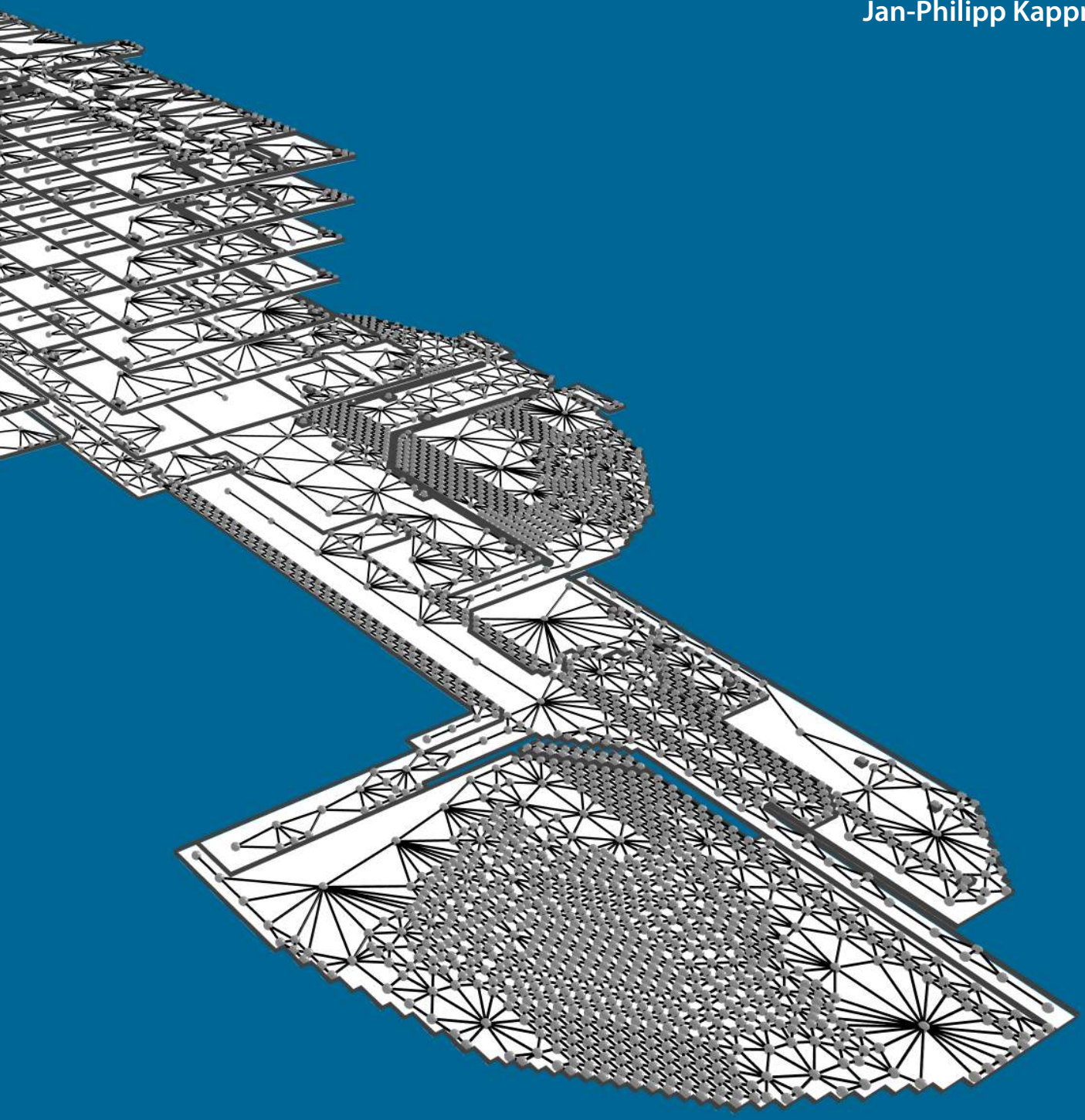
Generalizations of Flows over Time with Applications in Evacuation Optimization

Jan-Philipp Kappmeier



Generalizations of Flows over Time with Applications in Evacuation Optimization

Jan-Philipp Kappmeier

Cover image:
Visualization of the mathematical department of *Technische Universität Berlin* in the ZET software.

Generalizations of Flows over Time with Applications in Evacuation Optimization

vorgelegt von

Diplom-Informatiker
Jan-Philipp W. Kappmeier
geboren in Herten, Westf.

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. Wolfgang König
Berichter:	Prof. Dr. Martin Skutella
Berichter:	Prof. Dr. Ekkehard Köhler

Tag der wissenschaftlichen Aussprache: 19. 12. 2014

Berlin 2015

Bibliographische Informationen der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über `http://dnb.d-nb.de` aubrufbar.

Verlag:
epubli GmbH, Berlin, `www.epubli.de`
ISBN 978-3-7375-3238-9

Final Revision:
29. 01. 2015

Typesetting:
XeLaTex, KOMA-Script, TikZ, PGFPlots

Fonts:
Adobe Minion Pro, Adobe Myriad Pro, typoma Minion Math

This thesis would not have been possible without the many great free and open source projects, especially the work by Donald Knuth and Till Tantau.

Acknowledgements

This thesis is the result of the work of the last years in the COGA research group at Technische Universität, Berlin. I am very thankful for the nice atmosphere in the group that made it enjoyable to continue the work, even when some results refused to be proven. I am also glad to become friends with some members and want to thank everyone who was involved with the creation of this thesis in some way, especially during the coffee breaks in the *Nerdlich*.

First of all, I would like to thank my advisor Martin Skutella for bringing me in contact with the topic of network flows over time and for giving me the necessary degree of freedom for my research. Also, he is a rich source of new ideas in every situation. I am also very grateful to Ekkehard Köhler, who kindly agreed to take the second assessment of the thesis.

This thesis would not have been possible without the help of the people I did research with, in particular Jannik Matuschke and Britta Peis and our wonderful Dortmund-Connection with Martin Groß, Daniel Schmidt, and Melanie Schmidt.

I also would like to thank Martin Groß, Robert Lehmann, Daniel Schmidt, Melanie Schmidt, José Verschae and Wolfgang Welz for carefully reading parts of the thesis and giving me tons of helpful suggestions on the presentation. In particular, the discussions with Melanie were extremely helpful.

Finally, I want to thank Manu for her limitless support and unshakable belief in me.

Berlin, Oktober 2014 *Jan-Philipp Kappmeier*

Contents

Jeder Intellektuelle hat eine ganz besondere Verantwortung. Er hatte das Privileg und die Gelegenheit, zu studieren; dafür schuldet er es seinen Mitmenschen (oder „der Gesellschaft“), die Ergebnisse seiner Studien in der einfachsten und klarsten und verständlichsten Form darzustellen.

– Karl Popper

1 Introduction

We are surrounded by *networks* almost everywhere and our life strongly depends on these networks. Some are virtual, such as the networks we may build with colleagues and friends. The class of non-virtual networks divides again into hidden networks such as the internet (or telecommunication networks in general) and the publicly visible networks such as street networks. All of those non-virtual networks can be used to send some kind of goods, for example freight in logistic networks, data in communication networks and cars or people in the streets. Optimal routing of goods in such networks is the topic of this thesis and we will only consider non-virtual networks in the following. A network can be modelled using a graph of *vertices* that are connected by directed *arcs*. Goods are sent through the network along the arcs from starting vertices to target vertices, which we call *sources* and *sinks*. Huge effort is put in handling the transportation of a continuously growing amount of these goods through larger and larger networks.

The first formalization of the above setting is due to Monge [Mon81] who defined the *transportation problem* that deals with the task to send goods from sources to their respective sinks. Tolstoĭ [Tol30] studied the transportation problem in the Soviet railway network using *network flows*. A flow assigns a value to each arc that represents an amount of the goods that move from one vertex to another. The model is extended by some naturally deduced requirements to restrict capacity or enforce that no flow is lost. While Tolstoĭ tried to send goods through the network in the cheapest way, Ford and Fulkerson answered the question how the same Soviet rail network can be destroyed with lowest costs. They established the important structural property that the value of a maximum flow (from a source to a sink) in a network is equal to the value of a minimum cut [FF56]. Today, network flow theory is an important field of research that comprises many different models and covers various applications.

