec}(t, T) \\ &\overset{\text{Lem. 6.18}}{=} \gamma^T(t' \in S^- \mid t \prec t') - \gamma^T(t' \in S^- \mid t \preceq t'). \end{aligned}$$

Using Theorem 2.3, we also get for all $t \in S^-$,

$$v^\prec(t) = \gamma^T(t' \in S^- \mid t \preceq t') - \gamma^T(t' \in S^- \mid t \prec t').$$

Thus, we have $-x_{f_\prec} = v^\prec$ for each fixed total order $\prec$ on S^-. Overall, we thus have

$$\mathcal{B}(\gamma^T) = \operatorname{conv}(\{-x_{f_\prec} \mid \prec \text{ is a total order on } S^-\}),$$

which concludes the proof. □

With this lemma we are able to prove Theorem 6.21.

Proof of Theorem 6.21. We already argued that $\mathcal{S}^{\text{EAT}}(\mathcal{N},T) \subseteq \mathcal{B}(\gamma^T)$. We will now prove the other inclusion, i.e., we will show that for each $x \in \mathcal{B}(\gamma^T)$ the earliest arrival transshipment problem $(\mathcal{N},-x)_{\text{EAT}}$ has a solution.
Choose some $x \in \mathcal{B}(\gamma^T)$, i.e., we can obtain x as convex combination of vertices of $\mathcal{B}(\gamma^T)$. Let $\prec_1$ $,\ldots,\prec_d$ be total orders on S^- and $\lambda_1,\ldots,\lambda_d \geq 0$ convex coefficients such that $x = \lambda_1 v^{\prec_1} + \ldots + \lambda_d v^{\prec_d}$, where $v^{\prec_i}$ is the vertex of $\mathcal{B}(\gamma^T)$ corresponding to $\prec_i$ for all $i \in \{1,\ldots,d\}$. Define

$$f \coloneqq \lambda_1 f_{\prec_1} + \lambda_2 f_{\prec_2} + \ldots + \lambda_d f_{\prec_i}.$$

Here $f_{\prec_i}$ is the lex-max earliest arrival flow with time horizon T corresponding to $\prec_i$ for all $i \in \{1,\ldots,d\}$. Hence, the flow over time f has pattern $o^\theta(\{s\})$ and by Lemma 6.22 its characteristic vector is $-x$ and thus solves the problem $(\mathcal{N},-x)_{\text{EAT}}$. This also shows that in case of existence an earliest arrival transshipment solving $(\mathcal{N},b)_{\text{EAT}}$ can be obtained by a convex combination of lex-max earliest arrival flows. That $d \leq |S^-|$ follows with Carathéodory's theorem. □

We can now put all the results we achieved so far together to obtain a polynomial space algorithm that checks whether a given tight earliest arrival transshipment problem $(\mathcal{N},b)_{\text{EAT}}$ in a dynamic network $\mathcal{N}$ with only a single source s and minimal feasible time horizon $T \geq 0$ has a solution and that computes a suitable earliest arrival transshipment in case of existence (see Algorithm 25). According to Theorem 6.21, in order to check whether $(\mathcal{N},b)_{\text{EAT}}$ has a solution, it suffices to check whether $-b \in \mathcal{B}(\gamma^T)$. Thus, in order find out whether $(\mathcal{N},b)_{\text{EAT}}$ can be solved by an earliest arrival transshipment it is enough to test whether

$$\gamma^T(X) + b(X) \geq 0 \text{ for all } X \subseteq S^-,$$

which can be done by minimizing the submodular function $\gamma^T + b$. If the minimum is at least zero, an earliest arrival transshipment solving $(\mathcal{N},b)_{\text{EAT}}$ exists, otherwise it does not exist. By incorporating our PSPACE algorithm for evaluating γ^T into our favorite strongly polynomial time submodular function minimization algorithm we can thus test in PSPACE whether $(\mathcal{N},b)_{\text{EAT}}$ has a solution.

Corollary 6.23. *Given a tight earliest arrival transshipment problem $(\mathcal{N},b)_{\text{EAT}}$ in a dynamic network $\mathcal{N}$ with only a single source s we can check in polynomial space whether this problem has a solution.*

In order to come up with an earliest arrival transshipment solving $(\mathcal{N},b)_{\text{EAT}}$ in case of existence, Theorem 6.21 implies that a solution can be obtained by a convex combination of lex-max earliest arrival flows. Next, we will argue that a suitable convex combination of lex-max earliest arrival flows is essentially computed while minimizing the submodular function $\gamma^T + b$, provided an algorithm using the framework of Cunningham is used for the minimization.
When minimizing the submodular function $\gamma^T + b$ with an algorithm using the framework of Cunningham, then, besides the minimal value of the submodular function, also a convex combination of vertices of $\mathcal{B}(\gamma^T + b)$ giving the vector

$$x^* = \text{argmax}\{x^-(S^-) \mid x \in \mathcal{B}(\gamma^T)\},$$

is returned (see Section 4.3.2). More precisely, we get d total orders $\prec_1,\ldots,\prec_d$ on S^- and convex coefficients $\lambda_1,\ldots,\lambda_d \geq 0$ such that

$$x^* = \lambda_1 v^{\prec_1} + \ldots + \lambda_d v^{\prec_d}.$$

Here $v^{\prec_i}$ is the vertex of $\mathcal{B}(\gamma^T)$ corresponding to $\prec_i$ according to Theorem 2.3 for all $i \in \{1,\ldots,d\}$. Our first observation is that x^* is the zero vector if $(\mathcal{N},b)_{\text{EAT}}$ has a solution.

Lemma 6.24. *Given a tight earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a single source s and minimal feasible time horizon $T \geq 0$ it holds that if $(\mathcal{N}, b)_{\text{EAT}}$ has a solution, then*

$$x^* = \operatorname{argmax}\{x^-(S^-) \mid x \in \mathcal{B}(\gamma^T)\} = \mathbf{0}.$$

Proof. By assumption $(\mathcal{N}, b)_{\text{EAT}}$ has a solution, thus Theorem 6.21 implies that $-b \in \mathcal{B}(\gamma^T)$ and thus $\gamma(X) + b(X) \geq 0$ for all $X \subseteq S^-$. In particular, we have $\gamma^T(S^-) + b(S^-) = o^T(\{s\}) + b(S^-) = 0$. Thus, zero is the minimal value of the submodular function $\gamma^T + b$. Theorem 2.4 implies that $(x^*)^-(S^-) = 0$, i.e. all components of the vector x^* are non-negative. In particular, the vector $x^*(S^-)$ is in $\mathcal{B}(\gamma^T + b)$ and thus fulfills $x^*(S^-) = \gamma^T(S^-) + b(S^-) = 0$, which directly implies $x^* = \mathbf{0}$. □

Lemma 6.24 implies that if $(\mathcal{N}, b)_{\text{EAT}}$ has a solution, when minimizing the submodular function $\gamma^T + b$ with an algorithm using Cunningham's framework, the zero vector is returned as a convex combination of vertices of $\mathcal{B}(\gamma^T + b)$. Similar to Lemma 4.21, we can again see that

$$\mathcal{B}(\gamma^T) = \mathcal{B}(\gamma^T + b) - b,$$

and that the vertices $v^{\prec}$ and $\overline{v}^{\prec}$ of $\mathcal{B}(\gamma + b)$ and $\mathcal{B}(\gamma^T)$, respectively, are related as follows, $\overline{v}^{\prec} = v^{\prec} - b$. After the submodular function minimization with the framework of Cunningham, we get

$$\mathbf{0} = \lambda_1 v^{\prec_1} + \lambda_2 v^{\prec_2} + \ldots + \lambda_d v^{\prec_d}$$

and thus

$$\begin{aligned} -b &= \lambda_1(v^{\prec_1} - b) + \lambda_2(v^{\prec_2} - b) + \ldots + \lambda_d(v^{\prec_d} - b) \\ &= \lambda_1 \overline{v}^{\prec_1} + \lambda_2 \overline{v}^{\prec_2} + \ldots + \lambda_d \overline{v}^{\prec_d}. \end{aligned}$$

By Lemma 6.22 the lex-max earliest arrival flows $f_1, \ldots, f_d$ in $\mathcal{N}$ with time horizon T with respect to the total orders $\prec_1, \ldots, \prec_d$, respectively, have characteristic vectors $-\overline{v}^{\prec_1}, \ldots, -\overline{v}^{\prec_d}$, respectively. Thus, the flow over time $f \coloneqq \lambda_1 f_1 + \lambda_2 f_2 + \ldots + \lambda_d f_d$ has characteristic vector b. Since f also has pattern $\theta \mapsto o^\theta(\{s\})$ for $\theta \in [0, T)$ by construction, it is an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$. Overall, this shows the correctness of Algorithm 25.

Algorithm 25: An algorithm that checks whether a given tight earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ with minimal feasible time horizon T in a dynamic network $\mathcal{N}$ with only a single source s has a solution and computes the solution in case of existence.

Input : A tight earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ with minimal feasible time horizon $T \geq 0$ in a dynamic network $\mathcal{N} = (D, u, \tau, \{s\}, S^-)$.
Output : A generalized temporally repeated flow solving $(\mathcal{N}, b)_{\text{EAT}}$ in case of existence, INFEASIBLE otherwise.

$\lambda_1, \ldots, \lambda_d, \prec_1, \ldots, \prec_d, v_{\min} \leftarrow \text{SFM}_{\text{Orlin}}(\gamma^T + b)$
if $v_{min} < 0$ **then**
  return INFEASIBLE
else
  for $i = 1, 2, \ldots, d$ **do**
    $f_{\prec_i} \leftarrow$ LEXMAXEAT$(\mathcal{N}, T, \prec_i)$
  end
  return $\lambda_1, \lambda_2, \ldots, \lambda_d, f_{\prec_1}, f_{\prec_2}, \ldots, f_{\prec_d}$
end

Theorem 6.25 (Correctness of Algorithm 25).

Algorithm 25 computes an earliest arrival transshipment solving a tight earliest arrival transshipment $(\mathcal{N}, b)_{\mathrm{EAT}}$ in a dynamic network $\mathcal{N}$ with only a single source and minimal feasible time horizon $T \geq 0$ in case of existence. In particular,the computed earliest arrival transshipment is a generalized temporally repeated flow and the algorithm requires only polynomial space.

With this theorem we are finished with step 3 and step 4 from our outline given an the beginning of this section. It remains to fix the final step: Generalizing our results to general networks.

6.2.4 Tight Problems in General Dynamic Networks.

So far, we only concentrated on tight earliest arrival transshipment problems $(\mathcal{N}, b)_{\mathrm{EAT}}$ in dynamic networks with only a single source, and we derived an existence criterion and a PSPACE algorithm for this special type of dynamic networks. In this paragraph we generalize our previous results to tight earliest arrival transshipment problems in general networks.
Assume that $\mathcal{N}$ is a dynamic network with multiple sources and sinks and b a supply/demand vector on the terminals $S^+ \cup S^-$ such that $(\mathcal{N}, b)_{\mathrm{EAT}}$ is a tight earliest arrival transshipment problem with minimal feasible time horizon T. We now construct two auxiliary earliest arrival transshipment problems. Denote by $\mathcal{N}_s$ the dynamic network in which a super-source s is attached to all sources in S^+ by arcs with zero transit time and infinite capacity. This source s is the new single source of the dynamic network $\mathcal{N}_s$. By b_s we denote the restriction of b to the sinks. The dynamic network $\mathcal{N}_t$ is similarly defined to be the dynamic network obtained by attaching a super-sink t to all sinks in S^- by arcs with zero transit time and infinite capacity, while b_t is the restriction of b to S^+. The following observation is straightforward:

Observation 6.26. *The earliest arrival transshipment problems $(\mathcal{N}_s, b_s)_{\mathrm{EAT}}$ and $(\mathcal{N}_t, b_t)_{\mathrm{EAT}}$ are tight with minimal feasible time horizon T if $(\mathcal{N}, b)_{\mathrm{EAT}}$ is a tight earliest arrival transshipment problem with minimal feasible time horizon T. In particular the earliest arrival pattern for both problems is given by $\theta \mapsto o^\theta(\{s\})$ for all $\theta \in [0, T)$.*

The problem $(\mathcal{N}_t, b_t)_{\mathrm{EAT}}$ always has a solution, which for example can be obtained as a convex combination of lex-max flows over time with pattern $\theta \mapsto o^\theta(\{s\})$ for $\theta \in [0, T)$ (See Theorem 5.3.1). For the problem $(\mathcal{N}_s, b_s)_{\mathrm{EAT}}$ we can check whether a solution exists, and compute a suitable convex combination of lex-max earliest arrival flows in case of existence in PSPACE using Algorithm 25. It turns out that $(\mathcal{N}, b)_{\mathrm{EAT}}$ has a solution if and only if $(\mathcal{N}_s, b_s)_{\mathrm{EAT}}$ has a solution.

Theorem 6.27.

A tight earliest arrival transshipment problem $(\mathcal{N}, b)_{\mathrm{EAT}}$ in a dynamic network with multiple sources and multiple sinks has a solution if and only if $(\mathcal{N}_s, b_s)_{\mathrm{EAT}}$ has a solution.

Proof. Denote by $T \geq 0$ the minimal feasible time horizon of $(\mathcal{N}, b)_{\mathrm{EAT}}$. If $(\mathcal{N}, b)_{\mathrm{EAT}}$ has a solution, then a flow over time f solving this problem can simply be transformed into a flow over time solving $(\mathcal{N}_s, b_s)_{\mathrm{EAT}}$.
Next assume that $(\mathcal{N}_s, b_s)_{\mathrm{EAT}}$ has a solution and denote by f_s an earliest arrival transshipment solving this problem. We know by Corollary 6.5 that the flow over time f_s has pattern $\theta \mapsto o^\theta(S^+)$, for $\theta \in [0, T)$. Similarly, denote by f_t a flow over time solving the problem $(\mathcal{N}_t, b_t)_{\mathrm{EAT}}$ which has the same pattern.
Denote by x_{f_s} the static flow in $\mathcal{N}^T$ corresponding to f_s and by x_{f_t} the static flow in $\mathcal{N}^T$ corresponding to f_t. Both flows are maximum static flows in $\mathcal{N}^T$. That is, by Theorem 2.18 we can construct a static flow x in $\mathcal{N}^T$ which has the same departure pattern as x_{f_t} and the same arrival pattern as x_{f_s}. Denote by f the flow over time corresponding to x. This flow over time clearly fulfills all supplies *and* demands and also has pattern $\theta \mapsto o^\theta(\{s\})$ for all $\theta \in [0, T)$ and thus is an earliest arrival transshipment solving the problem $(\mathcal{N}, b)_{\mathrm{EAT}}$. Here, we consider all the copies of the sources

in S^+ in the time layers to be the sources of $\mathcal{N}^T$, while all the copies of the sinks in S^- are the sinks of $\mathcal{N}^T$. □

The above proof of Theorem 6.21 also yields an algorithm for solving tight earliest arrival transshipment problems in general networks. However, this algorithm works in the time-expanded network and it thus requires a pseudo-polynomial amount of space in the worst case. In the following we will derive a polynomial space algorithm to solve $(\mathcal{N}, b)_{\text{EAT}}$ in general networks. The first step of our algorithm will consist of checking whether $(\mathcal{N}_s, b_s)_{\text{EAT}}$ has a solution, and of determining suitable convex coefficients $\lambda_1, \dots, \lambda_{d_1} \geq 0$ and total orders $\prec_1, \dots, \prec_{d_1}$ on S^- in case of existence by using the submodular function minimization algorithm of Orlin. Note that $d_1 \leq |S^-|$ by Carathéodory's theorem and the fact that the algorithm of Orlin maintains minimal convex combinations throughout the algorithm. Denote by $f_1, \dots, f_{d_1}$ the lex-max earliest arrival flows with time horizon T corresponding to $\prec_1, \dots, \prec_{d_1}$, respectively.
In the second step, provided $(\mathcal{N}_s, b_s)_{\text{EAT}}$ was solvable, we compute the suitable convex combination of lex-max flows over time solving $(\mathcal{N}_t, b_t)_{\text{EAT}}$. Thus, we compute suitable convex coefficients $\mu_1, \dots, \mu_{d_2} \geq 0$ and total orders $\prec'_1, \dots, \prec'_{d_2}$ on S^+. Again, we obtain $d_2 \leq |S^+|$ and we denote by $g_1, \dots, g_{f_2}$ the lex-max flows over time with pattern $\theta \mapsto o^\theta(S^+)$ for $\theta \in [0, T)$ and time horizon T with respect to $\prec'_1, \dots, \prec'_{d_2}$.
In order to solve $(\mathcal{N}, b)_{\text{EAT}}$ we would like to construct flows over time $h_{i,j}$ with pattern $\theta \mapsto o^\theta(\{s\})$ for $\theta \in [0, T)$ and time horizon T for all $i \in \{1, \dots, d_2\}$ and and $j \in \{1, 2, \dots, d_1\}$ with the additional property that $h_{i,j}$ has the arrival pattern of f_j and the departure pattern of g_i on the single sinks and sources, respectively. The flow over time

$$h \coloneqq \sum_{i=1}^{d_2} \sum_{j=1}^{d_1} \lambda_j \mu_i h_{i,j}$$

solves $(\mathcal{N}, b)_{\text{EAT}}$ by construction. Thus, the only thing that remains to be done is to compute $h_{i,j}$ for all possible choices of i and j. We achieve an algorithm for computing such flows by combining the lex-max flow over time algorithm of Hoppe and Tardos with Algorithm 24. The base of our algorithm is the lex-max flow over time algorithm of Hoppe and Tardos for the special case of dynamic networks with only a single sink t and for total orders on the terminals in which the sink t occurs last in the order. The only difference is that instead of doing a simple minimum-cost flow computation in line 8 of the algorithm, we instead use Algorithm 25 to compute our initial minimum-cost flow. This way we make sure that each sink receives the correct amount of flow and the following lex-max flow over time computation ensures that the sources also send the correct amount of flow.