Ford and Fulkerson already discussed the first extension including non-negative *transit times* on arcs [FF58]. The transit time of an arc denotes the amount of time one unit of flow needs to travel from the beginning of the arc to the vertex at the end. Ford and Fulkerson have shown that in this setting it is possible to compute a *maximum flow over time* efficiently, e. g., a flow sending as many flow units as possible within a given time horizon. The inclusion of the temporal dimension allows the task to minimize the time horizon as an additional optimization objective.

The consideration of time in network flows comes at a cost: Solution sizes can grow large, because a flow over time does not only have to specify how much flow is on any arc, but has to specify this for each point in time in the worst case. This behaviour can lead to pseudo-polynomially large outputs which we would like to avoid. The maximum flow over time algorithm by Ford and Fulkerson [FF58] has the interesting property that the flow rate does not change over time. Therefore, the solution can still be encoded in polynomial size. Unfortunately, this property cannot be maintained if flow problems become more complex. A classical variant of solving problems including a temporal dimension is *time expansion*. A temporally expanded network consists of several copies of the original network and the flow over time is simulated by a static flow in the larger network. These networks easily grow too large and it is therefore advisable to avoid them. If this is not possible, the next best result is an

algorithm that takes the original network as input and works without time expansion.

Evacuation Optimization. A very important application of flows over time, that is also discussed in detail in this thesis, is the *evacuation problem*: Here the task is to find a way how the evacuees can escape from an endangered area as quickly as possible. Evacuation scenarios can be modelled using network flows. The environment is modelled as a network, the evacuees are placed in the sources and the safe area corresponds to sinks. Such models have first been mentioned by Berlin [Ber78] and Chalmet, Francis and Saunders [CFS82]. An evacuation can be represented by a flow over time with minimal time horizon that routes the evacuees to the sinks; a task that is referred to as *quickest transshipment problem*. However, because the situation may worsen at some unknown point in time during the evacuation process, it is most advisable to evacuate as many evacuees as early as possible. This stronger requirement is captured by *earliest arrival flows*.

The requirement that a single flow over time sends as many flow units as early as possible is quite high and it is not clear if such a flow exists. For the case of a single source and a single sink, i. e., all evacuees in an evacuation scenario are heading to a single safe place, this was proven by Gale [Gal59]. An algorithm to compute such a flow in networks with a single source and a single sink was given by Minieka [Min73]. The current state of the art for computing earliest arrival flows is an algorithm due to Baumann and Skutella [BS09]. It works in the case of multiple sources and a single sink. Its running time is polynomial in the input plus output size.

For many evacuation scenarios the restriction of a single available sink is not a big problem. Several exits can typically be modelled by a single sink representing a safe area of infinite capacity, e. g., outside of a building. More complicated scenarios using safe areas with limited capacities such as shelters or life boats on nautical vessels require the introduction of multiple sinks. In these examples a network flow satisfying the earliest arrival property does not necessarily exist. Simple scenarios with only two sinks already show this.

A common approach in such unsatisfactory situations, which do not allow for a solution, is to relax the requirements. In the case of earliest arrival transshipments an approximate solution can send less than the maximal possible flow at some points in time. Quality can then be measured by how far such an approximate flow deviates from the possible maximum flow value at each point in time. Baumann and Köhler [BK07] propose to allow flow units to arrive a certain factor late. Under this relaxed condition there exist approximate earliest arrival flows that send flow a constant factor later.

In case that no solution exists and the requirements have to be relaxed we are interested in solutions that violate the constraints only as little as possible. An ideal algorithm in this setting does not only guarantee a certain bound on the maximum violation, bot computes *instance optimal* solutions. That is, for every instance the solution is as good as possible and if an instance allows for a solution that adheres to the given constraints, such a solution is computed. This is especially important in the scenario of evacuation optimization where any improvement may safe lives. For the case of earliest arrival flows such an algorithm computes an earliest arrival flow when this is possible. If no earliest arrival flow exist an instance optimal algorithm computes a solution with minimal violation of the constraints.

Further Generalizations. Despite the applications of flows over time, current basic research analyses the generalization of other combinatorial problems with a temporal dimension. One of these problem is a dynamic formulation of more general packing integer programs. A first step into this

direction is the temporal extension of *abstract flows*. This model grasps the essence of what determines a network. The concept was introduced by Hoffman [Hof74] when he observed that Ford and Fulkerson's original proof for the Max-Flow=Min-Cut-Theorem in [FF56] only uses very few properties of networks. In the abstract setting, paths are defined to be arbitrary subsets over a fixed ground set allowing a certain *switching* operation: If two paths intersect in an element of the ground set, there is another path using only elements of the beginning of one path and only elements in the end of the other. A general framework that allows to define temporal variants of many combinatorial problems is due to Adjiashvili et al. [ABW+14].

Structure and Contributions of the Thesis

Mathematical research is often driven by real world problems, especially in the area of combinatorial optimization. The mentioned problems regarding the Soviet rail network are examples for such real world applications. Research on a given problem typically initializes two approaches. In a practical approach the original application is tackled. This usually involves building an abstracted model upon which solutions that are applicable in practice are produced. At the same time, a new line of research starts that aims for further generalizations and abstractions of the original problem. When such generalizations are established, it is an interesting question whether the generalizations again are applicable in practice. This thesis contributes to both the practical and theoretical areas of research.

The practical part focuses on the evacuation problem. Practitioners do not use advanced network flow models in existing evacuation tools. Most of those tools only provide a simulation framework to estimate outcomes of evacuation scenarios. From the perspective of combinatorial optimization, the tools are too limited and ignore the potential of *evacuation optimization*. In this thesis we propose approaches based on network flows over time to improve egress times of building evacuations. We also develop a network model for evacuation scenarios that can easily be implemented in existing software tools.

To better handle scenarios that do not allow for an earliest arrival flow we propose the concept of *value approximation*. Value-approximate flows relax the earliest arrival property such that at every point in time only a factor of the possible maximum amount of flow has to be sent. To draw a complete picture of the landscape of approximate earliest arrival flows we juxtapose the existing approach of temporal approximate earliest arrival flows with the new concept of value-approximate flows.

On the theoretical line of research we generalize network flow over time problems by allowing negative travel times on arcs. Instances of these types appear when we consider a temporal variant of the matching problem. We also discuss the existence of abstract earliest arrival flows.

In the following we briefly outline the structure of this thesis and the contributions of each chapter.

Chapter 1: Introduction. In the remainder of this chapter we briefly introduce important notations and definitions that we use throughout the thesis. We also give an introduction into static network flows which are the main topic of the thesis. The introduction covers a short historical review and introduces edge- and path based formulations as well as special variants of static network flows.

Chapter 2: Flows over Time. In Chapter 2 we introduce the field of network flows over time and the Quickest Transshipment Problem and Earliest Arrival Transshipment Problem

and consider time expansion as a method to solve flow over time problems. We in depths discuss the hardness of a generalization of the Quickest Transshipment Problem in which arcs are additionally equipped with release dates and deadlines. The extension was introduced and already briefly analysed by Hoppe [Hop95]. We show Minieka's algorithm to compute earliest arrival flows for instances with a single sink using the Successive Shortest Path Algorithm and discuss in which cases the algorithm can be applied to compute earliest arrival flows with multiple commodities.

Chapter 3: Evacuation Simulation and Optimization. In Chapter 3 we study how different network flow algorithms and especially earliest arrival flows can be used to improve evacuation times. To optimize evacuation times we propose *exit assignments*, which define the exit taken by each evacuee. We apply different algorithms to compute exit assignments and compare the effectiveness of the assignments by simulation runs with the cellular automaton implemented in the software suite zet[1]. Software tools used to model evacuation scenarios in practice are often built upon simulation frameworks using cellular automata. We present an approach to generate network flow models based on an already existing cellular automaton. To prove validity of our automatically generated network models we compare the results of an earliest arrival flow with simulation results and with measured data from a real world test evacuation of a 22-storey building mainly consisting of offices and seminar rooms.

Chapter 4: Approximating Earliest Arrival Flows. In Chapter 4 we study two relaxations of the earliest arrival property in cases with multiple sinks. In a time-approximate earliest arrival flow the flow units are allowed to arrive a given factor too late. We apply a technique by Baumann and Köhler [BK07] to show that 4-time-approximate earliest arrival flows do exist in instances with a single source and multiple sinks. The existence result is complemented by an example that does not allow for a 2-approximate earliest arrival flow.