Theorem 6.28 (Correctness of Algorithm 26).
Let $\mathcal{N}$ be a dynamic network with multiple sources and sinks and $T \geq 0$ a time horizon. Further let $\prec$ be a total order on S^- and $\prec'$ a total order on S^+. The flow over time f returned by Algorithm 26 with respect to these parameters is a feasible flow over time with time horizon T and the additional property that f has the departure pattern of a lex-max flow over time with time horizon T with respect to $\prec'$ on the sources and the arrival pattern of a lex-max earliest arrival flow with respect to $\prec$ on the sinks (in both cases the sources are assumed to have higher priority than all the sinks). In particular, f has pattern $\theta \mapsto o^\theta(S^+)$ for $\theta \in [0, T)$.

Proof. That f is a feasible flow over time with time horizon T follows from the correctness of the lex-max flow over time algorithm of Hoppe and Tardos. That f has the correct arrival and departure patterns on the sinks and sources, respectively, also follows directly from the way the lex-max flow over time algorithm of Hoppe and Tardos works. □

Overall, we thus obtain the following algorithm for general tight earliest arrival transshipment problems. The correctness follows immediately from our arguing above.

Algorithm 26: Algorithm for computing combinations of lex-max flows over time on the sources, and lex-max earliest arrival flows on the sinks

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, S^-, S^-)$, a linear order $\prec$ on $S^- = \{t_1, \dots, t_{k_1}\}$ with $t_1 \prec \dots \prec t_{k_1}$, a linear oder $\prec'$ on $S^+ = \{s_1, \dots, s_{k_2}\}$ with $s_1 \prec' \dots \prec' s_{k_2}$ and a time horizon $T \geq 0$

Output : A flow over time f with pattern $\theta \mapsto o^\theta(S^+)$ and departure pattern of a lex-max flow over time with time horizon T with respect to $\prec'$ and arrival pattern of a lex-max earliest arrival flow with respect to $\prec$ (the sources are assumed to have higher priority than all the sinks).

$k \leftarrow d_2 + 1$
$V \leftarrow V \cup \{s\}$
$A^{k+1} \leftarrow A \cup \{(s, s') \mid s' \in S^+\}$
Extent u to A^{k+1} by defining $u_{(s,s')} \coloneqq \infty$ for all $s' \in S^+$
Extent τ to A^{k+1} by defining $\tau_{(s,s')} = 0$ for all $s' \in S^+$
$A^{k+1} \leftarrow A^{k+1} \cup \{(t, s')\}$ with $u_{(t,s')} = \infty$ and $\tau_{(t,s')} = -T$
$\mathcal{N}^{k+1} \leftarrow (V, A^{k+1})$
$y, (\mathcal{P}, w) \leftarrow$ Path decomposition of the static y flow computed in Algorithm 25
$x^{k+1} \leftarrow y$
$X^{k+1} \leftarrow (\mathcal{P}, w)$
for $i \in \{d_2, d_2 - 1, \dots, 1\}$ **do**
    $A^i \leftarrow A^{i+1}$
    $A^i \leftarrow A^i \setminus \{(s, s_i)\}$
    $\mathcal{N}^i \leftarrow (V, A^i)$
    $y^i \leftarrow$ Minimum cost maximum s-s_i flow in $\mathcal{N}^i_{x^{i+1}}$ with τ as costs computed with SSPA
end
$x^i \leftarrow x^{i+1} + y^i$
Compute a path decomposition of the static flow y^i given by $(\mathcal{P}^i, w^i)$
$X^{i+1} \leftarrow X^i \cup (\mathcal{P}^i, w^i)$
return *Generalized temporally repeated flow corresponding to* X^1

Algorithm 27: An algorithm that checks whether a given tight earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ with minimal feasible time horizon T has a solution and computes the solution in case of existence.

Input : A tight earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ with minimal feasible time horizon T in a dynamic network $\mathcal{N} = (D, u, \tau, \{s\}, S^-)$.

Output : A generalized temporally repeated flow solving $(\mathcal{N}, b)_{\text{EAT}}$ in case of existence, INFEASIBLE.

$\lambda_1, \dots, \lambda_{d_1}, \prec_1, \dots, \prec_{d_1}, v_{\min} \leftarrow \text{SFM}_{\text{Orlin}}(\gamma^T + b_s)$
if $v_{min} < 0$ **then**
    return INFEASIBLE
else
    $\mu_1, \dots, \mu_{d_2}, \prec'_1, \dots, \prec'_{dd_2}, v'_{\min} \leftarrow \text{SFM}_{\text{Orlin}}(o^T - b_t)$
    for $i = 1, 2, \dots, d_1$ **do**
        for $j = 1, 2, \dots, d_2$ **do**
            $h_{i,j} \leftarrow$ flow over time computed by Algorithm 26 with respect to $\prec_i$ and $\prec'_j$
            $\lambda_{i,j} \leftarrow \lambda_i \cdot \mu_j$
        end
    end
    return $\lambda_{1,1}, \dots, \lambda_{1,d_2}, \dots, \lambda_{d_1,d_2}, h_{1,1}, \dots, h_{1,d_2}, \dots, h_{d_1,d_2}$
end

Theorem 6.29 (Correctness of Algorithm 27).

Let $(\mathcal{N}, b)_{\text{EAT}}$ be a tight transshipment problem in a dynamic network $\mathcal{N}$. Algorithm 27 returns a transshipment over time solving this problem in case of existence.

Remark about Computing Integral Earliest Arrival Transshipments. During this section we derived an algorithm to compute a fractional solution of a tight earliest arrival transshipment

problem in case of existence. However, by adapting the algorithm of Hoppe and Tardos, we can also compute an integral solution. Recall, that the algorithm of Hoppe and Tardos solves a quickest transshipment problem by reducing it to a lex-max flow over time problem. Using exactly the same construction as in the algorithm of Hoppe and Tardos but using the submodular function γ^T instead of o^T we can reduce a tight earliest arrival transshipment problem to a lex-max earliest arrival flow problem. This way we obtain a second algorithm for computing an earliest arrival transshipment in case of existence that trades an integral solution with a lot more parametric submodular function minimization required to achieve the solution. Since the construction is exactly similar to the one by Hoppe and Tardos, we do not go into more detail here but refer to [HT00].

6.3 The General Case

In this section we generalize our results from the previous section and derive a PSPACE algorithm that checks if a, not necessarily tight, earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network $\mathcal{N}$ with *only a single source* s has a solution and returns a suitable earliest arrival transshipment in case of existence. Our main results in this section are similar to the ones derived previously: We will again show that if $(\mathcal{N}, b)_{\text{EAT}}$ has a solution, then $-b$ lies inside a suitably defined polytope $\mathfrak{P}^{\text{EAT}}_{s,S^-}$. For the other direction, however, an additional condition needs to be fulfilled. We will give a PSPACE algorithm that checks these conditions and thus determines whether a given earliest arrival transshipment problem has a solution. It will again turn out that the vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ correspond to certain flows over time, which are a generalization of lex-max earliest arrival flows called **generalized lex-max earliest arrival flows**. This correspondence leads to our structural main result, which states that in case of existence an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ can be obtained as a convex combination of generalized lex-max earliest arrival flows. We will also provide a PSPACE algorithm to compute a suitable convex combination in case of existence. To obtain the results mentioned above, multiple steps are necessary of which we give an outline in the following:

1. Define a special class of flows over time, called **generalized lex-max earliest arrival flows** (Section 6.3.1).

2. Derive a PSPACE algorithm that checks whether a specific generalized lex-max earliest arrival flow does exist, and computes such a flow in case of existence (Section 6.3.1).

3. Define a polytope $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ as the product of suitably chosen base polytopes corresponding to the earliest arrival pattern of a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network $\mathcal{N}$ with a single source s (Section 6.3.2).

4. Show that the vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ correspond to generalized lex-max earliest arrival flows and that $(\mathcal{N}, b)_{\text{EAT}}$ has a solution if and only if $-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$ *and* all generalized lex-max earliest arrival flows corresponding to the vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ do exist. Give a PSPACE algorithm to check these conditions (Section 6.3.2).

5. Show that a suitable convex combination of generalized-lex max earliest arrival flows solving $(\mathcal{N}, b)_{\text{EAT}}$ can be computed in polynomial space (Section 6.3.2).

We will start by defining generalized lex-max earliest arrival flows and by describing a PSPACE algorithm to compute such flows in case of existence.

6.3.1 Generalized Lexicographically Maximum Earliest Arrival Flows

The class of flows over time we define in this section has a lot of similarities to the class of generalized lex-max flows over time. In particular, these flows also correspond to a partition $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$ of the set of sinks S^- of a given dynamic network $\mathcal{N} = (D = (V, A), u, \tau, s, S^-)$ with only a single source s, rational times $0 < \theta_1 < \theta_2 < \ldots < \theta_r$ and a total order $\prec$ on S^- that **respects the given**

partition of S^-. That is we have $t \prec t'$ for all $t \in T_i$ and $t' \in T_j$ with $i < j$. We start by giving some notation: Define

$$W_i \coloneqq \bigcup_{j=i}^{r} T_i \text{ and } \mathcal{N}_i \coloneqq (D = (V, A), u, \tau, s, W_i) \text{ for all } i \in \{1, \dots, r\}.$$

Moreover, for all $i \in \{1, \dots, r\}$ and $\theta \in [0, \theta_i)$, the set function

$$\gamma_i^\theta \colon 2^{W_i} \to \mathbb{R}$$

is defined by setting $\gamma_i^\theta(X)$ to be the maximal amount of flow that can reach the sinks in X in a flow over time in $\mathcal{N}_i$ with time horizon θ' and pattern $\theta' \mapsto o^\theta(\{s\} \cup S^- \setminus W_i)$ for $\theta' \in [0, \theta_i)$. By Theorem 6.11 we know that γ_i^θ is a submodular function for all $i \in \{1, \dots, r\}$ and all $\theta \in [0, \theta_i)$. A **generalized lex-max earliest arrival flow** with respect to $T_1 \sqcup T_2 \sqcup \dots \sqcup T_r$, the given times $\theta_1, \dots, \theta_r$ and the order $\prec$ can be imagined to be a multiple layer lex-max earliest arrival flow with the property that the sinks in the set T_i are only allowed to receive flow until time θ_i for all $i \in \{1, \dots, r\}$. More precisely, a generalized lex-max earliest arrival flow f with respect to the given parameters has the property that the restriction of f to $\mathcal{N}_i$ behaves like a lex-max earliest arrival flow with respect to the restriction of $\prec$ to W_i with time horizon θ_i. Additionally the sinks in T_i do not receive any flow after time θ_i for all $i \in \{1, \dots, r\}$. This leads to the following formal definition.

Definition 6.30.
*Let $\mathcal{N} = (D = (V, A), u, \tau, s, S^-)$ be a dynamic network with only a single source s, $S^- = T_1 \sqcup T_2 \sqcup \dots \sqcup T_r$, $\theta_0 \coloneqq 0 < \theta_1 < \theta_2 < \dots < \theta_r$ rational times, and $\prec$ a total order on S^- that respects the given partition of S^-. We say that f is a **generalized lex-max earliest arrival flow** with respect to $T_1, \dots, T_r$, $0 < \theta_1, \dots, \theta_r$ and $\prec$ if it fulfills the following properties:*

- *Denote by $\prec_i$ the restriction of $\prec$ to W_i. The flow over time f restricted to $\mathcal{N}_i$ behaves like a lex-max earliest arrival flow with time horizon θ_i for all $i \in \{1, 2, \dots, r\}$.*
- *After time θ_i no flow arrives at the sinks in T_i or is sent towards T_i for all $i \in \{1, 2, \dots, r\}$.*

In contrast to lex-max earliest arrival flows and generalized lex-max flows over time, generalized lex-max earliest arrival flows do *not* always exist (see Figure 6.10 for an example). It will be one main result from this section to derive an existence criterion for generalized lex-max earliest arrival flows. We start by deriving the pattern and characteristic vector of such flows over time, in case of existence.

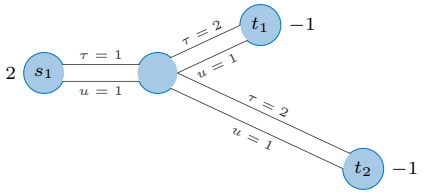


Figure 6.10: Consider the depicted dynamic network with $T_1 = \{t_1\}$, $T_2 = \{t_2\}$, $\theta_1 = 3$ and $\theta_2 = 4$. The generalized lex-max earliest flow with respect to these parameters and $t_1 \prec t_2$ does not exist. The argumentation is the same as in Figure 6.1 where we argued that no earliest arrival transshipment does exist.

Lemma 6.31. *Let $\mathcal{N}$ be a dynamic network with only a single source s, $S^- = T_1 \sqcup T_2 \sqcup \dots \sqcup T_r$, $0 < \theta_1 < \theta_2 < \dots < \theta_r = T$ rational times, and $\prec$ a total order on S^- that respects the given partition of S^-. In case of existence, a generalized lex-max earliest arrival flow in $\mathcal{N}$ with respect to these parameters has pattern*

$$|f|_\theta = o^\theta(\{s\} \cup S^- \setminus W_i) + \sum_{j=1}^{i-1} o^{\theta_j}(\{s\} \cup S^- \setminus W_j) - o^{\theta_j}(\{s\} \cup S^- \setminus W_{j+1}) \text{ for all } \theta \in [\theta_{i-1}, \theta_i).$$

The characteristic vector of f is given as follows: For all $i \in \{1,\dots,r\}$ we obtain for any $t \in T_i$

$$x_f(t) = \gamma_i^T(\{t' \in W_i \mid t \prec t'\}) - \gamma_i^T(\{t' \in W_i \mid t \preceq t'\}).$$

Proof. Let f be a generalized lex-max earliest arrival flow with respect to $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$, $0 = \theta_0 < \theta_1 < \theta_2 < \ldots < \theta_r$ and $\prec$. Fix some $i \in \{1,2,\dots,r\}$. By definition, the restriction of f to $\mathcal{N}_i$ behaves like a lex-max earliest arrival flow with respect to $\prec_i$ and time horizon θ_i for all $i \in \{1,\dots,r\}$. This in particular implies that until time θ_i the sinks in W_i receive as much flow as possible, while no flow arrives at the sinks in T_i after time θ_i. Thus, for all $i \in \{1,\dots,r\}$ we have for any $\theta \in [\theta_{i-1}, \theta_i)$,

$$-\operatorname{net}_f(W_i, \theta) = o^\theta(\{s\} \cup S^- \setminus W_i),$$

while for all $j \in \{1,\dots,i-1\}$ it follows that,

$$-\operatorname{net}_f(T_j, \theta) = -\operatorname{net}_f(T_j, \theta_j) = o^{\theta_j}(\{s\} \cup S^- \setminus W_j) - o^{\theta_j}(\{s\} \cup S^- \setminus W_{j+1}).$$

Putting this together yields for all $i \in \{1.\dots,r\}$ and all $\theta \in [\theta_{i-1}, \theta_i)$

$$|f|_\theta = o^\theta(\{s\} \cup W_i) + \sum_{j=1}^{i-1} o^{\theta_j}(\{s\} \cup S^- \setminus W_j) - o^{\theta_j}(\{s\} \cup S^- \setminus W_{j+1}).$$

The characteristic vector can easily be deduced from the characteristic vector for lex-max earliest arrival flows (see Lemma 6.18): Fix some $i \in \{1,\dots,r\}$ and some $t \in T_i$. By the definition of a generalized lex-max earliest arrival flow we know that the sinks in T_i do only receive flow until time θ_i and that f restricted to $\mathcal{N}_i$ behaves like a lex-max earliest arrival flow with time horizon θ_i with respect to the restriction of $\prec$ to W_i. Thus,

$$x_f(t) = \operatorname{net}_f(t,T) \quad = \quad \operatorname{net}_f(t,\theta_i) \overset{\text{Lem. } 6.18}{=} \gamma_i^\theta(\{t' \in W_i \mid t \prec t'\}) - \gamma_i^\theta(\{t' \in W_i \mid t \preceq t'\}).$$

□

It turns out that a generalized lex-max earliest arrival flow that is computed with respect to the pattern of a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a single source s fulfills the earliest arrival pattern p^* of $(\mathcal{N}, b)_{\text{EAT}}$.

Observation 6.32. *Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single source s. Assume that $T_1 \sqcup \ldots \sqcup T_r = S^-$ and $0 = \theta_0 < \theta_1 < \ldots < \theta_r = T$ are the sets and times corresponding to the earliest arrival pattern p^* of $(\mathcal{N}, b)_{\text{EAT}}$ as computed by Algorithm 23. Let $\prec$ be an arbitrary total order on S^- that respects the given partition. In case of existence, a generalized lex-max earliest arrival flow f with respect to these parameters has pattern p^*.*

Proof. Recall that the earliest arrival pattern p^* is given as follows: For all $i \in \{1,\dots,r\}$ and $\theta \in [\theta_{i-1}, \theta_i)$ we have

$$p^*(\theta) \overset{\text{Thm. } 6.7}{=} o^\theta(\{s\} \cup S^- \setminus W_i) - b(S^- \setminus W_i),$$

and for all $j \in \{1,\dots,i-1\}$ it also holds that

$$-b(T_j) \overset{\text{Thm. } 6.7}{=} o^{\theta_j}(\{s\} \cup S^- \setminus W_j) - o^{\theta_j}(\{s\} \cup S^- \setminus W_{j+1}).$$

Plugging this into the pattern of f according to Lemma 6.31 yields the statement of this observation. □

The above observation explains the name *generalized lex-max earliest arrival flow*. The overall goal of this section is to derive a PSPACE algorithm to check whether a specific generalized lex-max earliest arrival flow does exist and to compute such a flow over time in case of existence. In the next section

we will then use generalized lex-max earliest arrival flows corresponding to the earliest arrival pattern of a given earliest arrival transshipment problem to solve earliest arrival transshipment problems. In order to achieve such results, it turns out to be useful to derive a correspondence between lex-max earliest arrival flows and special static lex-max flows in the time-expanded network. Let $\mathcal{N}$ be a dynamic network with only a single source s, $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$, $\theta_1 < \theta_2 < \ldots < \theta_r = T$ and $\prec$ a total order on S^- that respects the given partition. Assume for the moment that $\theta_1, \ldots, \theta_r$ are integral. The time-expanded network, as described in Section 2.5.2, is denoted by $\mathcal{N}^T$. We are again interested in a variant of the time-expanded network. The only thing that we change this time is the set of sinks of $\mathcal{N}^T$. We are not interested in just considering the overall super-sink t^*, but we want to consider the copies t^θ of sinks $t \in S^-$ for each $\theta \in \{0, 1, \ldots, T\}$. However, we do not want to consider all of them. The copies we are interested in are connected with the properties of generalized lex-max earliest arrival flows. By definition, for each $i \in \{1, \ldots, r\}$, the sinks in T_i are only allowed to receive flow until time θ_i in a generalized lex-max earliest arrival flow. Thus, in our variant of the time-expanded network we are only interested in copies t^θ of $t \in T_i$ up to time layer θ_i. For each $i \in \{1, \ldots, r\}$ we denote the copies of the sinks in T_i on time layer $\theta \in \{1, \ldots, \theta_i\}$ by T_i^θ. The overall set of sinks in the time-expanded network that we are interested is thus given by

$$\overline{T} := \bigcup_{i=1}^{r} \bigcup_{j=1}^{\theta_i} T_i^j.$$

Denote by $\overline{\mathcal{N}}^T$ the time-expanded network $\mathcal{N}^T$ with set of sinks $\overline{T}$ and the unique overall super-source s^*. We aim for deriving a correspondence between generalized lex-max earliest arrival flows and static lex-max flows in the time-expanded network $\overline{\mathcal{N}}^T$ with the set of sinks $\overline{T}$. The first step is to define a total order $\prec_T$ on the terminals of $\overline{\mathcal{N}}^T$ that corresponds to the given order $\prec$ on S^-. In the order $\prec_T$ the source s^* will appear first, and sinks on a higher time layer are ordered before sinks on earlier time layers. Within a specific time layer the copies of sinks are ordered according to $\prec$. To formalize this, assume that $S^- = \{t_1, t_2, \ldots, t_k\}$ with $T_i = \{t_{|S^- \setminus W_i|+1}, \ldots, t_{|S^- \setminus W_{i+1}|}\}$ for all $i \in \{1, \ldots, r\}$ and that the indices are chosen to respect $\prec$, that is $t_{|S^- \setminus W_i|+1} \prec \ldots \prec t_{|S^- \setminus W_{i+1}|}$. The total order $\prec_T$ on $\overline{T}$ is now defined as follows

$$\begin{aligned}
s^* \prec_T\ & t^{\theta_r}_{|S^- \setminus W_r|+1} \prec_T t^{\theta_r}_{|S^- \setminus W_r|+2} \prec_T \ldots \prec_T t^{\theta_r}_k \\
\prec_T\ & t^{\theta_{r-1}}_{|S^- \setminus W_r|+1} \prec_T t^{\theta_{r-1}}_{|S^- \setminus W_r|+2} \prec_T \ldots \prec_T t^{\theta_{r-1}}_k \\
\prec_T\ & \ldots \prec_T \\
\prec_T\ & t^{\theta_{r-1}+1}_{|S^- \setminus W_r|+1} \prec_T t^{\theta_{r-1}+1}_{|S^- \setminus W_r|+2} \prec_T \ldots \prec_T t^{\theta_{r-1}+1}_k \\
\prec_T\ & \ldots \prec_T \\
\prec_T\ & t^{\theta_2}_{|S^- \setminus W_2|+1} \prec_T t^{\theta_2}_{|S^- \setminus W_2|+2} \prec_T \ldots \prec_T t^{\theta_2}_k \\
\prec_T\ & t^{\theta_2-1}_{|S^- \setminus W_2|+1} \prec_T t^{\theta_2-1}_{|S^- \setminus W_2|+2} \prec_T \ldots \prec_T t^{\theta_2-1}_k \\
\prec_T\ & \ldots \prec_T \\
\prec_T\ & t^{\theta_1+1}_{|S^- \setminus W_2|+1} \prec_T \ldots \prec_T t^{\theta_1+1}_k \\
\prec_T\ & t^{\theta_1}_{|S^- \setminus W_1|+1} \prec_T t^{\theta_1}_{|S^- \setminus W_1|+2} \prec_T \ldots \prec_T t^{\theta_1}_k \\
\prec_T\ & t^{\theta_1-1}_{|S^- \setminus W_1|+1} \prec_T t^{\theta_1-1}_{|S^- \setminus W_1|+2} \prec_T \ldots \prec_T t^{\theta_1-1}_k \\
\prec_T\ & \ldots \prec_T \\
\prec_T\ & t^{1}_{|S^- \setminus W_1|+1} \prec_T t^{1}_{|S^- \setminus W_1|+2} \prec_T \ldots \prec_T t^{1}_k.
\end{aligned}$$

As an example consider the instance of a generalized lex-max earliest arrival transshipment problem depicted in Figure 6.11. For this example, we obtain $\overline{T} = T_1^1 \cup T_2^1 \cup T_2^2 \cup T_2^3 \cup T_2^4$ with $T_1^1 = \{t_1^1\}$ and $T_2^i = \{t_2^i, t_3^i\}$ for $i = 1, 2, 3, 4$. The total order $\prec_T$ is given by

$$s^* \prec_T t_2^4 \prec_T t_3^4 \prec_T t_2^3 \prec_T t_3^3 \prec_T t_2^2 \prec_T t_3^2 \prec_T t_1^1 \prec_T t_2^1 \prec_T t_3^1.$$

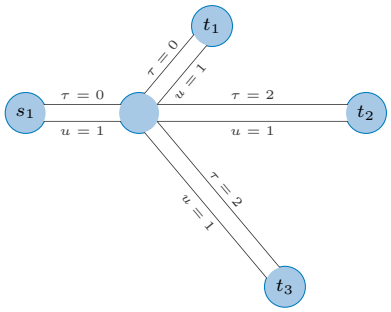


Figure 6.11: A dynamic network $\mathcal{N}$ with $T_1 = \{t_1\}, T_2 = \{t_2, t_3\}, \theta_1 = 1$ and $\theta_2 = 4$. Moreover, we have $t_1 \prec t_2 \prec t_3$.

The corresponding time-expanded network, together with the static lex-max flow in $\overline{\mathcal{N}}^4$ with respect to $\prec_T$ is visualized in Figure 6.12. For fractional values of $\theta_1, \dots, \theta_r$ works completely similar for a suitably chosen discretization of time.

Lemma 6.33. *Let $\mathcal{N} = (D = (V, A), u, \tau, s, S^-)$ be a dynamic network with only a single source s. A generalized lex-max earliest arrival flow f with respect to $S^- = T_1 \sqcup T_2 \sqcup \dots \sqcup T_r$, rational $0 < \theta_1 < \theta_2 < \dots < \theta_r = T$ and a total order $\prec$ on S^- that respects the given partition induces a static lex-max flow x_f with respect to $\prec_r$ in $\overline{\mathcal{N}}^T$.*
On the other hand, a static lex-max flow x with respect to $\prec_r$ in $\overline{\mathcal{N}}^T$ induces a generalized lex-max earliest arrival flow with respect to the given parameters if such a flow exists.

Proof. For simplicity we prove the lemma only for integral $\theta_1, \dots, \theta_r$. The proof for rational values works completely similar. We start by showing that the generalized lex-max earliest arrival flow f induces a static lex-max flow with respect to $\prec_r$ in the time-expanded network $\overline{\mathcal{N}}^T$ as above. By the definition of a generalized lex-max earliest arrival flow, the flow over time f has the property that for all $i \in \{1, \dots, r\}$ until time θ_i the flow over time f restricted to $\mathcal{N}_i$ behaves like a lex-max earliest arrival flow in $\mathcal{N}_i$ with respect to the restriction of $\prec$ to W_i denoted by $\prec_i$. That is, we have for all $\in \{1, \dots, r\}$

$$-\operatorname{net}_f(\{t' \in W_i \mid t \preceq t'\}) = \gamma_i^{\theta_i}(\{t' \in W_i \mid t \preceq t'\}) \text{ for all } t \in W_i.$$

Thus, for each $i \in \{1, \dots, r\}$ we can apply Lemma 6.12 which overall yields that f induces a static lex-max flow in $\overline{\mathcal{N}}_i^{\theta_i}$ with respect to the order $\prec_T^i$ given by

$$\begin{aligned} s^* \prec_T^i t^{\theta_i}_{|S^- \setminus W_i|+1} \prec_T^i t^{\theta_i}_{|S^- \setminus W_i|+2} \prec_T^i \dots \prec_T^i t^{\theta_i}_k \\ \prec_T^i t^{\theta_i - 1}_{|S^- \setminus W_i|+1} \prec_T^i t^{\theta_i - 1}_{|S^- \setminus W_i|+2} \prec_T^i \dots \prec_T^i t^{\theta_i - 1}_k \\ \prec_T^i \dots \prec_T^i \\ \prec_T^i t^1_{|S^- \setminus W_i|+1} \prec_T^i t^1_{|S^- \setminus W_i|+2} \prec_T^i \dots \prec_T^i t^1_k. \end{aligned}$$

Thus, the static flow over time x in $\overline{\mathcal{N}}^T$ induced by f fulfills all the properties of static lex-max flows with respect to the orders $\prec_T^i$ in $\overline{\mathcal{N}}^{\theta_i}$ for all $i \in \{1, \dots, r\}$. We can now use this fact to show that x is in fact a static lex-max flow with respect to the order $\prec_T$. In order to prove that x is such a lex-max flow, we need to show that for all $t \in \overline{T}$ we have

$$-\operatorname{net}_x(\{t' \in \overline{T} \mid t \prec_T t'\}) = \max_{\overline{\mathcal{N}}^T}(s^*, \{t' \in \overline{T} \mid t \prec_T t'\}). \tag{6.14}$$

We show this statement by induction. We start with $i = 1$. Let $\theta \in \{1, \dots, \theta_1\}$ and $t_j^\theta \in T_p^\theta$ for some $p \in \{1, \dots r\}$ and $j \in \{|S^- \setminus W_p| + 1, \dots, |S^- \setminus W_{p+1}|\}$. The set $\{t' \in \overline{T} \mid t_j^\theta \prec_T t'\}$ consists of all the terminals on lower time layers than θ and all the terminals from layer θ that are ordered behind t_j^θ. In particular, all terminals from this set are terminals in $\mathcal{N}_1^{\theta_1}$. By assumption, x is a

static lex-max flow in $\mathcal{N}_1^{\theta_1}$ with respect to $\prec_T^1$ and the terminals in $\{t' \in \overline{T} \mid t_j^\theta \prec_T t'\}$ form an upper ideal in this order, which implies that

$$-\operatorname{net}_x(\{t' \in \overline{T} \mid t_j^\theta \prec_T t'\}) = \max_{\overline{\mathcal{N}}^T}(s^*, \{t' \in \overline{T} \mid t_j^\theta \prec_T t'\}).$$

We now proceed by induction and assume that the statement was shown for $1 \leq i < r$. Let $\theta \in \{\theta_i+1, \dots, \theta_{i+1}\}$ and $t_j^\theta \in T_p^\theta$ for some $p \in \{i+1, \dots r\}$ and $j \in \{|S^- \setminus W_p|+1, \dots, |S^- \setminus W_{p+1}|\}$. We again investigate how the set $\{t' \in \overline{T} \mid t_j^\theta \prec_T t'\}$ looks like. It consists of all the terminals on layers below θ and of the terminals in layer θ that are ordered behind t_j^θ. That is

$$\{t' \in T' \mid t_j^\theta \prec_T t'\} = \underbrace{\bigcup_{l=1}^{r} \bigcup_{m=1}^{\min\{\theta_l,\theta_i\}} T_l^m}_{\text{sinks up to layer } \theta_i} \cup \underbrace{\bigcup_{l=i+1}^{r} \bigcup_{m=\theta_i+1}^{\theta-1} T_l^m}_{\text{sinks from layer } \theta_{i+1} \text{ to } \theta-1} \cup \{t_{j+1}^\theta, \dots, t_k^\theta\}.$$

By induction, we have

$$-\operatorname{net}_x\left(\bigcup_{l=1}^{r} \bigcup_{m=1}^{\min\{\theta_l,\theta_i\}} T_l^m\right) = \max_{\overline{\mathcal{N}}^T}\left(s^*, \bigcup_{l=1}^{r} \bigcup_{m=1}^{\min\{\theta_j,\theta_i\}} T_l^m\right).$$

Additionally, the sinks in

$$\bigcup_{l=i+1}^{r} \bigcup_{m=\theta_i+1}^{\theta-1} T_l^m \cup \{t_j^\theta, t_{j+1}^\theta, \dots, t_k^\theta\} \tag{6.15}$$

are sinks in $\mathcal{N}_{i+1}^{\theta_{i+1}}$ and we know that x is a static lex-max flow in this network with respect to the order $\prec_T^{i+1}$. In the overall order $\prec_r$ more sinks are ordered before the sinks in the set in (6.15) than in $\prec_T^{i+1}$. That is in a lex-max flow with respect to $\prec_T^{i+1}$ at least as much flow is sent towards these sinks as in a lex-max flow with respect to order $\prec_r$. This overall implies

$$-\operatorname{net}_x(\{t' \in \overline{T} \mid t_j^\theta \prec_T t'\} \cup \{t_j^\theta\}) = \max_{\overline{\mathcal{N}}^T}(s^*, \{t' \in \overline{T} \mid t_j^\theta \prec_T t'\} \cup \{t_j^\theta\}). \tag{6.16}$$

Overall, this yields that x is a static lex-max flow with respect to the order $\prec_T$.
Assume that a generalized lex-max flow over time f with respect to the given parameters does exist. We want to show that in this case, a static lex-max flow x in $\overline{\mathcal{N}}^T$ with respect to the order $\prec_T$ induces a generalized lex-max earliest arrival flow f_x. Denote by x_f the induced static flow in $\overline{\mathcal{N}}^T$ which is by the first part of the lemma a static lex-max flow with respect to the order $\prec_T$. However, as f is a generalized lex-max earliest arrival flow, we know that the restriction of f to $\mathcal{N}_i$ can be regarded as a lex-max earliest arrival flow with time horizon θ_i with respect to the total order $\prec_i$. This implies that the restriction of x_f to $\mathcal{N}_i^{\theta_i}$ is a lex-max flow with respect to the total order $\prec_T^i$ defined as above for all $i \in \{1, \dots, r\}$. This property thus also holds for the static lex-max flow x which by Lemma 6.14 implies that the flow over time induced by x needs to be a generalized lex-max earliest arrival flow. □

Lemma 6.34. *A generalized lex-max earliest arrival flow with respect to $S^- = T_1 \sqcup T_2 \sqcup \dots \sqcup T_r$, $0 < \theta_1 < \theta_2 < \dots < \theta_r = T$ and a total order $\prec$ on S^- that respects the given partition exists if and only if the static lex-max flow x in $\overline{\mathcal{N}}^T$ with respect to $\prec_T$ fulfills*

$$-\operatorname{net}_x\left(\bigcup_{j=1}^{\theta_i}\{T_i^j \cup T_{i+1}^j \cup \dots \cup T_r^j\}\right) = o^{\theta_i}(\{s\} \cup S^- \setminus W_i) \text{ for all } i \in \{1, \dots, r\}, \tag{6.17}$$

if $\theta_1, \dots, \theta_r$ are integral. The similar statement holds for rational values.