As second relaxation we study value-approximate earliest arrival flows which allows that only a certain factor of the maximum possible flow is sent at each point in time. This relaxation, which first has been used by Hoppe and Tardos [HT94], turns out to be surprisingly strong. We present a general framework that allows to answer the question if a c-value-approximate flow exists for arbitrary flow models. For the classic flow model with a single commodity we present an algorithm that computes a 2-value-approximate earliest arrival flow in a time-expanded network for instances with multiple sources *and* sinks. The algorithm can be implemented in polynomial time if all transit times are zero. For value-approximate earliest arrival flows, there exist instances that do not allow for a better approximation factor. For the case of multiple commodities we extend the algorithm such that a k-value-approximate earliest arrival flow can be computed in instances with k commodities that only use a single source and sink for *each* commodity. For instances with multiple sinks for the k commodities, we show that there is always a $2k$-value-approximate earliest arrival flow. These results are complemented by a class of networks that do not allow for a $k-1$-approximate earliest arrival flow (with only a single sink for each commodity).

Especially in the application of evacuation optimization best possible value-approximate earliest arrival flows are desired. We present a (time-expanded) linear program to compute instance optimal solutions. We then use geometrically condensed time-expanded networks introduced by Fleischer and Skutella [FS07] to compute the optimal approximation factor in polynomial time.

[1] `http://www.zet-evakuierung.de`

Chapter 5: Negative Travel Times. In Chapter 5 we investigate the implications of introducing negative transit times to flow over time problems. Even the relatively simple Maximum Flow over Time Problem becomes $\mathcal{NP}$-hard if arcs are equipped with negative travel times. However, on some instances the maximum flow can still be computed in polynomial size. We characterize instances with this property which is highly related to temporally repeated flows. For the Quickest Transshipment Problem we apply an approximation algorithm by Fleischer and Skutella [FS07] to derive a $(2+\varepsilon)$-time-approximate quickest transshipment.

As an application we study the Matching over Time Problem, an extension of the classical Matching Problem to the setting with a temporal dimension. For bipartite graphs the problem can be reduced to the Maximum Flow over Time Problem. However, instances generated by the reduction comprise negative travel times and also release dates for the arcs. For the sake of completeness we also show that the Matching over Time Problem is $\mathcal{NP}$-hard.

Chapter 6: Abstract Flows over Time. In Chapter 6 we study the existence of abstract earliest arrival flows. We generalize the notion of *lexicographically maximum flows* to the abstract setting and show that both abstract lexicographically minimum and maximum flows do exist. Repeated application of McCormick's maximum abstract flow algorithm [McC96] in growing abstract networks can be used to compute such flows. We then show that time-expanded abstract networks can be used to compute an abstract flow over time having the earliest arrival property. Therefore, we establish a correspondence between abstract lexicographically maximum flows in the time-expanded networks and and abstract earliest arrival flows. We use the framework established in Chapter 4 to show existence of 2-value approximate abstract earliest arrival flows. The result can be achieved by an application of a subroutine of the weighted abstract flow algorithm by Martens and McCormick [MM08].

Final Remarks. The thesis is mostly self contained and all necessary definitions and theorems are presented. For some established theorems the proofs are omitted if they are not of importance for the remainder of the thesis. In this cases we refer to appropriate points in the literature. We expect the reader to be familiar with basic concepts of combinatorial optimization, such as are being taught in introductory courses on discrete and linear optimization, and algorithms. In the introductory chapters we refer to general text books introducing the respective areas. For most of the mentioned techniques and definitions we also give a reference to an original publication. If there are more publications of the same results, e. g., a technical report, a conference paper or a journal publication, we only cite the most recent publication.

1.1 Preliminaries

We briefly introduce important notations and definitions that we use throughout the thesis. This allows us to use a consistent notation and also to speak clearly about the topics for which there may be several definitions in the literature. All of the covered topics are typically subject matter in undergraduate courses and we assume that they are generally known. Besides the necessary definitions we also recite some important theorems to make the thesis self contained. General introductions to the covered topics can be found in the textbooks by Schrijver [Sch03] or Korte and Vygen [KV12]. We also give additional hints to overview literature in the respective sections.

1.1.1 Complexity

In the following we will briefly review some of the basic terms of complexity that will be needed throughout the thesis. For a thorough introduction, see for example the books by Wegener [Weg05] and Papadimitriou [Pap94].

Running Time of Algorithms. A given **instance** I of an optimization problem is solved using an algorithm which is executed on the instance. We are interested in measuring the necessary resources of an algorithm. From the practical point of view, we are interested in the actual running time and the necessary memory use. These values can divert extremely depending on the specific implementation, the programming language and the machine on which the algorithm is executed. This somehow unsatisfactory situation is covered on the theoretical side by complexity theory.

An **algorithm** consists of a list of **simple operations**, which typically are small atomic operations like the addition of values or a decision for the next step that is to be performed. We assume that all these atomic operations need one time unit, and call the number of operations that an algorithm performs the **time complexity** of the algorithm which we also call **running time**. The **space complexity** of an algorithm is measured by the number of (possibly arbitrary large) values that are stored additionally to the input during its execution. These methods of counting space and time are also called **uniform costs**, which will suffice for the purpose of this thesis. As a formal model for calculating running times of algorithms, we refer to *random access machines*, which are a well known implementation of a model with uniform costs that is near enough to real computer systems for our needs. For our purposes the uniform cost model is sufficient. We just mention that operations with large numbers (such as addition) may take longer time than operations with small numbers. More advanced models that take this into account use the so-called *logarithmic costs*.

Both, the time and space complexity depend on the given instance. We thus express it in terms of the **encoding size** of an instance $\langle I \rangle$. The encoding size describes the number of bits necessary to define the instance. We denote the size for an instance $\langle I \rangle$ with $|\langle I \rangle|$. The actual running time for a given instance can vary depending on its structure. A function $f(n) : \mathbb{N} \to \mathbb{N}$ is called a **runtime function** for an algorithm, if for all instances $\langle I \rangle$ with $|\langle I \rangle| = n$ the running time of the algorithm on the input is bounded by $f(n)$. As instances and their specifics are typically not known in advance we are interested in the **worst-case running time**. We denote the set of all valid instances for a problem by $\mathcal{I}$ and define the worst-case running time as $\max\{\min\{f(|\langle I \rangle|) \mid f \text{ runtime function for } \langle I \rangle\} \mid \langle I \rangle \in \mathcal{I}\}$. We say that an algorithm has a **polynomial** running time, if it has a runtime function $f(n)$ that is bounded by a polynomial in n, otherwise we call the running time **super-polynomial**. In most cases, we consider an algorithm to be efficient if it has a polynomial worst case running time. We are not interested in knowing the exact running time but its *order* which is measured by means of $\mathcal{O}$-notation. For a function $g : \mathbb{N} \to \mathbb{N}$, we define a set $\mathcal{O}(g) := \{f : \mathbb{N} \to \mathbb{N} \mid \exists c \in \mathbb{R}^+ : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 \in \mathbb{N} : f(n) \leq cg(n)\}$, which contains all functions that grow at most with the same order as g. For brevity, we sometimes omit logarithmic terms and therefore define $\tilde{\mathcal{O}}(g) := \{f : \mathbb{N} \to \mathbb{N} \mid f \in \mathcal{O}(g(n) \cdot \log_k(g(n))\}$.

For a problem let x be the maximum value in the input $\langle I \rangle$. We say it is a **number problem** if x cannot be bounded by $|\langle I \rangle|$. Most problems that we discuss throughout the thesis are number problems. We call an algorithm for a number problem **pseudo-polynomial**, if its worst-case running time is bounded by a polynomial in both $|\langle I \rangle|$ and x. The effect of pseudo-polynomial algorithms is that its running time strongly depends on the size of the numbers in the input.