Proof. We do the proof only for integer valued $\theta_1, \dots, \theta_r$. The proof for rational values works completely similar. First, we will show that, if a static lex-max flow x with respect to the order $\prec_T$ fulfills the above equation, this implies that x restricted to $\mathcal{N}_i^{\theta_i}$ is a static lex-max flow with

respect to the order $\prec_T^i$ for all $i \in \{1, \dots, r\}$. In order to see this, note that the order $\prec_r$ has the property that terminals on lower layers are ordered after nodes on higher layers. In the specific layers the nodes are ordered according to $\prec$. This means that the amount of flow that the copies of the terminals in W_i in layer $\theta \in \{1, \dots, \theta_i\}$ can receive is independent of the amount of flow that that the copies of terminals in W_i received in previous layers. Thus, the fact that we have

$$-\operatorname{net}_x\left(\bigcup_{j=1}^{\theta_i}\{T_i^j \cup T_{i+1}^j \cup \ldots \cup T_r^j\}\right) = o^{\theta_i}(\{s\} \cup S^- \setminus W_i)$$

implies that it also needs to hold

$$-\operatorname{net}_x\left(\bigcup_{j=1}^{\theta}\{T_i^j \cup T_{i+1}^j \cup \ldots \cup T_r^j\}\right) = o^{\theta}(\{s\} \cup S^- \setminus W_i) \text{ for all } \theta \in \{1, \dots, \theta_i\}.$$

Since in each layer θ the copies of terminals corresponding to nodes in W_i are ordered by $\prec_T$ as by $\prec_T^i$, we get that x is a lex-max flow in $\mathcal{N}_i^{\theta_i}$ with respect to order $\prec_T^i$.

Thus, we can apply Lemma 6.14 overall r times and get that f_x restricted to $\mathcal{N}_i$ is a lex-max earliest arrival flow on $\mathcal{N}_i$ with time horizon θ_i with respect to the order $\prec_i$. Also by construction, the sinks in T_i do not receive any flow after time θ_i. Thus, f_x is a generalized lex-max earliest arrival flow with respect to the given parameters.
For the converse direction, assume that f is a lex-max earliest arrival flow with respect to the given parameters. Denote by x_f the static flow in $\overline{\mathcal{N}}^T$ induced by f. By the properties of a generalized lex-max earliest arrival flow, the restriction of x_f to $\mathcal{N}_i^{\theta_i}$ is a static lex-max flow with respect to the order $\prec_T^i$, which implies the statement of the lemma (see Lemma 6.33). □

For an illustration of Lemma 6.34 consider again the generalized lex-max earliest arrival flow problem shown in Figure 6.11. Consider the lex-max flow x in $\overline{\mathcal{N}}^4$ with respect to the order $\prec_T$ on $\overline{T}$ with

$$s^* \prec_T t_2^4 \prec_T t_3^4 \prec_T t_2^3 \prec_T t_3^3 \prec_T t_2^2 \prec_T t_3^2 \prec_T t_1^1 \prec_T t_2^1 \prec_T t_3^1.$$

See Figure 6.12 for an illustration of this static flow. The flow x clearly fulfills

$$-\operatorname{net}_x(T_1^1 \cup T_2^1) = -\operatorname{net}_x(\{t_1^1, t_2^1, t_3^1\}) = 1 = o^1(\{s\}) = o^1(\{s\} \cup S^- \setminus W_1),$$

and thus, equation (6.17) is fulfilled for $i = 1$. However, for $i = 2$, we get

$$-\operatorname{net}_x(T_2^1 \cup T_2^2 \cup T_2^3 \cup T_2^4) = 1 < o^4(\{s\} \cup \{t_1\}) = 2.$$

Thus, Lemma 6.34 implies that the generalized lex-max earliest arrival flow solving the problem from Figure 6.11 does not exist. That this is true can be seen immediately from the structure of the network. With the two lemmas above, we have also derived an algorithm to check whether a specific generalized lex-max flow over time does exist, and to compute such a flow in case of existence. However, the algorithm relies on computing static lex-max flows in the time-expanded network. The correspondence we derived above will be useful for proving the correctness of the PSPACE algorithm for computing generalized lex-max earliest arrival flows that we describe next (see Algorithm 28).
We start by giving the intuition behind the algorithm. Let $\mathcal{N}$ be some dynamic network with only a single source, $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$, $0 < \theta_1 < \theta_2 < \ldots < \theta_r$ and $\prec$ some total order in S^- that respects the given partition of S^-. By definition, a generalized earliest arrival flow with respect to these parameters behaves like a lex-max earliest arrival flow with time horizon θ_i in $\mathcal{N}_i$ for all $i \in \{1, \dots, r\}$. This is why the first iteration of our algorithm is essentially identical to Algorithm 25 for computing lex-max earliest arrival flows. After time θ_1 the sinks in T_1 are not supposed to receive any more flow. Thus, in the second iteration of our algorithms the sinks in T_1 are disregarded and only the remaining sinks receive more flow. In order to make sure that the sinks in W_2 receive as much flow as possible, we send flow towards them again as in Algorithm 25. To make sure that also the paths that were previously occupied by flow towards T_1 can be used to send flow towards the

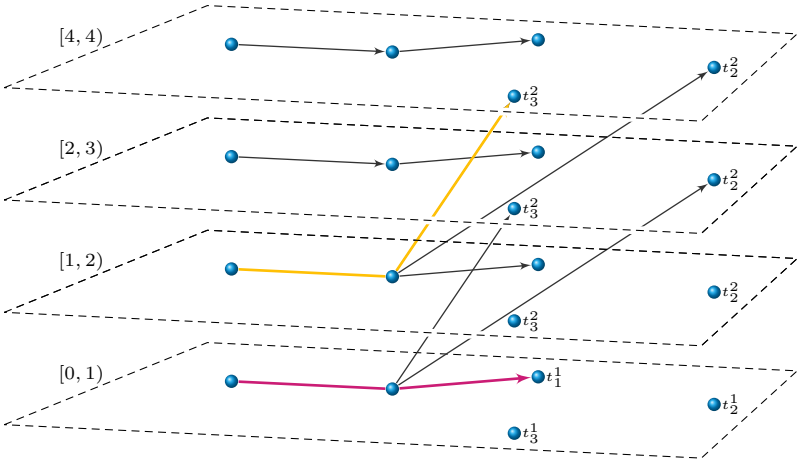


Figure 6.12: The time-expanded network $\overline{\mathcal{N}}^4$ corresponding to the generalized lex-max earliest arrival flow problem depicted in Figure 6.11. The static lex-max flow with respect to $\prec_T$ is indicated.

remaining sinks, we add arcs from the sinks in T_1 to the source with infinite capacity and transit time $-\theta_1$ at the beginning of the second iteration. In the following iterations we proceed analogously. The minimum-cost flow computation that we do in the last step of the algorithm is required to make sure that the time horizons of all sets T_i are respected.

Theorem 6.35 (Correctness of Algorithm 28).

Let $\mathcal{N}$ be a dynamic network with only a single source s, $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$ and $0 < \theta_1 < \theta_2 < \ldots < \theta_r = T$ and $\prec$ some total order on S^- that respects the given partition. The flow over time f returned by Algorithm 28 with respect to these parameters is a generalized lex-max earliest arrival flow with respect to the given parameters in case of existence.

Proof. The constructed flow over time f is a feasible flow over time, because we always augment flow along shortest paths, and because the changes on the considered network $\mathcal{N}_x$ we do during the course of the algorithm do not decrease any distances from the super-source s' to any node in the dynamic network.
We need to show that the static flow x_f induced by the flow over time f is a static lex-max flow with respect to the order $\prec_T$. The proof of this fact is similar to the proof of Theorem 6.19. Essentially, Algorithm 28 iterates for all $l \in \{1, \ldots, T-1\}$ over all sinks that are supposed to receive flow after time l and augments flow towards these sinks along shortest path of length l within the given order. Assume that $l \in \{\theta_{j-1}, \ldots, \theta_j - 1\}$. For each such l and each $i \in \{|S^- \setminus W_j| + 1, \ldots, k\}$ we construct a slight adaption $\mathcal{N}_{l,i}$ of the dynamic network $\mathcal{N}$ as follows: We add a super-sink t to the network, that we attach to the sinks by,

- an arc of transit time $l - \theta_p$ and infinite capacity to each of the sinks in T_p with $p < j$,
- arcs with transit time 0 and infinite capacity to $t_i, \ldots, t_k$,
- arcs with transit time 1 and infinite capacity to $t_{|S^- \setminus W_j|+1}, \ldots, t_{i-1}$.

Further we add an arc (t, s) with infinite capacity and transit time $-l$ to the network. We denote by $x_{l,i}$ the static flow obtained after iteration i for the path length l. We can consider the static flow $x_{l,i}$ also as static flow in $\mathcal{N}_{l,i}$. The first observation is again that this flow is a minimum-cost circulation in $\mathcal{N}_{l,i}$. This is true, because we always augment along shortest path. The fact that we have not augmented along path of length l towards $t_{|S^- \setminus W_j|+1}, \ldots, t_{i-1}$ is compensated by the fact that we connected those sinks with t by arcs with transit time 1. That the sinks in T_p with $p < i$ are only considered for path length up to θ_p is similarly compensated by attaching the sinks in T_p to t by arcs with transit time $l - \theta_p$. Thus, by Fact 2.21 a (generalized) temporally repeated flow corresponding to $x_{l,i}$ induces a maximum flow over time $f_{l,i}$ ins $\mathcal{N}_{l,i}$ with time horizon l. Regarded as a dynamic flow in $\mathcal{N}$ with time horizon l, this flow has the property that flow arrives at the

Algorithm 28: Computation of generalized lex-max earliest arrival flows

Input : A dynamic network $\mathcal{N} = (D = (V, A), u, \tau, s, S^-)$, $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$, $0 < \theta_1 < \theta_2 < \ldots < \theta_r$ and a total order $\prec$ on S^- that respects the given partition. Let $S^- = \{t_1, t_2, \ldots, t_k\}$ with $T_i = \{t_{|S^- \setminus W_i|+1}, \ldots, t_{|S^- \setminus W_{i+1}|}\}$ for all $i \in \{1, \ldots, r\}$ such that the indices are chosen to respect $\prec$, that is $t_{|S^- \setminus W_i|+1} \prec \ldots \prec t_{|S^- \setminus W_{i+1}|}$.

Output : A generalized lex-max earliest arrival flow with respect to the given parameters in case of existence

$V \leftarrow V \cup \{s'\}$
$A \leftarrow A \cup \{(s, s')\}$ with $u_{(s',s)} = \infty$ and $\tau_{(s',s)} = 0$
$x_P \leftarrow 0$ for all $P \in \overleftrightarrow{\mathcal{P}}$
$x \leftarrow$ static s'-S^- flow with generalized path decomposition $(x_P)_{P \in \overleftrightarrow{\mathcal{P}}}$
for $j \in \{1, 2, \ldots, r\}$ **do**
  while $d(\mathcal{N}_x, s', S^-) < \theta_j$ **do**
    $l \leftarrow$ length of a shortest s'-S^- path in $\mathcal{N}_x$
    for $i \in \{k, k-1, \ldots, \sum_{n=1}^{j-1} |T_n| + 1\}$ **do**
      while $d(\mathcal{N}_x, s', t_i) = l$ **do**
        $P \leftarrow$ shortest s'-t_i path in $\mathcal{N}_x$
        $\gamma \leftarrow \min\{u(a) \mid a \in P\}$
        augment x along P by γ
      end
    end
  end
  $A \leftarrow A \cup \{(t, s') \mid t \in T_j\}$ with $u_{(t,s')} = \infty$ and $\tau_{(t,s')} = -\theta_j$ for all $t \in T_j$
  Extent the static flow x to the resulting network
end
$A \leftarrow A \setminus \{(s, s')\}$
$(y_P)_{P \in \overleftrightarrow{\mathcal{P}}} \leftarrow$ minimum-cost flow in $\mathcal{N}_x$ from s' to S^-
return *Generalized temporally repeated flow f corresponding to x and y*

sinks in T_p for $p < j$ only until time θ_p while flow arrives at the sinks $t_{|S^- \setminus W_j|+1}, \ldots, t_{i-1}$ only until time $l - 1$. Moreover, $f_{l,i}$ is a flow over time with maximal value such that these time horizons are respected. Since during the course of the algorithm the amount of flow that arrives at each of these sinks until the respective points in time is not changed, we obtain

$$-\operatorname{net}_{x_f}(\{t' \in \overline{T} \mid t_i^l \prec_T t'\}) = \max_{\overline{N}^T}(s^*, \{t' \in \overline{T} \mid t_i^l \prec_T t'\}).$$

Thus, x_f is a static lex-max flow in $\overline{\mathcal{N}}^T$ with respect to $\prec_T$. By Lemma 6.33 the flow over time f' induced by x_f is a generalized lex-max earliest arrival flow with respect to the given parameters in case of existence. Since the amount of flow that has arrived at each of the sinks is the same in f' as in f for each point in time, this implies that f is also a generalized lex-max earliest arrival flow. □

Using Algorithm 28 we can thus compute a flow over time f with respect to $S^- = T_1 \sqcup \ldots \sqcup T_r$, $0 < \theta_1 < \ldots < \theta_r$ and $\prec$ some total order on S^- that respects the given partition. The flow over time f is a generalized lex-max earliest arrival transshipment with respect to these parameters if such a flow exists. In order to check this on the fly during the computation, we just have to check by Lemma 6.34 whether

$$-\operatorname{net}_f\left(\bigcup_{j=1}^{\theta_i} W_i\right) = o^{\theta_i}(\{s\} \cup S^- \setminus W_i) \text{ for all } i \in \{1, \ldots, r\},$$

and this can be done during the course of the algorithm in polynomial space.

Observation 6.36. *Using Algorithm 28, we can check in polynomial space whether a generalized lex-max earliest arrival flow corresponding to some given parameters does exist.*

The resulting flow, when we apply Algorithm 28 to the problem depicted in Figure 6.11, can be seen in Figure 6.13. We conclude this section with the following technical corollary.

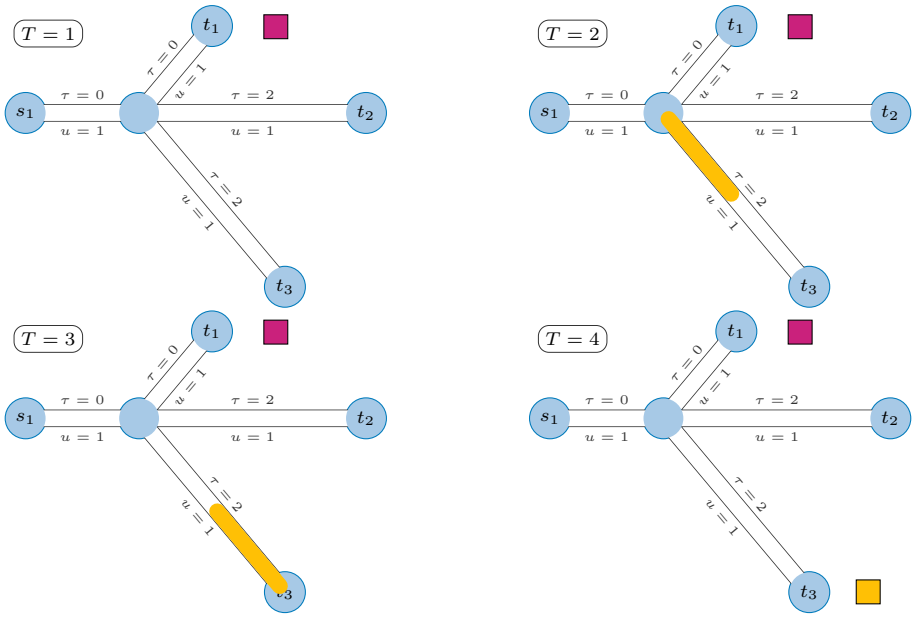


Figure 6.13: The result of Algorithm 24 when applying it to the problem from Figure 6.11. That the pictured flow over time f is not a generalized lex-max earliest arrival flow can be deduced by the above observation through the fact that $1 = -\operatorname{net}_f(\{t_2, t_3\}, 4) < o^4(\{s, t_1\}) = 2$.