Hardness of Problems. A **decision problem** is a problem which only has two valid solutions that we call `yes` and `no`. Accordingly, the instances to such problems are called **`yes`-instance** and **`no`-instance**. Many optimization problems naturally define a decision variant. If the optimization goal is to minimize (or maximize) a given value, the corresponding decision problem answers the question whether a solution of a given value exists. The complexity class $\mathcal{P}$ is the set of all decision problems for which an algorithm with polynomial time complexity exists. The class $\mathcal{NP}$ consists of all decision problems for which, given a certificate of polynomial size, it can be decided for all instances in polynomial time if the certificate describes a correct solution. Thus, we have immediately $\mathcal{P} \subseteq \mathcal{NP}$. A decision problem A can be **polynomially reduced** to a problem B if there exists an algorithm that transforms an instance $\langle I_A \rangle$ for A into an instance $\langle I_B \rangle$ for problem B in polynomial time such that any algorithm for B accepts $\langle I_B \rangle$ if and only if $\langle I_A \rangle$ was a `yes`-instance for A. We say, a problem A is $\mathcal{NP}$**-hard**, if every problem in $\mathcal{NP}$ can be polynomially reduced to A, e. g., $B \leq_p A$ for all $B \in \mathcal{NP}$. If also $A \in \mathcal{NP}$, we say that A is $\mathcal{NP}$**-complete**. Thus, if a polynomial algorithm for any $\mathcal{NP}$-hard problem is found $\mathcal{P} = \mathcal{NP}$ would follow, which is commonly not believed to be true. Since the discovery of the first $\mathcal{NP}$-complete problem, a lot of problems have been shown to be $\mathcal{NP}$ complete, a lot of them can be found in the textbook by Garey and Johnson [GJ79]. Throughout this thesis we will assume that $\mathcal{P} \neq \mathcal{NP}$ and sometimes see results that only follow under this assumption.

If a pseudo-polynomial algorithm for an $\mathcal{NP}$-hard problem exists, we also call it **weakly** $\mathcal{NP}$**-hard**. On the other hand, if a problem remains $\mathcal{NP}$-hard even if all numbers in an instance are bounded by $|\langle I \rangle|$, we say it is **strongly** $\mathcal{NP}$**-hard**.

An example for a weakly $\mathcal{NP}$-hard problem is Partition, one of the first 21 $\mathcal{NP}$-hard problems described by Karp [Kar72]. We use Partition throughout the thesis to show $\mathcal{NP}$-hardness for several other problems.

Problem: Partition

Instance: $a_1, a_2, \ldots, a_n \in \mathbb{N}$ with $A := \sum_{i=1}^{n} a_i$ even.

Task: Find an index set $I \subseteq \{1, 2, \ldots, n\}$ such that

$$\sum_{i \in I} a_i = \sum_{j \notin I} a_j = \frac{A}{2}.$$

Approximation Algorithms. To handle the fact, that (under the assumption of $\mathcal{P} \neq \mathcal{NP}$) it is impossible to find efficient algorithms for $\mathcal{NP}$-hard problems, approximation algorithms have been developed. The main goal here is to find a polynomial algorithm that probably does not find the optimal solution but finds a solution that is "as good as necessary". Extensive coverage of approximation algorithms and several important techniques are presented in the books by Hochbaum [Hoc97], Vazirani [Vaz01] and Williamson and Shmoys [WS11], whose notation we use in the following.

Let $c > 0$. A c-**approximation algorithm** for an optimization problem is a polynomial algorithm that computes a solution whose value lies within a factor of c of the value OPT of the optimal value. If the problem is a maximization problem, the solution of the algorithm has at least a value of OPT/c. For minimization problems, the solution has at most a value of $c \cdot OPT$.

While it is nice to find a constant c such there exists a c-approximation algorithm for a hard combinatorial optimization problem, it would be even better to get arbitrarily close to the optimal solution. In this case we could compute a solution that is as good as we wish it to be. The best we can hope for is a polynomial approximation scheme that gives a $(1+\varepsilon)$-approximation algorithm for any positive ε. A **polynomial-time approximation scheme** (**PTAS**) is a family of algorithms $\{A_\varepsilon\}$, where for each $\varepsilon > 0$ there is an algorithm A_ε which is a $(1+\varepsilon)$-approximation algorithm. While a PTAS already gives us basically what we want, namely an arbitrarily good optimization of the optimal solution, we still might not be satisfied with the running time for very small ε. This will be covered by fully polynomial-time approximation schemes. A **fully polynomial-time approximation scheme** (**FPTAS**) is a family of algorithms $\{A_\varepsilon\}$, where for each $\varepsilon > 0$ there is an algorithm A_ε which is a $(1+\varepsilon)$-approximation algorithm and whose running time is polynomial in the input size and in $\frac{1}{\varepsilon}$.

Parametric Search. A common task in combinatorial optimization is to find a minimal value for a parameter λ, such that a problem remains feasible. In the context of network flows over time, the parameter may be the time horizon. The task is then to find the minimal time horizon that allows to ship a given amount of goods from starting points to end points. If we know upper and lower bounds λ_{max} and λ_{min}, we can use binary search to detect the optimal value. However, this method requires $\mathcal{O}(\log(\lambda_{max} - \lambda_{min}))$ calls of the decision algorithm as subroutine and usually does not guarantee strongly polynomial running time as the input most likely relies on some numbers in the input. Megiddo [Meg79] presents a **parametric search** framework that improves this procedure by bounding the number of calls of the decision algorithm.

We briefly describe the method. The basic idea of Megiddo's parametric search framework is to replace every variable in an algorithm by linear functions (with λ being the parameter of the function). Actually, we are interested in the unknown optimal value λ^*. By calculating with linear functions instead of simple real valued variables, we can execute the algorithm for many (and in the beginning: all) possible values. When the execution is started, we have the possible interval $[\lambda_{min}, \lambda_{max}]$ (which totally may be $]-\infty, \infty[$) in which λ^* can lie. During the execution, we shrink the possible interval until we know the actual value λ^*. It is possible to add linear functions, however decisions within the algorithm create some problems.

We define the search version of a given algorithm by replacing each variable i within the algorithm by a linear function $a_i + \lambda b_i$. During the execution, the algorithm performs additions and comparisons based on the new linear function variables and we have to define its behaviour. The sum of two variables i and j is computed as $a_i + a_j + \lambda(b_i + b_j)$. Whenever the modified algorithm has to perform a comparison, we gain information on the optimal value λ^*. Basing on the new information we shrink the possible interval. Let i and j be two variables and the algorithm branches based on a comparison $i < j$ using the two corresponding functions $a_i + \lambda b_i$ and $a_j + \lambda b_j$. The two distinct linear functions cross at most once; let λ' be the value for λ with $a_i + \lambda' b_i = a_j + \lambda' b_j$. Then for all $\lambda \leq \lambda'$ and $\lambda \geq \lambda'$ the comparison $a_i + \lambda b_i < a_j + \lambda b_j$ gives the same result. Thus, we only need to know in which part of the interval (left or right of λ') the optimal value λ^* lies. To get this information, we execute the original algorithm (with normal variables) with parameter λ' and can therefore decide in which of the two intervals $[\lambda_{min}, \lambda']$, or $[\lambda', \lambda_{max}]$ the optimal value λ^* lies. We update the bounds and continue the algorithm with the decision $i < j$.

The overhead for the algorithm with linear functions is constant for storing the variables and results of additions. However, whenever a decision occurs, we have to execute the original algorithm again. But this can occur only once for each of the original algorithm's operations which means that

the runtime is basically squared. The method can only be applied if additions, comparisons and function evaluations are the only used operations, which is true for many algorithms.

Theorem 1.1 (parametric search). *Let $\mathcal{A}_{dec}$ be an algorithm for the decision variant of a given problem. Let the input for $\mathcal{A}_{dec}$ include a parameter λ and the question is, if the problem is feasible for λ. The optimization variant of the problem asks for the value λ^*, such that $\mathcal{A}_{dec}$ returns `no` for all $\lambda < \lambda^*$ and `yes` for all $\lambda \geq \lambda^*$.*

Let p be the number of comparisons performed by $\mathcal{A}_{dec}$ and q the number of additions. If $\mathcal{A}_{dec}$ only uses additions, comparisons and function evaluations and has a strongly polynomial running time, then we can find the optimal value λ^* in strongly polynomial running time $\mathcal{O}\big(p(p+q)R\big)$, where R is the running time of $\mathcal{A}_{dec}$.

1.1.2 (Integer) Linear programming

One of the most important and fruitful tools in the area of combinatorial optimization is linear programming. It is possible to formulate many problems in terms of so called (integer) linear programs, such that either the linear program allows to find an optimal integral solution or a relaxed solution can be used to generate good approximation algorithms.

Problem:	Linear Programming
Instance:	A matrix $A \in \mathbb{Q}^{m \times n}$, a cost vector $c \in \mathbb{Q}^m$ and $b \in \mathbb{Q}^n$
Task:	Find a vector $x^* \in \mathbb{Q}^m$ in $P := \{x \in \mathbb{Q}^m \mid Ax \leq b, x \geq 0\}$ that maximizes $c^T x$, or either decide that the maximum s infinite, or P is empty.

We denote the input given by the matrix and the vectors as **linear program** (**LP**). For an **integer program** (**IP**), we additionally require integrality, i. e., $x \in \mathbb{Z}^m$.. Linear and integer programs are typically not described simply as matrix, but as a list of constraints of some variables. A simple example for linear programs is given with the linear program (PF) for maximum flow path formulation in Section 1.2 later in this chapter.

Linear programs can be solved efficiently in practice using the simplex method introduced by Dantzig already 1951 [Dan51b]. However, most of the existing pivoting rules have been shown to have exponential worst case running time [Zad73] and no general results showing polynomial running time are known. The first polynomial-time algorithm for solving linear programs has been found by Hačijan [Hač79]. The algorithm is based on the so called *ellipsoid method* and is not useful in practice. However, the algorithm is of importance from the theoretical point of view. Later Karmarkar noticed that the *interior points method* can also solve a Linear Programming Problem in polynomial time [Kar84]. Modern implementations of the interior points method are also practically efficient and can compete with the simplex algorithm. Integer linear programming is $\mathcal{NP}$-complete [Kar72] and all $\mathcal{NP}$-complete problems can be formulated as integer program. If there is no immediate representation the problems can be reduced to the satisfiability problem which has an immediate representation as integer program.

There exist several commercial and non-commercial implementations to solve both, linear and integer linear programs. Well known algorithmic packages are e. g., CPLEX [IBM14], Gurobi [Gur14], GLPK [GLP14] and SCIP [SCI14].

Duality. Maximization and minimization of linear programs are strongly related by means of duality. Any linear maximization problem can be expressed as a minimization problem with equal optimal value, if the optima are finite. Then the duality theorem of linear programming, first proven by von Neumann [vN47] and by Gale et al. [GKT51], states, that optimizing of a linear program and optimizing the corresponding dual is essentially the same.

Theorem 1.2 (Duality Theorem). *Let $A \in \mathbb{Q}^{m\times n}$ be a matrix and $b \in \mathbb{Q}^m, c \in \mathbb{Q}^n$ vectors. Then*

$$\max\left\{c^T x \mid Ax \leq b, x \geq 0\right\} = \min\left\{y^T b \mid y \geq 0, y^T A = c^T\right\},$$

if the values are finite and both programs are feasible.

In the above theorem, we call the minimization problem the **dual program** (**DP**) and y is the **dual solution**. Notice, that if the original problem was already a minimization problem, the corresponding maximization problem is the dual. The correlation between the two problems is given by the correspondence between their respective constraints and variables. The matrix A occurs in transposed form in the dual problem. As rows in the matrix correspond to constraints, while columns are associated with variables, this relation is reversed in the dual problem. Via the process of **dualization**, for any LP the dual can be computed by introduction of **dual variables** for the constraints of the original problem while each variable introduces a constraint. Also, the cost vector of the LP becomes the right-hand side of the DP. Depending on the form of inequalities, the duality theorem can have several forms. For a detailed introduction see e. g., [Sch03].

Equivalence of Optimization and Separation. The aforementioned ellipsoid method may not be useful for solving linear programs in practice, but allows to derive theoretical insights in a problem's complexity. We will briefly introduce the method, which basically is referred to as *equivalence of separation and optimization*. The results where first established by Grötschel, Lovász and Schrijver [GLS81] and independently discovered by Karp and Papadimitriou [KP82] and also by Padberg and Rao [PR81]. For a detailed introduction into these concepts we refer to the book by Grötschel, Lovász and Schrijver [GLS88]. For each LP we define the corresponding **Separation Problem** to be the problem that checks, if a solution is feasible, and returns a violated constraint, if not.

Problem:	Separation
Instance:	A linear program, $x \in \mathbb{Q}^m$.
Task:	Decide, whether x is feasible for the LP, or find a violated constraint.

Theorem 1.3 (Equivalence of Optimization and Separation). *The optimum solution for a linear program can be found in polynomial time if and only if the corresponding separation problem can be solved in polynomial time.*

The main result of Grötschel, Lovász and Schrijver [GLS88] is the above theorem, stating, that the solution of an LP can be found efficiently, if the separation problem can be solved efficiently. Their proof relies on the ellipsoid method and is therefore not applicable in practice. The crucial property of the ellipsoid is, that it may be used without looking at all constraints, which dramatically reduces the runtime, if a linear program consists of exponentially many constraints. Thus, their approach gives raise to efficient implementations of the **cutting plane method** developed by Gomory [Gom58; Gom60] and Chvátal [Chv73]. Assume an LP with exponentially many constraints. Obviously we do not want to store the problem by its matrix description. Instead, we hope it is enough to store only "few" constraints to compute an optimal solution. We start with an empty LP without any constraints and find an optimal (probably unbounded) solution. We then use the separation problem to find a violated constraint (in polynomial time). If this constraint is added to the LP, the original solution is not valid any more and we continue solving a new solution of the now extended LP. We repeat the process until we find a solution of our LP that does not violate any of the constraints, i. e., the separation problem returns that the current solution is feasible. By **column generation** we call the process of solving a problem with exponentially many variables. Observe, that instead of using the cutting plane method we can simply dualize the problem and use column generation.

Total Dual Integrality. Linear optimization is a powerful tool in practice. However, if integer solutions are desired, it is not guaranteed that an optimum is integral, even if the input is integral. A concept dealing with this problems is due to early work by Hoffman [Hof74] and Edmonds and Giles [EG77]. A given linear program specified by inequalities $Ax \leq b$ and $x \geq 0$ with $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Z}^m$ is **totally dual integral**, if the corresponding dual program $\min\{y^T b \mid y \geq 0, y^T A = c^T\}$ has an integer optimal solution $y \in \mathbb{Z}^m$ for every cost vector $c \in \mathbb{Z}^n$, if it is finite. Such a solution also implies an integral optimum solution for the linear program $\max\{c^T x \mid Ax \leq b, x \geq 0\}$. For further details see e. g., the textbook by Schrijver [Sch98]. Notice that totally dual integral systems of inequalities are very interesting, because in this case optimizing the linear program also gives the optimum of the IP, which makes these programs a polynomial special case.

1.1.3 Graphs

Directed Graphs. A **directed graph** or just **graph** is a pair $G = (V, E)$ where V is a finite set of **vertices** or **nodes** and E is a family of ordered pairs in $V \times V$. The elements of E are called **directed edges** or **arcs**. The edges form a family and it is possible that a graph contains multiple copies of the same **parallel** edges. Edges of the form (v, v) for any node v are denoted as **loops**. If not stated otherwise, we typically denote the number of nodes with $n := |V|$ and the number of arcs by $m := |E|$.

Let $e = (u, v)$ be an arc between two nodes $u, v \in V$. The vertices that form the arc are addressed as **tail** and **head** with the notation $\text{tail}(e) := u$ and $\text{head}(e) := v$. The arc e is an **outgoing** arc of u and and **incoming** arc of v. For a vertex v the set of incoming arcs is defined as $\delta^-(v)$ and the set of outgoing arcs as $\delta^+(v)$. This notation is extended to subsets $U \subseteq V$ by defining $\delta^+(U) := \bigcup_{v \in U} \{e \in \delta^+(v) \mid \text{head}(e) \notin U\}$ and $\delta^-(U) := \bigcup_{v \in U} \{e \in \delta^-(v) \mid \text{head}(e) \notin U\}$ to be the set of arcs entering and leaving U, respectively.

Undirected Graphs. An **undirected graph** is a pair $G = (V, E)$ where V is a finite set of **vertices** and E is a family of **(undirected) edges** $e = \{v, w\} \subseteq V$ that are subsets of V of cardinality 2.

Series Parallel Graphs. A **series-parallel** graph is a graph having two distinguished vertices s and t hat can be composed according to he following recursive definition. An edge (s,t) is a series-parallel graph. Let G_1 and G_2 be two series parallel graphs with nodes s_1, t_1 and s_2, t_2, respectively. The *series-composition* defines a series-parallel graph G' by identifying t_1 and s_2. The *parallel-composition* defines a series-parallel graph G' by identifying s_1 with s_2 and t_1 with t_2. Series-parallel graphs build an important subclass of graphs that are used to provide either specialized and therefore faster or simpler algorithms, or show that a problem is hard already on these simple instances.