Corollary 6.37. *Let $\mathcal{N} = (D = (V, A), u, \tau, s, S^-)$ be a dynamic network with only a single source s, $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$ and $0 < \theta_1 < \theta_2 < \ldots < \theta_r = T$. If for some $\prec$ on S^- that respects the given partition, the corresponding generalized lex-max earliest arrival flow does exist, then it exists for all total orders respecting the given partition.*

Proof. Assume f is a generalized lex-max earliest arrival flow with respect to some order $\prec$ on S^- that respects the given partition. By Lemma 6.34 this implies that the static lex-max flow x with respect to $\prec_T$ fulfills the equations in (6.17). However, that these equations are fulfilled is independent of the specific choice of a total order respecting the partition. □

6.3.2 An Existence Criterion for General Earliest Arrival Transshipments and a PSPACE Algorithm

Now that we have derived how to compute generalized lex-max earliest arrival flow in case of existence, we are ready to use them for our algorithm to check whether a given earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ has a solution. For this purpose, we assume in the following that $(\mathcal{N}, b)_{\text{EAT}}$ is an earliest arrival transshipment problem in a dynamic network with only a single source and an earliest arrival pattern characterized by $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$ and times $0 < \theta_1 < \theta_2 < \ldots < \theta_r$. To derive our existence criterion for an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$, we need a few definitions. For all $i \in \{1, 2, \ldots, r\}$, the set function $g_i \colon 2^{T_i} \to \mathbb{R}$ is defined by

$$g_i(T) \coloneqq \gamma_i^{\theta_i}(W_{i+1} \cup T) - \gamma_i^{\theta_i}(W_{i+1}) \text{ for all } T \subseteq T_i.$$

Thus, we can define a polytope $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ by

$$\mathfrak{P}^{\text{EAT}}_{s,S^-} \coloneqq \bigtimes_{i=1}^{r} \mathcal{B}(g_i).$$

We start by deriving a necessary condition for the existence of an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$.

Lemma 6.38. *Let $(\mathcal{N},b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single source s and earliest arrival pattern characterized by $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$ and times $0 < \theta_1 < \theta_2 < \ldots < \theta_r$. If there exists an earliest arrival transshipment solving $(\mathcal{N},b)_{\text{EAT}}$, then we have*

$$-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}.$$

To prove Lemma 6.38, we need the following technical lemma.

Lemma 6.39. *Let $(\mathcal{N},b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single source s and an earliest arrival pattern characterized by $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$ and times $0 < \theta_1 < \theta_2 < \ldots < \theta_r$. If there exist an earliest arrival transshipment f solving $(\mathcal{N},b)_{\text{EAT}}$, then f restricted to $\mathcal{N}_i$ has pattern $\theta \mapsto o^\theta(S^- \setminus W_i)$ for $\theta \in [0,\theta_i]$ and we have*

$$-\operatorname{net}_f(W_{i+1},\theta) = \gamma_i^\theta(W_{i+1}) = o^\theta(S^- \setminus W_{i+1}) \text{ for all } \theta \in [0,\theta_i]. \tag{6.18}$$

Proof. Let f be an earliest arrival transshipment solving $(\mathcal{N},b)_{\text{EAT}}$. We prove the statement of the lemma inductively. We start with $i = 1$. By Theorem 6.7 we know that for all $\theta \in [0,\theta_1]$ we have

$$|f|_\theta = o^\theta(\{s\}) \overset{\text{Thm 6.7}}{=} \gamma_1^\theta(W_1),$$

which shows the first part of the lemma. Again, we have by Theorem 6.7

$$-\operatorname{net}_f(W_2,\theta_1) \overset{\text{Thm. 6.7}}{=} o^{\theta_1}(\{s\} \cup S^- \setminus W_2).$$

Since f restricted to $\mathcal{N}_1$ has pattern $o^\theta(\{s\})$ until time θ_1, we can apply Corollary 6.15 and obtain that

$$-\operatorname{net}_f(W_2,\theta) = \gamma_1^\theta(W_2) \overset{\text{Cor. 6.15}}{=} o^\theta(\{s\} \cup S^- \setminus W_2) \text{ for all } \theta \in [0,\theta_1].$$

We now proceed by induction and assume that the statements of the lemma holds for some $i \geq 0$. Thus, the restriction of f to $\mathcal{N}_i$ has pattern $\theta \mapsto o^\theta(\{s\} \cup S^- \setminus W_i)$ for all $\theta \leq \theta_i$ and

$$-\operatorname{net}_f(W_{i+1},\theta) = o^\theta(\{s\} \cup S^- \setminus W_{i+1}) \text{ for all } \theta \in [0,\theta_1].$$

By Theorem 6.7 we know that

$$-\operatorname{net}_f(W_{i+1},\theta) \overset{\text{Thm. 6.7}}{=} o^\theta(\{s\} \cup S^- \setminus W_{i+1}) \text{ for all } \theta \in [\theta_i,\theta_{i+1}].$$

Thus, f restricted to $\mathcal{N}_{i+1}$ has pattern $\theta \mapsto o^\theta(\{s\} \cup S^- \setminus W_{i+1})$ for all $\theta \leq \theta_{i+1}$, showing the first part of the lemma. Again, we obtain from Theorem 6.7 that

$$-\operatorname{net}_f(W_{i+2},\theta_{i+1}) \overset{\text{Thm. 6.7}}{=} o^{\theta_{i+1}}(\{s\} \cup S^- \setminus W_{i+2}).$$

Thus, we can again apply Corollary 6.15 which yields

$$-\operatorname{net}_f(W_{i+2},\theta) \overset{\text{Cor. 6.15}}{=} o^\theta(\{s\} \cup S^- \setminus W_{i+2}) \text{ for all } \theta \in [0,\theta_{i+1}].$$

□

With the help of Lemma 6.39 we can now proceed to the proof of Lemma 6.38.

Proof of Lemma 6.38. Assume that f is an earliest arrival transshipment solving $(\mathcal{N},b)_{\text{EAT}}$. In order to show $-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$, we need to show that for all $i \in \{1,\ldots,r\}$ the restriction of the vector $-b$ to T_i denoted by $-b|_{T_i}$ is contained in $\mathcal{B}(g_i)$. That is, our goal is to show that

$$-b|_{T_i}(X) \leq \gamma_i^{\theta_i}(W_{i+1} \cup X) - \gamma_i^{\theta_i}(W_{i+1}) \text{ for all } X \subseteq T_i,$$

and $g_i(T_i) = \gamma_i^{\theta_i}(W_i) - \gamma_i^{\theta_i}(W_{i+1}) = -b(T_i)$. The second fact follows immediately from Lemma 6.39,

$$\begin{aligned} g_i(T_i) &= \gamma_i^{\theta_i}(W_i) - \gamma_i^{\theta_i}(W_{i+1}) \\ &\overset{\text{Lem. 6.39}}{=} o^{\theta_i}(\{s\} \cup S^- \setminus W_i) - o^{\theta_i}(\{s\} \cup S^- \setminus W_{i+1}) \\ &= -b(T_i). \end{aligned}$$

It remains to show $-b(X) \leq g_i(X)$ for all $X \subseteq T_i$. To see this, note that by Lemma 6.39 we have

$$-\operatorname{net}_f(W_{i+1}, \theta_i) = \gamma_i^{\theta_i}(W_{i+1}) \overset{\text{Lem. 6.39}}{=} o^{\theta_i}(\{s\} \cup S^- \setminus W_{i+1}).$$

This implies

$$-b(T) = -\operatorname{net}_f(W_{i+1} \cup T, \theta_i) + \operatorname{net}_f(W_{i+1}, \theta_i) \leq \gamma_i^{\theta_i}(W_{i+1} \cup T) - \gamma_i^{\theta_i}(W_{i+1}).$$

□

Note, that $-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$ is not a sufficient criterion for the existence of an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ (see Figure 6.14 for an example). The reason for this is, that even if $-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$

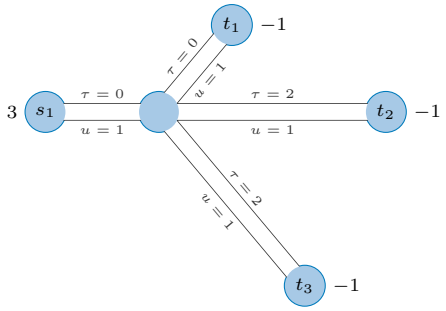


Figure 6.14: Consider the depicted earliest arrival transshipment problem. The corresponding polytope $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ is the convex hull of $(1, 2, 0)$ and $(1, 0, 2)$ (we assume that the x-coordinate corresponds to t_1, the y-coordinate to t_2 and the z-coordinate to t_3). Clearly, $(1, 1, 1) \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$, but also obviously the earliest arrival transshipment problem does not have a solution. This is due to the fact that the generalized lex-max earliest arrival flows corresponding to the vertices of this polytope do not exist (see Lemma 6.40 and Figure 6.13).

it might happen that the generalized lex-max earliest arrival flows corresponding to the vertices of this polytope might not exist. In order to state our main theorem we at first need to derive the following correspondence between the vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ and generalized lex-max earliest arrival flows with respects to the sets T_i and times θ_i for all $i \in \{1, \dots, r\}$.
Our goal is to show that the vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ correspond to generalized lex-max earliest arrival flows and vice versa. By definition we have $\mathfrak{P}^{\text{EAT}}_{s,S^-} = \bigtimes_{i=1}^{r} \mathcal{B}(g_i)$ and we know that the vertices of $\mathcal{B}(g_i)$ are completely characterized by total orders on the ground set of g_i (see Theorem 2.3). We will now proceed symmetrically to what we did in Section 5.3.1. Let $\prec$ be a total order on S^- that respects the given partition. That is, we can split $\prec$ into r total orders $\prec^1, \dots, \prec^r$ on $T_1, \dots, T_r$, respectively, which induce vertices of the corresponding base polytopes and hence yield a vertex of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ and vice versa.

Lemma 6.40. *Denote by $u^{\prec}$ the vertex of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ corresponding to a total order on S^- that respects a given partition of S^-. It holds that*

$$u^{\prec} = -x_{f_{\prec}},$$

where $f_{\prec}$ is a generalized lex-max earliest arrival flow with respect to $T_1, \dots, T_r$, $\theta_1, \dots, \theta_r$ and $\prec$.

Proof. Let $\prec$ be a total order on S^- that respects the given partition and denote by $\prec^1, \dots, \prec^r$ the reduced total orders on $T_1, \dots, T_r$, respectively. For each $i \in \{1, \dots, r\}$ the total oder $\prec^i$ induces a vertex $v^{\prec^i}$ of $\mathcal{B}(g_i)$. The total order $\prec$ thus induces a vertex $u^\prec$ of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ by

$$u^\prec = (v^{\prec^r}, \dots, v^{\prec^1}).$$

Using Lemma 6.31 we get for $t \in T_i$ for all $i \in \{1, \dots, r\}$

$$\begin{aligned} x_{f_\prec}(s) &= \operatorname{net}_{f_\prec}(t) \\ &\overset{\text{Lem. 6.31}}{=} \gamma_i^{\theta_i}(\{t' \in T_i \mid t \prec t'\}) - \gamma_i^{\theta_i}(\{t' \in T_i \mid t \preceq t'\}). \end{aligned}$$

Using the Greedy Algorithm, we get

$$\begin{aligned} u^\prec(t) &= v^{\prec^i}(t) \\ &= \gamma_i^{\theta_i}(\{t' \in T_i \mid t \preceq t'\}) - \gamma_i^{\theta_i}(\{t' \in T_i \mid t \prec t'\}), \end{aligned}$$

and thus $u^\prec = -x_{f_\prec}$. □

Theorem 6.41.

Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single source s. Further assume that $S^- = T_1 \sqcup \dots \sqcup T_r$ and $\theta_1 < \dots < \theta_r$ are the sets and times corresponding to the earliest arrival pattern of $(\mathcal{N}, b)_{\text{EAT}}$.
The earliest arrival transshipment problem has a solution if and only if $-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$ and all the lex-max earliest arrival flows corresponding to the vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ do exist. In case of existence an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ can be obtained as a convex combination of $d \leq |S^-| - r + 1$ generalized lex-max earliest arrival flows with respect to $T_1, \dots, T_r$ and $\theta_1, \dots, \theta_r$. That is, there are total orders $\prec_1, \dots, \prec_d$ on S^- that respect the given partition and convex coefficients $\lambda_1, \dots, \lambda_d \geq 0$ such that

$$\lambda_1 f_{\prec_1} + \dots + \lambda_d f_{\prec_d}$$

solves $(\mathcal{N}, b)_{\text{EAT}}$. Here, $f_{\prec_i}$ is the generalized lex-max earliest arrival flow with respect to $T_1, \dots, T_r$, $\theta_1, \dots, \theta_r$ and $\prec_i$ for all $i \in \{1, \dots, r\}$.

For the proof of this theorem, we need an additional lemma.

Lemma 6.42. *Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single source s. Further assume that $S^- = T_1 \sqcup \dots \sqcup T_r$ and $\theta_1 < \dots < \theta_r$ are the sets and times corresponding to the earliest arrival pattern of $(\mathcal{N}, b)_{\text{EAT}}$. If an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$ does exist, then all generalized lex-max earliest arrival transshipments corresponding to the given partition and times do exist.*

Proof. Let $\prec$ be an arbitrary order on S^- that respects the given partition and let f be an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$. Assume that the generalized earliest arrival transshipment $f_\prec$ corresponding to the given parameters does *not* exist. By Lemma 6.34 there has to be some $i \in \{1, \dots, r\}$ such that

$$-\operatorname{net}_{f_\prec}(W_i, \theta_i) < o^{\theta_i}(\{s\} \cup S^- \setminus W_i).$$

However, whether we have equality in this inequality is independent of the specific choice of an order respecting the partition according to which the flow was sent. In particular this implies,

$$-\operatorname{net}_f(W_i, \theta_i) < o^{\theta_i}(\{s\} \cup S^- \setminus W_i),$$

contradiction. □

Proof of Theorem 6.41. If the earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ has a solution, then we get $-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$ by Lemma 6.38, that is we can obtain the vector $-b$ as convex combination of vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$. Thus, there are convex coefficients $\lambda_1, \dots, \lambda_d$ and total order $\prec_1, \dots, \prec_d$ on S^- that respect the given partition such that

$$-b = \sum_{i=1}^{d} \lambda_i u^{\prec_i}.$$

Here $u^{\prec_i}$ is the vertex of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ corresponding to $\prec_i$. By the previous lemma, we know that all the generalized lex-max earliest arrival flows corresponding to $\prec_1, \dots, \prec_d$ do exist. Define

$$f \coloneqq \sum_{i=1}^{d} \lambda_i f_{\prec_i}.$$

By Observation 6.32 all these flows have pattern p^*, and hence so has f. By Lemma 6.40, we then get

$$x_f = -\sum_{i=1}^{d} \lambda_i u^{\prec_i} = b.$$

Thus, f is an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$. On the other hand, if we assume $-b \in \mathfrak{P}^{\text{EAT}}_{S^+,t}$ and all generalized lex-max earliest arrival flows exist, we can by the same argumentation obtain an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$. That $d \leq |S^-| - r + 1$ follows again from Carathéodory's theorem. □

By doing overall r submodular function minimizations we can check whether $-b \in \mathfrak{P}^{\text{EAT}}_{s,S^-}$. In order to check whether $(\mathcal{N}, b)_{\text{EAT}}$ has a solution we also need to be able to check whether the generalized lex-max earliest arrival flows corresponding to the vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ do exist. Note, that all of the generalized lex-max earliest arrival flows corresponding to the vertices of this polytope correspond to the same partitions and to the same time. They only differ in the choice of the specific total order on S^- respecting the given partition. Thus, Corollary 6.37 implies that all generalized lex-max earliest arrival flow corresponding to the vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ do exist if and only if *one* generalized lex-max earliest arrival flow with respect to $T_1, \dots, T_r$ and $\theta_1, \dots, \theta_r$ does exist. We can thus choose any order $\prec$ on S^- that respects the given partition, and check whether the corresponding generalized lex-max earliest arrival flow does exist.

To compute a suitable convex combination, we can again proceed similar as in Section 5.3.2. Doing submodular function minimization to check whether $-b\big|_{T_i} \in \mathcal{B}(g_i)$ also returns a convex combination of vertices of this polytope yielding the vertex $-b\big|_{T_i}$. As in Algorithm 21 we can combine the r convex combination we get this way to a single convex combination of vertices of $\mathfrak{P}^{\text{EAT}}_{s,S^-}$ yielding the vertex $-b$. Computing the corresponding convex combination of generalized lex-max earliest arrival flow results in a solution to the earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$. Since the resulting algorithms are completely analogue to the ones in Section 5.3.2, we do note state them here. Overall, we just note:

Fact 6.43. *Given an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a single source, we can check in polynomial space whether this problem has a solution and compute one in case of existence.*

6.3.3 Summary, Conclusions and Open Questions

So far, we presented PSPACE algorithms that check whether a given earliest arrival transshipment problems $(\mathcal{N}, b)_{\text{EAT}}$ has a solution and computes this solution in case of existence for the special case of dynamic networks with a single source, or tight problems in general dynamic networks. These algorithms are the first efficient algorithms to compute earliest arrival transshipment problems in dynamic networks with multiple sinks. The straightforward remaining open question is thus to come

up with an efficient way to check the existence of a solution to a non-tight problem $(\mathcal{N}, b)_{\text{EAT}}$ in general dynamic network with multiple sources and multiple sinks and to compute the solution in case of existence.

6.4 Complexity

The main result of this section is the following hardness result:

Theorem 6.44.

Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem. It is $\mathcal{NP}$-hard to decide whether there exists an earliest arrival transshipment solving $(\mathcal{N}, b)_{\text{EAT}}$.

We achieve this result by reducing from Partition. An instance $\mathcal{I}$ of Partition is given by a set $A = \{a_1, a_2, \ldots, a_n\} \subseteq \mathbb{Z}_+$ of natural numbers. The decision problem is to determine whether there exists a subset $A_1 \subseteq A$ such that $\sum_{a \in A_1} a = \sum_{a \in A \setminus A_1} a$.
In our hardness proof we use network gadgets due to Disser and Skutella [DS15]. See [DS15] for an in-depth introduction to these gadgets. We only give a high level description. The gadget Counter(n) (see Figure 6.15) has a source s and a sink t and the property that the SSPA from s to t needs 2^n iterations such that in iteration $j \in \{1, 2, \ldots, 2^n - 1\}$ the shortest s-t path has length j. Assume now that we are given an instance $\mathcal{I}$ of Partition by a set A of size n. The gadget G^A

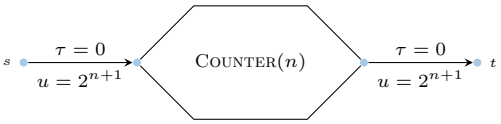
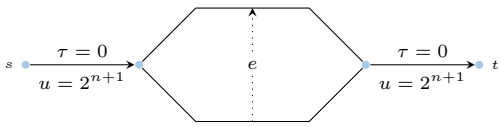
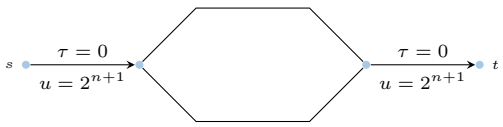


Figure 6.15: The counter gadget

(See Figure 6.16a) corresponding to $\mathcal{I}$ has a source s and a sink t and the main property that the SSPA from s to t sends flow through the special arc e if and only if $\mathcal{I}$ has a solution. Moreover, the SSPA from s to t needs 2^{n+1} iterations. The intuition behind G^A is that in each pair of iterations i and $i+1$ for odd i of the SSPA basically one possible solution for the given partition problem is tried out. If during these iterations a solution for $\mathcal{I}$ is found, then in iteration i the shortest s-t path is a path using arc e in forward direction while in iteration $i+1$ the shortest path uses e in backwards direction. If in iterations i and $i+1$ for odd i no solution to $\mathcal{I}$ is found, then the paths chosen in these two iterations do not use arc e.



(a) The gadget G^A



(b) The gadget $G^A \setminus \{e\}$

Figure 6.16: The gadgets G^A and $G^A \setminus \{e\}$

We will also need the gadget G^A without the special arc e, denoted by $G^A \setminus \{e\}$ (see Figure 6.16b). In each iteration the paths chosen by the SSPA in G^A and $G^A \setminus \{e\}$ are the same, except in iterations i and $i+1$ in which a solution to the partition problem is found. In this case the paths chosen in G^A in these iterations are strictly shorter than the paths chosen in $G^A \setminus \{e\}$. After iterations i and $i+1$ the two gadgets are in the same state again. Given an instance $\mathcal{I} = A$ of Partition, consider the dynamic network $\mathcal{N}_A$ depicted in Figure 6.17. Let T be a time-horizon chosen large enough. Define $b(t_1) = -o^T(\{s, t_2\})$ and $b(t_2) = -(o^T(\{s\}) - o^T(\{s, t_2\}))$. Clearly, $(\mathcal{N}_A, b)$ is feasible with time horizon T since it can be solved by a lex-max flow over time in $\mathcal{N}_A$ with respect to $t_2 \prec t_1$. The following lemma immediately implies Theorem 6.44.

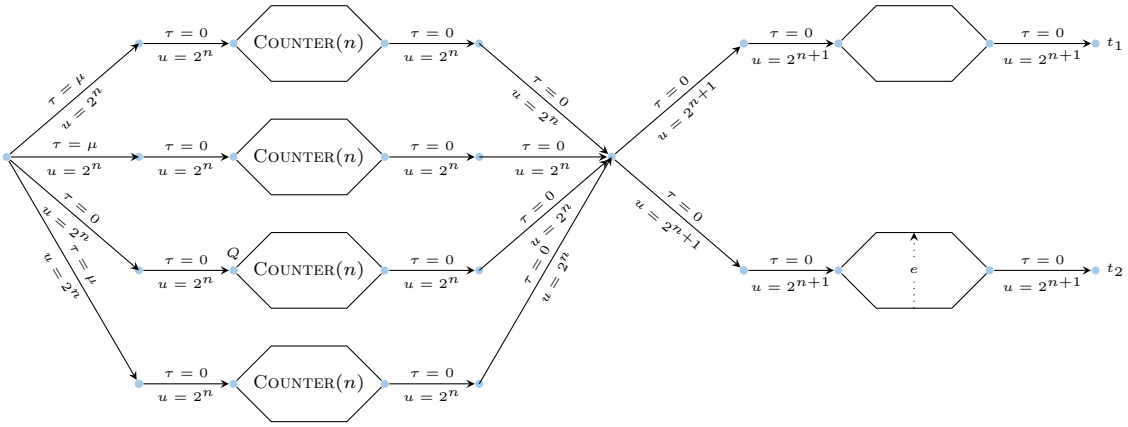


Figure 6.17: The dynamic network $\mathcal{N}_A$ with $0 < \mu < 1$.

Lemma 6.45. *The transshipment problem $(\mathcal{N}_A, b)$ can be solved by an EAT if and only if the given instance of* PARTITION *does not have a solution.*

Proof. By construction the quickest transshipment problem $(\mathcal{N}_A, b)$ is tight for time-horizon T. Thus, we know that the earliest arrival pattern p^*_A is given by $p^*_A(\theta) = o^\theta(\{s\})$. The following statements are equivalent:

- $(\mathcal{N}_A, b)_{\text{EAT}}$ has a solution.
- It is $-b \in \mathcal{B}(\gamma^T)$.
- Algorithm 24 with $t_2 \prec t_1$ computes an earliest arrival transshipment solving $(\mathcal{N}_A, b)_{\text{EAT}}$.
- The given instance of PARTITION does not have a solution.

Theorem 6.21 implies that $(\mathcal{N}_A, b)_{\text{EAT}}$ can be solved by an earliest arrival transshipment if and only if $-b \in \mathcal{B}(\gamma^T)$. This shows the equivalence of the first two points. By definition of γ^T we have $-b \in \mathcal{B}(\gamma^T)$ if and only if $\gamma_T(\{t_1\}) = o^T(\{s, t_2\})$. Thus if $-b$ lies in $\mathcal{B}(\gamma^T)$, the vector $-b$ is also the vertex of this polytope corresponding to the total order $t_2 \prec t_1$. The earliest arrival transshipment corresponding to this vertex can be computed using Algorithm 24. This shows the equivalence of point two and three.

To see the last equivalence we investigate how Algorithm 24 behaves in $\mathcal{N}_A$ depending on whether the given instance of the partition problem has a solution or not. We at first note that in $\mathcal{N}_A$ the algorithm Algorithm 24 works in phases of 4 iterations. At the beginning of each phase both partition gadgets are in the same state. Assume at first that PARTITION does not have a solution. Then in each phase flow is send towards t_1 using the path along node Q and one path through one of the other counters. Through the remaining counters flow is sent towards t_2. Clearly, the paths towards t_1 are the same that the successive shortest path algorithm only towards t_1 chooses. Thus, the flow computed by Algorithm 24 fulfills the demands b.

On the other hand, if PARTITION does have a solution, then there is some phase, in which at the beginning of the phase the path along Q towards t_2 using the arc e is shorter than any path towards t_1. In this iteration intuitively a solution for the given partition problem is found. Thus, Algorithm 24 chooses this path towards t_2 at the beginning of the phase. This however directly implies that f does not satisfy b. □

6.5 Approximation of Earliest Arrival Transshipments

We finish this chapter about earliest arrival transshipments in dynamic networks with multiple sinks with an existence result regarding the approximation of such flows. The only open question regarding the existence of approximations in this setting is the setting of dynamic networks with multiple

sinks and only a single source regarding time-approximation. Here the best known approximation algorithm yields a 4-time approximation while the best known lower bound gives a factor of 2 (see Section 3.2.3). We will show that a 2-time approximation for earliest arrival transshipments in this setting exists. Thus, we close the gap between 2 and 4 and achieve a tight result.
The main idea behind computing such a 2-time approximation is more a consequence of the results from Chapter 5 than of this chapter. Recall, that we showed in Chapter 5 that we can solve a given earliest arrival transshipment problem in a dynamic network with only a single sink by a convex combination of generalized lex-max flows over time that fulfill the earliest arrival property. In this section we will define generalized lex-max flows over time for dynamic networks with multiple sinks and a single source – in such networks these flows do **not** always exist. Completely similar to our results in Chapter 5 it follows that an earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a single source but multiple sinks can be solved by a convex combination of generalized lex-max flows over time that fulfill the earliest arrival property (in case such flows exist). To obtain the approximation factor of 2 we will give a 2-time approximation of generalized lex-max flows over time, i.e., we give an algorithm that computes flows over time with the same characteristic vector as generalized lex-max flows over time that are only by at most a factor of 2 off to fulfill the earliest arrival property. We start by defining generalized lex-max flows over time for dynamic networks with only a single source.

Definition 6.46 (Generalized Lex-Max Flow Over Time).
Let $\mathcal{N} = (D = (V, A), u, \tau, s, S^+)$ be a dynamic network with only a single source s, $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$, $0 < \theta_1 < \theta_2 < \ldots < \theta_r$ and consider a total order $\prec$ on S^- with the following property:

- *The order $\prec$ **respects the given partition** $\prec$, i.e., $s' \prec s$ for all $s \in T_i$ and $s' \in T_j$ with $i < j$.*

*A **generalized lexicographically maximum (lex-max) flow over time** f with respect to $T_1, \ldots, T_r$, $\theta_1, \theta_2, \ldots, \theta_r$, and $\prec$ is a feasible flow over time that fulfills the following properties,*

1. *For $i = 1, 2, \ldots, r$, let $\mathcal{N}_i$ be the dynamic network obtained by attaching a new super-sink ψ_i to the sinks in $T_{i+1} \cup T_{i+2} \cup \ldots \cup T_r$ by arcs with zero transit time and infinite capacity. Denote by $\prec_i$ the total order on $\{\psi_i\} \cup T_i \cup \{s\}$ which is the restriction of $\prec$ to T_i such that $s \prec t \prec \psi_i$ for all $t \in T_i$.*
   We require that the flow sent by f into the sinks in $\{\psi_i\} \cup T_i$ until time θ_i is a lex-max flow over time with respect to $\prec_i$ and time horizon θ_i, for $i = 1, 2, \ldots, r$. Note that we can regard the flow over time f (which is a flow in $\mathcal{N}$ by definition) as a flow over time in $\mathcal{N}_i$ by assuming that the flow sent into the sinks in $T_{i+1} \cup \ldots \cup T_r$ is sent into ψ_i in $\mathcal{N}_i$. This works because the ingoing arcs of ψ_i all have zero transit time and infinite capacity.
2. *For $i = 1, 2, \ldots, r$, after time θ_i no flow sent towards T_i remains in the network and no flow is sent towards the sinks in T_i such that it would arrive at later times.*

From the definition and Lemma 6.18 is is easy to deduce the characteristic vector of a generalized lex-max flow over time:

Lemma 6.47. *Assume we are given a generalized lex-max flow over time problem by a dynamic network $\mathcal{N}$ with a single source s, a partition of S^- into r disjoint subsets $S^- = T_1 \sqcup T_2 \sqcup \ldots \sqcup T_r$, rational times $0 < \theta_1 < \theta_2 < \ldots \theta_r$ and a total order $\prec$ on S^- that respects the given partition of S^-. For a generalized lex-max flow over time f with respect to these parameters we have*

$$\begin{aligned} x_f(t) &= \operatorname{net}_f(t, \theta_i) \\ &= o^{\theta_i}\Big(\{s\} \cup \bigcup_{j=i+1}^{r} T_j \cup \{t' \in T_i \mid t \prec t'\} \cup \{t\}\Big) - o^{\theta_i}\Big(\bigcup_{j=i+1}^{r} T_j \cup \{t' \in T_i \mid t \prec t'\}\Big), \end{aligned}$$

for all $i \in \{1, \ldots, r\}$ and all $t \in T_i$.

Proof. For the proof we refer to the proof of Lemma 6.31, which is completely symmetric to the proof of this lemma. □

Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single source. Let $T_1, \dots, T_r$ and $0 < \theta_1 < \dots < \theta_r$ be the sets and times corresponding to the earliest arrival pattern p^* of $(\mathcal{N}, b)_{\text{EAT}}$ as computed by Algorithm 23. Again completely similar to the symmetric result in Chapter 5 we can deduce:

Theorem 6.48.

An earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network $\mathcal{N}$ with only a single source s with pattern p^* can be solved by a convex combination of generalized lex-max flows over time with respect to $T_1, \dots, T_r$ and $\theta_1 < \dots < \theta_r$ that have pattern p^*. Here $T_1, \dots, T_r$ and $\theta_1 < \dots < \theta_r$ are the sets and times corresponding to the earliest arrival pattern p^* of $(\mathcal{N}, b)_{\text{EAT}}$.

Proof. For the proof we again refer to the proof of the symmetric theorem from Chapter 5 (Theorem 5.19). □

Let $\prec$ be an order on S^- that respects the given partition of S^-. We will present an algorithm that computes a flow over time f with characteristic vector $x_{f_\prec}$, where $f_\prec$ is the generalized lex-max flow over time with respect to $\prec$, the given times and the given partition of S^-. Moreover, f has additionally the property

$$|f|_\theta \geq p^*(\theta/2) \text{ for all } \theta \in [0, 2T).$$

Together with Theorem 6.48 this implies that a 2-time approximation of $(\mathcal{N}, b)_{\text{EAT}}$ does exist.
Our algorithm will be essentially a greedy algorithm. We proceed as in Algorithm 24 with the successive shortest path algorithm that chooses the shortest path according to the given order $\prec$. However, once a sink runs full (i.e., it receives the amount of flow it would receive in a generalized lex-max flow over time with respect to the given parameters), we do not send flow into this sink any more. The algorithm we achieve can be seen in Algorithm 29. It is clear that our algorithm computes a feasible flow over time as we only augment along shortest paths. The only thing that remains to be shown is that the flow over time computed by the algorithms fulfills the desired approximation properties.

Lemma 6.49. *Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single source. Let $T_1, \dots, T_r$ and $0 < \theta_1 < \dots < \theta_r$ be the sets and times corresponding to the earliest arrival pattern p^* of $(\mathcal{N}, b)_{\text{EAT}}$ and $\prec$ an order on S^- that respects the partition $S^- = T_1 \sqcup T_2 \sqcup \dots \sqcup T_r$ of S^-. The flow over time f computed by Algorithm 29 has the property that*

$$|f|_\theta \geq p^*(\theta/2) \text{ for all } \theta \in [0, 2T).$$

and $x_f = x_{f_\prec}$, where $x_{f_\prec}$ is the characteristic vector of the corresponding generalized lex-max flow over time.