Paths and Cycles. A **sequence** is a list of edges $(e_1, e_2, \dots, e_k)$ such that the edges are connected, i. e., $\text{head}(e_i) = \text{tail}(e_{i+1})$ for $i = 1, 2, \dots, k-1$. The sequence is called a **(directed) path**, if $e_i \neq e_j$ for all $i, j \in \{1, 2, \dots, k\}$. If also each vertex on a path is visited once, e. g., $\text{head}(e_i) \neq \text{head}(e_j)$ and for all $i \neq j$ and $\text{head}(e_k) \neq \text{tail}(e_1)$, the path is **simple**. An u-v-path is a path with $u = \text{tail}(e_1)$ and $v = \text{head}(e_k)$. A **cycle** is a path whose last arc ends at the first arc, e. g., $\text{head}(e_k) = \text{tail}(e_1)$. The set of u-v-paths in a graph are denoted as $\mathscr{P}^{uv}$. If the nodes are fixed we just refer to the set of paths as $\mathscr{P}$.

Shortest Paths. We denote the minimum distance between two nodes v and w by $\text{dist}(v,w)$. If the arc lengths are non-negative, the distance and also the actual shortest paths between nodes v and w can be computed by Dijkstra's algorithm [Dij59] or any of the countless improvements. Practically efficient implementations using heap data structures have a running time in $\mathcal{O}(n \cdot \log(m))$. For the case of possibly negative arc length (but without a negative cycle), shortest paths can be computed using the Moore-Bellman-Ford-Algorithm[Bel58; For56; Moo59] with a worst-case running time in $\mathcal{O}(n \cdot m)$.

Cuts. A subset $C \subseteq E$ is called a (forward) **cut**, if a subset $U \subseteq V$ exists, such that $\delta^+(U) = C$. Any given subset $U \subseteq V$ defines a cut $C(U, V \setminus U) \coloneqq \delta^+(U)$. If there are two nodes $s \in U$ and $t \in V \setminus U$, the cut is an s-t-cut. If the arcs have assigned edge costs $c_e \in \mathbb{R}$, the value of a cut is defined to be $\sum_{e \in C} c_e$.

1.2 Classical Network Flows

In this section we introduce network flows which we will also call static or classical network flows. The first textbook covering network flows was the seminal book by Ford and Fulkerson from 1962 [FF62]. This book also already introduced network flows over time, which are defined in Chapter 2. After their introduction in the 1950's a lot of research on the topic of static network flows has been done. The textbook by Ahuja, Magnanti, and Orlin [AMO93] covers a vast part of the topic of static network flows and is an excellent resource.

History of Network Flows. The foundations of network flows, which the research in this thesis is based upon, were laid in the 1950's. Although, as we will see, the starting point was rather military driven, the successive research was most beneficial and the current applications are legion. Initially, Harris and Ross [HR55] posed the question, how the Russian railway system could be "cut" between the east and western parts in a cheapest way. Subsequently, Ford and Fulkerson [FF56] showed that the value of a minimum cut and the value of a maximum flow is equal and thus, the computation of

the cut reduces to the computation of a maximum flow. This was also observed independently by Elias, Feinstein, and Shannon [EFS56]. The first proof was non-constructive and does not allow to generate a direct algorithm from it, a fact that we will elaborate on in Chapter 6. The corresponding algorithm was then proposed in [FF57]. Dantzig and Fulkerson were the first to observe, that the network flow theory is a special case of general linear programming duality [DF56], and therefore variants of the simplex algorithm can be used to solve it [FD55]. All this research allowed Ford and Fulkerson to answer the question on the Russian railway network. However, the network could not be built the way we would do it, because the resulting instance would be too big to solve. Harris and Ross [HR55] describe a technique that allows to aggregate parts of a network to reduce network size. Finally, they came up with the instance depicted in Figure 1.1, which then was solved by using Boldyreff's heuristic [Bol55], because the algorithm of Ford and Fulkerson was considered to be too computationally expensive. Interestingly, the same algorithm is today considered one of the simplest known algorithm that is also reasonably fast on small to medium size instances. A comprehensive overview over the beginning of research on the topic of network flows is due to Schrijver [Sch02]. We will now define network flows in the modern and concise way as we use them throughout the thesis.

Edge Flows. A **(static) network** $\mathcal{N} = (G = (V, E), u, S^+, S^-)$ consist of a graph $G = (V, E)$, capacities u_e on the arcs and disjoint sets $S^+ \subset V$ and $S^- \subset V$ of **sources** and **sinks** , respectively. In the case of a single sink and single source we also write it as $\mathcal{N} = (G, u, s, t)$. We also denote the nodes $v \in V \setminus S^+ \cup S^-$ as **intermediate** nodes.

A **(static) flow** in a network is an assignment $f : E \to \mathbb{R}^+$ satisfying **capacity constraints**

$$f(e) \leq u_e$$

for each arc $e \in E$ and **flow conservation**

$$\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$$

on intermediate nodes $v \in V \setminus S^+ \cup S^-$. A network flow that also satisfies flow conservation at the terminals $S^+ \cup S^-$ is called **circulation**. We call $|f| := \sum_{e \in \delta^+(s)} f(e)$ the **value** of f. The most famous network flow problem is the (edge based) maximum flow problem, which has been studied by Ford and Fulkerson extensively [FF62].

Problem:	Maximum Network Flow
Instance:	A network $\mathcal{N} = (G, u, s, t)$.
Task:	Find a network flow f feasible in $\mathcal{N}$ with maximum value $\|f\|$.

Theorem 1.4 (Max-Flow=Min-Cut). *Let $\mathcal{N} = (G, u, s, t)$ be a static network. Then, the maximum flow value $|f|$ equals the value of a minimum feasible cut separating s and t in $\mathcal{N}$.*

This result, which is a special case of the duality Theorem 1.2, has been extended to several other types of network flow problems, including dynamic network flows and abstract flows, as we see in

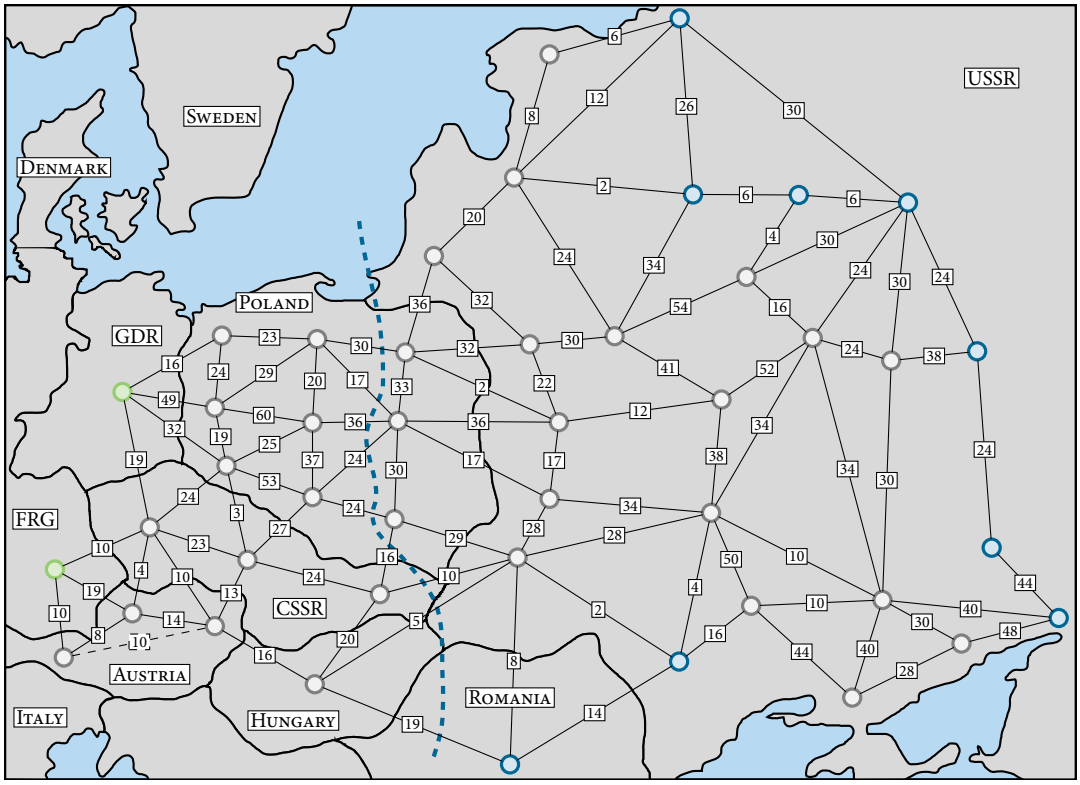


Figure 1.1: The Soviet rail network used as motivational example for the start of network flow research by Ford and Fulkerson. The dashed line represents the minimum cut.