Proof. Denote by f the flow over time returned by Algorithm 29. Assume at time η_1 the first sink $t \in S^-$ runs full. That implies in particular that until time η_1 we have

$$|f|_\theta = o^\theta(\{s\}) \text{ for all } \theta \in [0, \eta_1). \tag{6.19}$$

By the construction of the algorithm, no flow arrives at t after time η_1, i.e., all flow towards this sink is rerouted towards other sinks after time η_1. The worst thing that can happen is that the flow sent towards t blocks all paths towards the other sinks, i.e., the worst case is that until time η_1 no flow has arrived at all the other sinks. Another worst case is that the sink t has the highest priority of all the sinks, i.e., flow towards this sink is not supposed to block paths from the source towards any other sink. Assume that $S^- = \{t_1, \dots, t_k\}$ with $t_1 \prec \dots \prec t_k$. We now assume that $t = t_1$ and that the flow that arrives at t_1 until time η_1 blocks all paths towards all other sinks. After time

Algorithm 29: Algorithm for computing an approximation of a generalized lex-max flow over time with earliest arrival pattern

Input : An earliest arrival transshipment problem $(\mathcal{N}, b)_{\text{EAT}}$ in a dynamic network with only a single source s, the partition $S^- = T_1 \sqcup \ldots \sqcup T_r$ and times $\theta_1, \ldots, \theta_r$ corresponding to the earliest arrival pattern p^* of this problem, a total order $\prec$ respecting the partition of S^-, and demands on S^- given by the characteristic vector of the generalized lex-max flow over time with respect to the given parameters. Let $S^- = \{t_1, \ldots, t_k\}$ and $t_1 \prec t_2 \prec \ldots \prec t_k$.

Output : A flow over time f with approximate earliest arrival pattern p^*

$x_P \leftarrow 0$ for all $P \in \overleftrightarrow{\mathcal{P}}$
$x \leftarrow$ static s - S^- flow with generalized path decomposition $(x_P)_{P \in \overleftrightarrow{\mathcal{P}}}$
while *Not all demands are fulfilled* **do**
  $l \leftarrow d(\mathcal{N}_x, s, S^-)$
  for $t \in S^-$ **do**
    if *t runs full before time l* **then**
      $\theta_t \leftarrow$ time at which t runs full
      Attach an arc (t, s) to $\mathcal{N}$ with infinite capacity and transit time $-\theta_t$
    end
  end
  Extend x to the extended network
  for $i = k, k-1, \ldots, 1$ **do**
    if *t_i does not run full before time l* **then**
      while $d(\mathcal{N}_x, s, t_i) = l$ **do**
        $P \leftarrow$ shortest s - t_i path in $\mathcal{N}_x$
        $\gamma \leftarrow \min\{\tau(a) \mid a \in P\}$
        augment x along P by γ
      end
    end
  end
end
$T' \leftarrow$ time horizon at which the last sink runs full
$f \leftarrow$ generalized temporally repeated flow with time horizon T' corresponding to $(x_P)_{P \in \overleftrightarrow{\mathcal{P}}}$.

η_1 no flow arrives at t_1 by construction. Assume that at time η_2 the next sink $t \in S^-$ runs full. We will show that during time $[\eta_1, \eta_2]$ the flow over time f still fulfills inequality (6.19). We again assume that the flow towards t blocks all the paths from s towards any other sink in $S^- \setminus \{t_1, t\}$ and that $t = t_2$. We know that t_1 blocked all the paths towards t_2 until time η_1 and that t_2 has a lower priority than t_1. In order to fulfill the earliest arrival property after time η_1, a flow over time g in $\mathcal{N}$ thus would need to send flow towards t_2 fulfilling its demand and then towards t_1. In such flow over time g the property holds that all the flow towards t_1 arrives at the same time or earlier than all the flow sent towards t_2 (because of the properties of a generalized lex-max flow over time). Assume that at time T_1 the demands of t_1 and t_2 are fulfilled in the flow over time g. Let us consider two specific paths P_1 and P_2 towards t_1 and t_2, respectively. Assume that these paths block each other and that in the above mentioned flow g at first flow is sent into P_2 during $[0, \gamma_2]$ and during $[\gamma_2, \gamma_2 + \gamma_1]$ flow is sent into P_1 such that the overall amount of flow sent into P_1 has arrived at t_1 at the same time or earlier than the whole flow sent towards t_2. In our flow over time f at first flow is sent into P_1 during $[0, \gamma_1]$ and then during $[\gamma_1, \gamma_1 + \gamma_2]$ into P_2. That is, compared to the earliest arrival transshipment the flow into P_2 is late by $\gamma_1 < \eta_1$ time units. In general one can say that in the flow over time f the flow sent towards t_2 is delayed by at most η_1 compared to a flow over time g that fulfills the earliest arrival property after time η_1. The flow over time f has pattern $p^*(\theta)$ until time η_1 by construction. That is, until time $2\eta_1$ the flow over time f for sure fulfills inequality (6.19). If $\eta_2 \leq 2\eta_1$, we are done. If not, we can argue as follows: Since the flow towards t_2 is delayed by at most η_1 in f and the demands of t_1 is already fulfilled at time η_1, we get for all $x \geq 0$

$$|f|_{2\eta_1 + x} \geq p^*(\eta_1 + x) \geq p^*(\eta_1 + x/2).$$

Thus, inequality (6.19) is fulfilled during $[\eta_1, \eta_2]$. Note that than we have in particular $\eta_2 \leq 2T_1$.

We can now proceed similarly and assume that until time η_3 the next sink t is full. Again we assume that t blocks all the paths towards the other sinks and that $t = t_3$. Consider again a flow over time g that fulfills the earliest arrival property after time T_1. In g, in order to fulfill the earliest arrival property after time T_1, at first flow needs to be sent towards t_3, then towards t_2 and then towards t_1. Denote by T_2 the time horizon when all this flow has arrived. We want to show that during $[\eta_2, \eta_3]$ inequality (6.19) is satisfied.
Considering paths P_1, P_2 and P_3 towards t_1, t_2 and t_r, respectively, our assumptions imply that g sends flow into P_3 during $[0, \gamma_3]$, during $[\gamma_3, \gamma_3 + \gamma_2]$ into P_2 and during $[\gamma_3 + \gamma_2, \gamma_3 + \gamma_2 + \gamma_1]$ into P_1 and again all the flow arrives at the sinks at the same time, or at t_1 earliest then at t_2 and then at t_3. In our flow over time f, the order the flow is sent is the other way around. At first flow is sent towards t_1 during $[0, \gamma_1]$, then into P_2 during $[\gamma_1, \gamma_1 + \gamma_2]$ and into P_3 during $[\gamma_1 + \gamma_2, \gamma_1 + \gamma_2 + \gamma_3]$. That is, the flow towards P_3 is overall only delayed by $\gamma_1 + \gamma_2 < T_1$ compared to the flow over time g. Again note that until time $2T_1$ the flow over time f for sure fulfills inequality (6.19) because $|f|_{\eta_2} = p^*(T_1)$ and $\eta_2 \leq 2T_1$. If $\eta_3 \leq 2T_1$ we are thus done. If not, we can argue as follows: Since the flow towards t_3 is delayed by at most T_1 and the demands of t_1 and t_2 are already fulfilled at time η_2, we get for all $x \geq 0$

$$|f|_{2T_1+x} \geq p^*(T_1 + x) \geq p^*(T_1 + x/2).$$

Thus, inequality (6.19) is fulfilled during $[\eta_1, \eta_2]$. Again not that than we have in particular $\eta_3 \geq 2T_2$. Proceeding similarly we can inductively show that f fulfills inequality 6.19 during $[0, 2T]$. □

Overall, we have thus shown that:

Theorem 6.50.

Let $(\mathcal{N}, b)_{\text{EAT}}$ be an earliest arrival transshipment problem in a dynamic network with only a single source. A 2-time approximation solving this problem does exist.

List of Figures

List of Algorithms

Bibliography

[AD86] J. E. Aronson and B. David Chen. **A Forward Network Simplex Algorithm for Solving Multiperiod Network Flow Problems**. *Naval Research Logistics Quarterly*, 33(3): pages 445–467, Wiley, 1986.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. **Network Flows: Theory, Algorithms, and Applications**. Prentice-Hall Inc., 1993.

[Bau07] N. Baumann. **Evacuation by Earliest Arrival FLows**. PhD thesis. TU Dortmund, 2007.

[BCT85] R. E. Bixby, W. H. Cunningham, and D. M. Topkis. **The Partial Order of a Polymatroid Extreme Point**. *Mathematics of Operations Research*, 10(3): pages 367–378, Institute for Operations Research and the Management Sciences INFORMS, 1985.

[BDK93] R. E. Burkard, K. Dlaska, and B. Klinz. **The Quickest Flow Problem**. *Zeitschrift für Operations Research*, 37(1): pages 31–58, Physica-Verlag, 1993.

[Ber09] A. Bernstein. **Fully Dynamic $(2+\varepsilon)$ Approximate All-Pairs Shortest Paths with Fast Query and Close to Linear Update Time**. In: *50th Annual IEEE Symposium on Foundations of Computer Science,* pages 693–702. FOCS 2009. IEEE, 2009.

[Ber78] G. N. Berlin. **The Use of Directed Routes for Assessing Escape Potential**. *Fire Technology*, 14(2): pages 126–135, Kluwer Academic Publishers, 1978.

[BG60] R. G. Busacker and P. J. Gowen. **A Procedure for Determining a Family of Minimum-Cost Network Flow Patterns**. Tech. rep. The Johns Hopkins University, 1960.

[BK04] N. Baumann and E. Köhler. **Approximating Earliest Arrival Flows with Flow-Dependent Transit Times**. In: V. Fiala J.and Koubek and J. Kratochvíl (eds.), *Mathematical Foundations of Computer Science 2004*, pages 599–610. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[Bol54] A. W. Boldyreff. **Determination of the Maximal Steady State Flow of Traffic through a Railroad Network**. *Research Memoranda*, RM-1532, RAND Corporation, 1954.

[BS09] N. Baumann and M. Skutella. **Earliest Arrival Flows with Multiple Sources**. *Mathematics of Operations Research*, 34: pages 499–512, Institute for Operations Research and the Management Sciences INFORMS, 2009. An extended abstract appeared in the Proceedings of FOCS 2006.

[Car07] C. Carathéodory. **Über den Variabilitätsbereich der Koeffizienten von Potenzreihen, die gegebene Werte nicht annehmen**. *Mathematische Annalen*, 64(1): pages 95–115, Springer-Verlag, 1907.

[CFS82] L. G. Chalmet, R. L. Francis, and P. B. Saunders. **Network Models for Building Evacuation**. *Fire Technology*, 18(1): pages 90–113, Springer US, 1982.

[Che77] B. V. Cherkasski. **Algorithm of Construction of Maximal Flows in Networks with Complexity of $\mathcal{O}(|V^2|\sqrt{E})$ Operations**. *Mathematical Methods of Solution of Economical Problems*, 7: pages 112–125, 1977. In Russian.

[Chr+11] P. Christiano, A. A. Kelner, A. Mądry, D. A. Spielman, and S. Teng. **Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs**. In: *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 273–282. STOC 2011. ACM, 2011.

[CHT88] W. Choi, H. W. Hamacher, and S. Tufekçi. **Modeling of building evacuation problems by network flows with side constraints**. *European Journal of Operational Research*, 35(1): pages 98–110, Elsevier BV, 1988.

[Cun83] W. H. Cunningham. **Decomposition of Submodular Functions**. *Combinatorica*, 3(1): pages 53–68, Springer-Verlag, 1983.

[Cun84] W. H. Cunningham. **Testing Membership in Matroid Polyhedra**. *Journal of Combinatorial Theory, Series B*, 36(2): pages 161–188, Elsevier BV, 1984.

[Cun85] W. H. Cunningham. **On Submodular Function Minimization**. *Combinatorica*, 5(3): pages 185–192, Springer-Verlag, 1985.

[Dan51] G. B. Dantzig. **Application of the Simplex Method to a Transportation Problem**. *Activity Analysis of Production and Allocation*: pages 359–373, Wiley, 1951.

[Din70] E. A. Dinic. **Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation**. *Soviet Math Doklady*, 11: pages 1277–1280, 1970.

[Dre+10] D. Dressler, M. Gross, J. W. Kappmeier, et al. **On the use of network flow techniques for assigning evacuees to exits**. In: *Proceedings of the International Conference on Evacuation Modeling and Management*. Volume 3 of *Procedia Engineering*, pages 205–215. Elsevier BV, 2010.

[Dre+11] 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 (eds.), *Operations Research Proceeding*, pages 239–244. Springer, Berlin, Heidelberg, 2011.

[DS15] Y. Disser and M. Skutella. **The Simplex Algorithm is NP-mighty**. In: P. Indyk (ed.), *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 858–872. SODA 2015. Society for Industrial and Applied Mathematics SIAM, 2015.

[DT03] G. B. Dantzig and M. N. Thapa. **Linear Programming 2: Theory and Extensions**. 1st ed. *Springer Series in Operations Research and Financial Engineering*. Springer New York, 2003.

[Edm70] J. Edmonds. **Submodular Functions, Matroids and Certain Polyhedra**. In: *Proceedings of the Calgary International Conference on Combinatorial Structures and Their Applications*, pages 69–87. 1970.

[EFS56] P. Elias, A. Feinstein, and C.E. Shannon. **A Note on the Maximum Flow Through a Network**. *IRE Transactions on Information Theory*, 2(4): pages 117–119, 1956.

[EK72] J. Edmonds and R. M. Karp. **Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems**. *Journal of the ACM (JACM)*, 19(2): pages 248–264, Association for Computing Machinery ACM, 1972.

[ET75] S. Even and R. E. Tarjan. **Network Flow and Testing Graph Connectivity**. *SIAM Journal on Computing*, 4(4): pages 507–518, SIAM, 1975.

[FF54] L. R. Ford and D. R. Fulkerson. **Maximal Flow through a Network**. *Research Memoranda*, RM-1400, RAND Corporation, 1954.

[FF55] L. R. Ford and D. R. Fulkerson. **A Simple Algorithm for Finding Maximal Network Flows and an Application to the Hitchcock Problem**. *Research Memoranda*, RM-1604, RAND Corporation, 1955.

[FF56] L. R. Ford and D. R. Fulkerson. **Maximal Flow Through A Network**. *Canadian Journal of Mathematics*, 8: pages 399–404, Canadian Mathematical Society, 1956.

[FF58] L. R. Ford and D. R. Fulkerson. **Constructing Maximal Dynamic Flows from Static Flows**. *Operations Research*, 6(3): pages 419–433, Institute for Operations Research and the Management Sciences INFORMS, 1958.

[FF62] L. R. Ford and D. R. Fulkerson. **Flows in Networks**. Princeton University Press, Princeton, NJ, USA, 1962.

[FI00] L. Fleischer and S. Iwata. **Improved Algorithms for Submodular Function Minimization and Submodular Flow**. In: *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, pages 107–116. STOC 2000. ACM, 2000.

[Fin] K. Finley. **High-frequency trading: when milliseconds mean millions**. March 2013.

[Fle01] L. Fleischer. **Faster Algorithms for the Quickest Transshipment Problem**. *SIAM Journal on Optimization*, 12(1): pages 18–35, Society for Industrial and Applied Mathematics SIAM, 2001.

[FS07] L. Fleischer and M. Skutella. **Quickest Flows Over Time**. *SIAM Journal on Computing*, 36(6): pages 1600–1630, Society for Industrial and Applied Mathematics SIAM, 2007.

[Fuj58] S. Fujishige. **Submodular Functions and Optimization**. 2nd ed. Volume 58 of *Annals of Discrete Mathematics*. Elsevier, 1958.

[Ful61] R. D. Fulkerson. **An Out-of-Kilter Method for Minimal-Cost Flow Problem**. *Journal of The Society for Industrial and Applied Mathematics*, 9(1): pages 18–27, Society for Industrial and Applied Mathematics SIAM, 1961.

[Gal58] T. Gallai. **Maximum-Minimum Sätze über Graphen**. *Acta Mathematica Academiae Scientiarum Hungaricae*, 9(3-4): pages 395–434, Kluwer Academic Publishers, 1958.

[Gal59] D. Gale. **Transient flows in networks.** *The Michigan Mathematical Journal*, 6(1): pages 59–63, Michigan Mathematical Journal, 1959.

[Gal78] Z. Galil. **A New Algorithm for the Maximal Flow Problem**. In: *19th Annual Symposium on Foundations of Computer Science*, pages 231–245. SFCS 1978. IEEE, 1978.

[GGJ17] M. X. Goemans, S. Gupta, and P. Jaillet. **Discrete Newton's Algorithm for Parametric Submodular Function Minimization**. In: *International Conference on Integer Programming and Combinatorial Optimization*, pages 212–227. 2017.

[GGT89] G. Gallo, M.D. Grigoriadis, and R.E. Tarjan. **A Fast Parametric Maximum Flow Algorithm and Applications**. *SIAM Journal on Computing*, 18(1): pages 30–55, SIAM, 1989.