Chapter 2 and Chapter 6, respectively. It is also a very important type of a total dual integral system, as Hoffman [Hof74] showed.

Ford and Fulkerson [FF56], and independently Elias, Feinstein, and Shannon [EFS56], first established a so called Max-Flow=Min-Cut-Theorem, which states that the value of a maximum flow equals the value of a minimum cut. It is a special case of the duality theorem of linear programming, and the linear programs for maximum flow and minimum cut are in fact dual linear programs. A related problem occurs if we do not want to send as much flow as possible but if a certain amount of given supplies and demands should be satisfied.

Problem:	Transshipment
Instance:	A network $\mathcal{N} = (G = (V, E), u, S^+, S^-)$, supplies and demands b_v for sources and sinks.
Task:	Find a network flow f feasible in $\mathcal{N}$ that satisfies the given supplies and demands, i. e., $\sum_{e \in \delta^+(s)} f(e) = b_s$ and $\sum_{e \in \delta^-(t)} f(e) = -b_t$ holds for sources s and sinks t, respectively.

The Extended Network. Traditionally the maximum network flow problem is defined on networks containing a single source and a single sink (and without incoming arcs to the source and outgoing arcs of the sink). This is no limitation because instances with multiple sources can be reduced to the special case by using the so called **extended network** which contains two more nodes that we call **super source** and **super sink**. As the name already indicates these two nodes will serve as single source and single sink in the new network.

Definition 1.5. (Extended Network) Let $\mathcal{N} = (G = (V, E), u, S^+, S^-)$ be a classical network. We define the **extended network** $\mathcal{N}^* = (G^*, u^*, s^*, t^*)$ by introducing two additional nodes, the **super source** s^* and the **super sink** t^* on the set $V^* := V \cup \{s^*\} \cup \{t^*\}$ of nodes and the edges $E^* := E \cup \bigcup_{s \in S^+} \{(s^*, s)\} \bigcup_{t \in S^-} \{(t, t^*)\}$. The capacities are extended as $u_{(s^*,s)} := \infty$ for sources and $u_{(t,t^*)} := \infty$ for sinks.

If additional supplies and demands are given for sources and sinks we define the capacities as $u_{(s^*,s)} := b_s$ and $u_{(t,t^*)} := b_t$ for sources and sinks. In this case the new supplies and demands are defined to be $b_s = b_t = 0$ for sources $s \in S^+$ and sinks $S^- \in S^-$, respectively. The supplies for the super source are defined as $b_{s^*} := \sum_{s \in S^+} := b_s$ and the demands at the super sink are defined as $b_{t^*} := \sum_{t \in S^-} b_t$. ◁

In the case of the Maximum Flow Problem, the capacity of the new arc is unlimited as we try to maximize the flow that is sent from sources to sinks. For restricted problems, e. g., the Transshipment Problem, the amount of flow starting from sources is limited, therefore the capacity of the arcs incoming to the original sources are limited, the same applies for the sinks.

Observation 1.6 ([FF62]). *Let $\mathcal{N} = (G = (V, E), u, S^+, S^-)$ be a network. Then the value of a maximum flow in $\mathcal{N}$ and the extended network $\mathcal{N}^*$ are the same. If supplies and demands $b : V \to \mathbb{R}$ are given, a transshipment exits in $\mathcal{N}$ if and only if it also exists in $\mathcal{N}^*$.*

Proof. Let f^* be a flow in $\mathcal{N}^*$. The flow respects capacities on all arcs, and flow conservation holds on all intermediate nodes. Therefore the flow $f(e) := f^*(e)$ for $e \in E$, also respects capacities and flow conservation on intermediate nodes. Flow conservation is only violated at sources and sinks, which is allowed.

If we have supplies and demands at the sources and sinks, we have to verify that f respects them. The flow conservation is at most violated by the flow sent on arcs (s^*, s), and (t, t^*) for sources s and sinks t. Therefore, we bound the capacity on these arcs by b_s and b_t, respectively. □

Path Flows. Network flows can also be defined in a path based form. This alternative definition allows for a very simple linear program because flow conservation is automatically achieved on a path, but the size of the LP usually increases exponentially due to the huge amount of paths in a network.

Definition 1.7 (Path Flow). Let $\mathcal{N} = (G = (V, E), u, S^+, S^-)$ be a network and $\mathscr{P}$ the set of all s-t-paths for any pair of sources $s \in S^+$ and $t \in S^-$. The subset of paths using a given arc e is defined as $\mathscr{P}_e := \{P \in \mathscr{P} \mid e \in P\} \subseteq \mathscr{P}$. A **path flow** is an assignment $x : \mathscr{P} \to \mathbb{R}_{\geq 0}$ that adheres to **capacity constraints**

$$\sum_{P \in \mathscr{P}_e} x_P \leq u_e.$$

The **value** of x is defined as $|x| = \sum_{P \in \mathscr{P}} x_P$. ◁

Notice, that due to the path formulation, we do not explicitly have to consider flow conservation any more. Any path flow immediately gives an edge flow, by just summing up the flow values on all paths going through an edge. The other direction is not as clear, but there is also a path flow belonging to each edge flow.

Path Decomposition. Edge flows and path flows are strongly related and in fact, are identical in the sense that each edge flow can be expressed by an equivalent path flow and vice versa: Any path flow defines an edge flow and the flow value on an edge can be set to the sum of the flow on all paths using this edge. A **path decomposition** of an edge flow f is a set of paths $\mathscr{P}' \subset \mathscr{P}$ of all source-sink paths $\mathscr{P}$ together with values x_P for paths $P \in \mathscr{P}'$ such that x perceived as path flow (by setting $x_P = 0$ for all $P \in \mathscr{P} \setminus \mathscr{P}'$) has the same value $|x| = |f|$ and the flow values on arcs of the induced flow is at most the flow value of the original flow on any arc. The last constraint originates from the fact that flow can be sent along cycles without changing the value of an edge flow.

Theorem 1.8 (Path Decomposition Theorem, [Gal58; FF62]). *There exists a polynomial size path decomposition of any edge flow f.*

We will only review a sketch of the proof. The path decomposition can easily be constructed by simply taking an arbitrary edge (that has positive flow on it) and (seeing it as a path) extending it in the front and the end by adding more flow carrying edges until source and sink are reached. It may be the case, that a cycle occurs. In this case, the flow on the cycle can be reduced and the process can be continued. Thus, we see that edge and path flows can be converted into each other. However, the Maximum Flow Problem in the path based formulation cannot be solved in polynomial time easily without using a separation oracle because of the possibly exponential number of paths.

The task to find a maximum flow as path flow can be formulated as a linear program. We use variables x_P for each path to denote how much flow is sent along the path and one constraint to ensure that the flow adheres to the capacity constraints.

$$\begin{aligned} \max \quad & \sum_{P \in \mathscr{P}} x_P, && && \text{(PF)} \\ \text{s.t.} \quad & \sum_{P \in \mathscr{P}_e} x_P && \leq u_e \quad && \text{for all } e \in E, \\ & x_P && \geq 0 \quad && \text{for all } P \in \mathscr{P}. \end{aligned}$$

By introducing dual variables y_e for each edge and dualization of the path based maximum flow problem we get an LP formulation of the minimum cut problem.

$$\begin{aligned} \min \quad & \sum_{e \in E} y_e, && && \text{(MC)} \\ \text{s.t.} \quad & \sum_{e \in P} y_e && \geq 1 \quad && \text{for all } P \in \mathscr{P}, \\ & y_e && \geq 0 \quad && \text{for all } e \in E. \end{aligned}$$

Problem:	Minimum Cut
Instance:	A network $\mathcal{N} = (G, u, s, t)$ with the set of s-t-paths $\mathscr{P}$.
Task:	Find a cut $C \subseteq E$ minimizing the dual program (MC).

The two linear programs (PF) and (MC) are a first and very simple example of dualization. The primal program describing the path flow, has one variable x_P for every path and one constraint for each edge $e \in E$. Consequently, the dual program has one variable y_e for each edge and one constraint for each path. It is easy to verify, that the path formulation of the minimum cut problem equals the above definition. Let C be an s-t-cut. By definition there is no s-t-path any more if the corresponding edges are removed. If we assign a value of $y_e = 1$ for all $e \in C$, this gives a solution of (MC). Because the system is totally dual integral [Sch84], any solution also specifies a cut by using the arcs whose variables are set to 1.