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver. **The Ellipsoid Method and its Consequences in Combinatorial Optimization**. *Combinatorica*, 1(2): pages 169–197, Springer-Verlag, 1981.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. **Geometric Algorithms and Combinatorial Optimization**. Volume 2 of *Algorithms and Combinatorics*. Springer Nature, 1988.

[GN80] Z. Galil and A. Naamad. **An $\mathcal{O}(|E||V|\log^2|V|)$ algorithm for the maximal flow problem**. *Journal of Computer and System Sciences*, 21(2): pages 203–217, Elsevier BV, 1980.

[GR98] A. V. Goldberg and S. Rao. **Beyond the Flow Decomposition Barrier**. *Journal of the ACM (JACM)*, 45(5): pages 783–797, ACM, 1998.

[Gro+12] M. Gross, J. W. Kappmeier, D. R. Schmidt, and M. Schmidt. **Approximating Earliest Arrival Flows in Arbitrary Networks**. In: L. Epstein and P. Ferragina (eds.), *Algorithms - ESA 2012*. ESA 2012. Volume 7501 of *Lecture Notes in Computer Science*, pages 551–562. Springer, Berlin, Heidelberg, 2012.

[GT87] A. V. Goldberg and R. E. Tarjan. **Solving Minimum-cost Flow Problems by Successive Approximation**. In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. STOC '87*, pages 7–18. ACM, 1987.

[GT88] A. V. Goldberg and R. E. Tarjan. **A New Approach to the Maximum-Flow Problem**. *Journal of the ACM (JACM)*, 35(4): pages 921–940, ACM, 1988.

[GT90] A. V. Goldberg and R. E. Tarjan. **Finding Minimum-Cost Circulations by Successive Approximation**. *Mathematics of Operations Research*, 15(3): pages 430–466, Institute for Operations Research and the Management Sciences INFORMS, 1990.

[HHS03] A. Hall, S. Hippler, and M. Skutella. **Multicommodity Flows Over Time: Efficient Algorithms and Complexity**. In: J.C.M. Baeten, J.K. Lenstra, and G.J. Woeginger (eds.), *Automata, Languages and Programming*. ICALP 2003. Volume 2719 of *Lecture Notes in Computer Science*, pages 397–409. Springer, Berlin, Heidelberg, 2003.

[Hit41] F. L. Hitchcock. **The Distribution of a Product from Several Sources to Numerous Localities**. *Journal of Mathematics and Physics*, 20(1-4): pages 224–230, Wiley-Blackwell, 1941.

[HO84] B. Hajek and R. G. Ogier. **Optimal Dynamic Routing in Communication Networks with Continuous Traffic**. *Networks*, 14(3): pages 457–487, Wiley-Blackwell, 1984.

[HR55] T. E. Harris and F. S. Ross. **Fundamentals of a Method for Evaluating Rail Net Capacities**. *Research Memoranda*, RM-1573, RAND Corporation, 1955.

[HT00] B. Hoppe and É. Tardos. **The Quickest Transshipment Problem**. *Mathematics of Operations Research*, 25(1): pages 36–62, Institute for Operations Research and the Management Sciences INFORMS, 2000. Parts of this article were published as extended abstracts in [HT94] and [HT95].

[HT02] H. W. Hamacher and S. A. Tjandra. **Mathematical Modelling of Evacuation Problems: A State of the Art**. In: M. Schreckenberg and S. D. Sharma (eds.). **Pedestrian and Evacuation Dynamics**, pages 227–266. Springer, 2002.

[HT94] B. Hoppe and É. Tardos. **Polynomial Time Algorithms for Some Evacuation Problems**. In: D. D. K. Sleator (ed.), *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 433–441. SODA 1994. Society for Industrial and Applied Mathematics SIAM, 1994.

[HT95] B. Hoppe and É. Tardos. **The Quickest Transshipment Problem**. In: K. L. Clarkson (ed.), *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 512–521. SODA 1995. Society for Industrial and Applied Mathematics, 1995.

[IFF01] S. Iwata, L. Fleischer, and S. Fujishige. **A Combinatorial Strongly Polynomial Algorithm for Minimizing Submodular Functions**. *Journal of the ACM (JACM)*, 48(4): pages 761–777, Association for Computing Machinery ACM, 2001.

[Iri60] M. Iri. **A new Method for Solving Transportation-Network Problems**. *Journal of the Operations Research Society of Japan*, 3: pages 27–87, 1960.

[Jew58] W. S. Jewell. **Optimal flow through networks**. Tech. rep. Interim Technical Report 8. MIT, 1958.

[Joc] K. Jochum. **'Primal fear' as people across Hawaii get false alert of missile attack**. January 2018.

[JR82] J. J. Jarvis and H. Ratliff. **Note—Some Equivalent Objectives for Dynamic Network Flow Problems**. 28: pages 106–109, Institute for Operations Research and the Management Sciences INFORMS, 1982.

[Kap14] J. W. Kappmeier. **Generalizations of Flows over Time with Applications in Evacuations Optimization**. PhD thesis. TU Berlin, 2014.

[Kar74] A. V. Karzanov. **Nakhozhdenie maksimal'nogo potoka v seti metodom predpotokov**. *Doklady Akademii Nauk SSSR*, 215(1): pages 49–52, 1974. English translation: A. V. Karzanov. **Determining a Maximal Flow in a Network by the Method of Preflows**. *Soviet Mathematics Doklady*, 15(2): 434–437, 1974.

[Kel+14] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford. **An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations**. In: C. Chekuri (ed.), *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 217–226. SODA 2014. Society for Industrial and Applied Mathematics SIAM, 2014.

[KKT06] N. Kamiyama, N. Katoh, and A. Takizawa. **An Efficient Algorithm for Evacuation Problems in Dynamic Network Flows with Uniform Arc Capacity**. In: SW. Cheng and C. K. Poon (eds.), *Algorithmic Aspects in Information and Management*. AAIM 2006. Volume 4041 of *Lecture Notes in Computer Science*, pages 231–242. Springer, Berlin, Heidelberg, 2006.

[KKT09] N. Kamiyama, N. Katoh, and A. Takizawa. **An efficient algorithm for the evacuation problem in a certain class of networks with uniform path-lengths**. *Discrete Applied Mathematics*, 157: pages 3665–3677, Elsevier BV, 2009.

[Kle67] M. Klein. **A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems**. *Management Science*, 14(3): pages 205–220, Institute for Operations Research and the Management Sciences INFORMS, 1967.

[Kli] B. Klinz. Cited as personal comunication (1994) in [HT00].

[KRT94] V. King, S. Rao, and R. E. Tarjan. **A Faster Deterministic Maximum Flow Algorithm**. *Journal of Algorithms*, 17(3): pages 447–474, Elsevier BV, 1994.

[KV12] B. Korte and J. Vygen. **Combinatorial Optimization: Theory and Algorithms**. 5th ed. Springer, Berlin, Heidelberg, 2012.

[LJ15] M. Lin and P. Jaillet. **On the Quickest Flow Problem in Dynamic Networks - A Parametric Min-Cost Flow Approach**. In: P. Indyk (ed.), *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1343–1356. SODA 2015. Society for Industrial and Applied Mathematics SIAM, 2015.

[LRS13] Y. T. Lee, S. Rao, and N. Srivastava. **A New Approach to Computing Maximum Flows using Electrical Flows**. In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 755–764. STOC 2013. ACM, 2013.

[LS14] Y. T. Lee and A. Sidford. **Path Finding Methods for Linear Programming: Solving Linear Programs in $\widetilde{\mathcal{O}}(\sqrt{\text{rank}})$ Iterations and Faster Algorithms for Maximum Flow**. In: *55th Annual IEEE Symposium on Foundations of Computer Science*, pages 424–433. FOCS 2014. IEEE, 2014.

[LSW15] Y. T. Lee, A. Sidford, and S. C.-W. Wong. **A Faster Cutting Plane Method and Its Implications for Combinatorial and Convex Optimization**. In: *56th Annual Symposium on Foundations of Computer Science*, pages 1049–1065. SFCS 2015. IEEE, 2015.

[Mąd13] A. Mądry. **Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back**. In: *54th Annual IEEE Symposium on Foundations of Computer Science*, pages 253–262. FOCS 2013. IEEE, 2013.

[Mąd16] A. Mądry. **Computing Maximum Flow with Augmenting Electrical Flows**. In: *57th Annual IEEE Symposium on Foundations of Computer Science*, pages 593–602. FOCS 2016. IEEE, 2016.

[Mam+06] S. Mamada, T. Uno, K. Makino, and S. Fujishige. **An $\mathcal{O}(n \log^2 n)$ Algorithm for a Sink Location Problem in Dynamic Tree Networks**. *Discrete Applied Mathematics*, 154: pages 2387–2401, Elsevier BV, 2006.

[McC05] S. T. McCormick. **Submodular Function Minimization**. In: K. Aardal, G.L. Nemhauser, and R. Weismantel (eds.), *Discrete Optimization*. Volume 12 of *Handbooks in Operations Research and Management Science*, pages 321–391. Elsevier, 2005.

[Meg74] N. Megiddo. **Optimal Flows in Networks with Multiple Sources and Sinks**. *Mathematical Programming*, 7(1): pages 97–107, Springer-Verlag, 1974.

[Meg79] N. Megiddo. **Combinatorial Optimization with Rational Objective Functions**. *Mathematics of Operations Research*, 4(4): pages 414–424, Institute for Operations Research and the Management Sciences INFORMS, 1979.

[Meg83] N. Megiddo. **Applying Parallel Computation Algorithms in the Design of Serial Algorithms**. *Journal of the ACM (JACM)*, 30(4): pages 852–865, Association for Computing Machinery ACM, 1983.

[Men27] K. Menger. **Zur Allgemeinen Kurventheorie**. *Fundamenta Mathematicae*, 10(1): pages 96–115, Institute of Mathematics Polish Academy of Sciences, 1927.

[Min73] E. Minieka. **Maximal, Lexicographic, and Dynamic Network Flows**. *Operations Research*, 21(2): pages 517–527, Institute for Operations Research and the Management Sciences INFORMS, 1973.

[MKM78] V. M. Malhotra, M. P- Kumar, and S. N. Maheshwari. **An $\mathcal{O}(|V|^3)$ Algorithm for Finding Maximum Flows in Networks**. *Information Processing Letters*, 7(6): pages 277–278, Elsevier BV, 1978.

[Nag07] K. Nagano. **A Faster Parametric Submodular Function Minimization Algorithm and Applications**. Tech. rep. University of Tokyo, Technical Report METR, 2007.

[Orl09] J. B. Orlin. **A faster strongly polynomial time algorithm for submodular function minimization**. *Mathematical Programming*, 118(2): pages 237–251, Springer, Berlin, Heidelberg, 2009.

[Orl85] J. B. Orlin. **On the Simplex Algorithm for Networks and Generalized Networks**. In: R. W. Cottle (ed.). **Mathematical Programming Essays in Honor of George B. Dantzig Part I**, pages 166–178. 4th ed. Springer, Berlin, Heidelberg, 1985.

[Orl93] J. B. Orlin. **A Faster Strongly Polynomial Minimum Cost Flow Algorithm**. *Operations Research*, 41(2): pages 338–350, Institute for Operations Research and the Management Sciences INFORMS, 1993.

[Orl97] J. B. Orlin. **A polynomial time primal network simplex algorithm for minimum cost flows**. *Mathematical Programming*, 78(2): pages 109–129, Springer-Verlag, Aug. 1997.

[Phi90] A. Philpott. **Continuous-Time Flows in Networks**. *Mathematics of Operations Research*, 15(4): pages 640–661, Institute for Operations Research and the Management Sciences INFORMS, 1990.

[PJO95] W. B. Powell, P. J., and A. Odoni. **Chapter 3 Stochastic and dynamic networks and routing**. In: *Network Routing*. Volume 8 of *Handbooks in Operations Research and Management Science*, pages 141–295. Elsevier, 1995.

[Rob50] J. Robinson. **A Note on the Hitchcock-Koopmans Problem**. *Research Memoranda*, RM-407, RAND Corporation, 1950.

[RSS11] S. Ruzika, H. Sperber, and M. Steiner. **Earliest Arrival Flows on Series-Parallel Graphs**. *Networks*, 57(2): pages 169–173, Wiley-Blackwell, 2011.

[RT] D. Richardson and É. Tardos. Cited as personal comunication (2002) in [FS07].

[Sch00] A. Schrijver. **A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time**. *Journal of Combinatorial Theory, Series B*, 80(2): pages 346–355, Elsevier BV, 2000.

[Sch03] A. Schrijver. **Combinatorial Optimization: Polyhedra and Efficiency**. Springer, Berlin, Heidelberg, 2003.

[She13] J. Sherman. **Nearly Maximum Flows in Nearly Linear Time**. In: *54th Annual IEEE Symposium on Foundations of Computer Science*, pages 263–269. FOCS 2013. IEEE, 2013.

[Shi78] Y. Shiloach. **An $\mathcal{O}(n \cdot I \log^2 I)$ Maximum-flow Algorithm**. Computer Science Department, Stanford University, 1978.

[Sku09] M. Skutella. **An Introduction to Network Flows Over Time**. In: W. Cook, L. Lovász, and J. Vygen (eds.), *Research Trends in Combinatorial Optimization*, pages 451–482. Springer, Berlin, Heidelberg, 2009.

[Sle80] D. D. K. Sleator. **An $\mathcal{O}(nm \log n)$ Algorithm for Maximum Network Flow**. PhD thesis. Stanford, California: Department of Computer Science, Stanford University, 1980.

[SS14] M. Schmidt and M. Skutella. **Earliest Arrival Flows in Networks with Multiple Sinks**. *Discrete Applied Mathematics*, 164: pages 320–327, Elsevier BV, 2014.

[SS17a] M. Saho and M. Shigeno. **Cancel-and-Tighten Algorithm for Quickest Flow Problems**. *Networks*, 69(2): pages 179–188, Wiley Online Library, 2017.

[SS17b] M. Schlöter and M. Skutella. **Fast and Memory-Efficient Algorithms for Evacuation Problems**. In: Philip N. Klein (ed.), *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms. SODA 2017*, pages 821–840. Society for Industrial and Applied Mathematics SIAM, 2017.

[ST83] D. D. K. Sleator and R. E. Tarjan. **A data structure for dynamic trees**. *Journal of Computer and System Sciences*, 26(3): pages 362–391, Elsevier, 1983.

[SW10] S. Stiller and A. Wiese. **Increasing Speed Scheduling and Flow Scheduling**. In: O. Cheong, K.-Y. Chwa, and K. Park (eds.), *Algorithms and Computation. ISAAC 2010*. Volume 6507 of *Lecture Notes in Computer Science*, pages 279–290. Springer Berlin Heidelberg, 2010.

[Tar97] R. E. Tarjan. **Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm**. *Mathematical Programming*, 78(2): pages 169–177, Springer-Verlag, 1997.

[Tol30] A. Tolstoĭ. **Metody nakhozhdeniya naimen'shego summovogo kilometrazha pri planirovanii perevozok v prostranstve**. *Planirovanie Perevozok, Sbornik pervyi*, 1: pages 23–55, Transpechat'NKPS, 1930. In Russian: **Methods of Finding the Minimal Total Kilometrage in Cargo-Transportation Planning in Space**. *Transpartation Planning*, Volume I, TransPress of the National Commissariat of Transportation.

[Tom71] N. Tomizawa. **On some techniques useful for solution of transportation network problems**. *Networks*, 1(2): pages 173–194, Wiley-Blackwell, 1971.

[Top78] D. M. Topkis. **Minimizing a Submodular Function on a Lattice**. *Operations Research*, 26(2): pages 305–321, Institute for Operations Research and the Management Sciences INFORMS, 1978.

[Tov] A. Tovey. **High-frequency trading: when milliseconds mean millions**. April 2014.

[VV02] R. Van Oostrum and R. C. Veltkamp. **Parametric Search Made practical**. In: *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, pages 1–9. SCG 2002. Association for Computing Machinery ACM, 2002.

[Wil71] W. L. Wilkinson. **An Algorithm for Universal Maximal Dynamic Flows in a Network**. *Operations Research*, 19(7): pages 1602–1612, Institute for Operations Research and the Management Sciences INFORMS, 1971.

[WP05] I. Wegener and R. Pruim. **Complexity Theory: Exploring the Limits of Efficient Algorithms**. Springer, New York, 2005.

[WS11] D. P. Williamson and D. B. Shmoys. **The Design of Approximation Algorithms**. 1st. Cambridge University Press, New York, NY, USA, 2011.

[Zad73] N. Zadeh. **A Bad Network Problem for the Simplex Method and other Minimum Cost Flow Algorithms**. *Mathematical Programming*, 5(1): pages 255–266, Springer-Verlag, 1973.