Both problems are easy to write down, but they are huge by means of complexity. The path based maximum flow problem has exponential many variables and the dual has exponential many constraints. Because of Theorem 1.8 we know, that there is an optimal solution that only has a polynomial number number of paths that actually carry flow. Thus, the path flow can be computed in polynomial time using column generation.

Solving the Maximum Flow Problem. Despite the usefulness of the path flow in theory and also on the practical side of solving dynamic network flow problems, practically efficient methods of solving static network flow problems are almost all based on edge flows. Basically, the known network flow algorithms can be divided into (mostly path based) augmentation algorithms and preflow push algorithms. These two approaches also reflect the duality of the linear programs. While the augmentation algorithms always maintain a feasible, not optimal flow, the latter class of algorithms maintains a primary infeasible solution and reduces the infeasibility.

Central to most network flow algorithms is the concept of a residual network. This construction allows algorithms to take back flow that has been sent in earlier iterations of an algorithm. The idea is to double all arcs and let the copy account for the network flow, that has been sent already on an arc.

Definition 1.9 (Residual Graph). Let $\mathcal{N} = (G, u, s, t)$ be a network and f be an edge based flow.

For an arc $e = (v, w) \in E$ we define $\overleftarrow{e} := (w, v)$ to be the **reverse arc** of e. With the set of residual edges $\overleftarrow{E} := E \cup \{\overleftarrow{e} \mid e \in E\}$ we define the **residual graph** $\overleftrightarrow{G} = (V, \overleftrightarrow{E})$ as the set of reverse arcs.

The **residual capacity** with respect to the flow f is defined as

$$u_f(e) := \begin{cases} u_e - f(e) & \text{if } e \in E, \\ f(e) & \text{if } e \in \overleftarrow{E}. \end{cases}$$

◁

Most algorithms to compute a maximum flow are executed on a residual network. The only exceptions are basically McCormick's abstract flow algorithm [McC96] and solving the linear program directly. McCormick's algorithm has been applied to classical networks in [Kap09]. Most algorithms belong to either the category of augmenting path algorithms, like the original algorithm by Ford and

Fulkerson, or to the class of push-relabel algorithms which were the fastest (both, in theory and practice) known algorithms for a long time. We will shortly give an overview over the development in algorithms. Only recently, Orlin [Orl13] answered a long-standing open question and gave an algorithm with a running time in $\mathcal{O}(nm)$ which even improves to $\mathcal{O}\left(n^2/\log(n)\right)$ if the number of arcs is in $\mathcal{O}(n)$.

Evolution of Flow Augmenting Algorithms. The first maximum flow algorithm for directed networks is the famous algorithm by Ford and Fulkerson [FF62], which essentially describes a framework to solve the Maximum Flow Problem. The algorithm consists of a loop that augments flow on an arbitrary path in the residual network, until no residual s-t-path can be found any more. When the algorithm terminates, the current flow in the network is maximal. The runtime of the algorithm strongly depends on the choice how an augmenting path is selected. Already Ford and Fulkerson noticed [FF62], that the algorithm does not necessarily terminate if the capacities on the arcs are not integral but irrational numbers and smaller examples have been provided by Zwick [Zwi95]. The general running time is bounded by $\mathcal{O}(U \cdot n \cdot m)$ with n and m being the number of nodes and arcs, respectively and U being the highest arc capacity, and thus may be pseudo-polynomial. Edmonds and Karp [EK72], and independently Dinic [Din70] found that the problem of the running time is due to unwisely chosen paths. If the augmenting paths are not chosen arbitrarily, but shortest, the running time becomes strictly polynomial and reduces to $\mathcal{O}(nm^2)$. The augmenting path algorithms can be improved by augmenting not only on paths but by using *flows*, especially so-called *blocking flows*. This led to an $\mathcal{O}(n^2m)$ algorithm by Dinic [Din70] that was improved to have worst-case running time of $\mathcal{O}(n^3)$ by Karzanov [Kar74; MKM78].

Push-Relabel Algorithms. The next improvements were obtained by the introduction of the push-relabel (that are sometimes also referred to as preflow-push) algorithms by Goldberg and Tarjan [GT88]. Their generic algorithm has been improved by introducing several new selection rules and also sophisticated data structures. Improvements led to the algorithm of King, Rao and Tarjan [KRT94] with a running time of $\mathcal{O}\left(nm \cdot \log_{m/n\log(n)}(n)\right)$. Further improvements led to an algorithm with worst-case running time of $\mathcal{O}\left(\min\left\{n^{2/3}m, m^{1/2}\right\} \cdot \log\left(n^2/m\right) \cdot \log(U)\right)$, developed by Goldberg and Rao [GR98].

Many of the (at their time) best results are based on concepts, that are hard to implement, or are not efficient in practice. However, in parallel to the development of better theoretical results, fast practical implementations and heuristics have been developed. Cherkassky and Goldberg [CG97] presented an efficient implementation of the push-relabel algorithm which is known to be one of the fastest implementations of the push-relabel algorithm. They introduced several heuristics that improve the running time dramatically on practical instances. However the theoretical worst-case running time remains the same. Later, Goldberg [Gol08] found an improved maximum flow algorithm which is comparably fast in general and faster on some instances.

Minimum Cost Flows. We consider two generalized variants using costs on arcs. If $c : E \to \mathbb{R}$ is a cost function, the cost of a flow f is

$$c(f) := \sum_{e \in E} c_e \cdot f_e.$$

Based on the costs, we now define minimum cost problems.

Problem:	Minimum Cost Circulation
Instance:	A graph $G = (V, E)$ with capacities u_e on and costs c_e for each arc $e \in E$.
Task:	Find a circulation f with minimal costs $c(f)$.

Note that a flow with strictly positive value will only be sent, if there are negative cycles in the instance, because otherwise sending flow will only create costs that can be avoided.

The minimum cost circulation problem and the maximum flow problem are similar in the sense that the amount of flow that is to be sent is only limited by the capacities on the arcs. It is common to add additional balances $b : V \to \mathbb{N}$ to the nodes and to require to send (or receive) the specified amount of flow at minimum costs. In this setting, flow will also be sent on expensive paths, because the demands have to be satisfied.

Problem:	Minimum Cost Flow
Instance:	A network $\mathcal{N} = (G = (V, E), u, S^+, S^-)$, costs c_e for each arc $e \in E$ and balances b_v for the nodes $v \in V$. The balances satisfy $b_v > 0$ for $s \in S^+$, $b_v < 0$ for $t \in S^-$, $b_v = 0$ for intermediate nodes $v \in V \setminus S^+ \dot\cup S^-$, and $\sum_{v \in V} b_v = 0$.
Task:	Find a flow f that meets the supplies and demands with minimum costs $c(f)$.

A minimum cost flow that also satisfies balances is called a **transshipment**.

Solving the Minimum Cost Flow Problem. Similarly to the maximum flow problem, there exist a lot of algorithms for solving the minimum cost flow problem. The first algorithm was already given by Ford and Fulkerson [FF62], and a lot of algorithms followed. Many of the early algorithms only have pseudo-polynomial running time [Zad73], one of which is the following.

Algorithm 1.1: Successive Shortest Path

Input: $\mathcal{N} = (G = (V, E), u, S^+, S^-)$.

Output: f minimizing $\sum_{e \in E} c_e f_e$.

1. Find a shortest (with respect to the arc costs c_e) S^--S^+-path P in the residual network G_f. Costs on residual arcs $\bar{e}$ are defined to be $c_{\bar{e}} := -c_e$.
2. If there is no path P, return f.
3. Augment f by as much flow on P as possible.

Using $B = \frac{1}{2} \sum_{v \in V} |b_v|$ as the sum of demands, the Successive Shortest Path Algorithm has a pseudo-polynomial running time in $\mathcal{O}(nm + B(m + n \log n))$ [Tom71; EK72], but it will be heavily

used in the remainder of the thesis and is of importance in the area of network flows over time.

Also, polynomial and strongly polynomial algorithms for the minimum cost flow problem have been developed. Orlin's algorithm has a running time of $\mathcal{O}((m \log m)(m + n \log n))$ [Orl93], which currently is still the best strongly polynomial running time for minimum cost flows.

Practically, the network simplex algorithm [Dan51a; Cun76], which is a specific implementation of the simplex algorithm exploiting the special structure of minimum cost flow problems, and Goldberg's implementation of a scaling algorithm [Gol97] are fast algorithms to solve the Minimum Cost Flow Problem in practice.

Multi-commodity Flows. The problems we have seen so far are so-called *single-commodity* network flow problems, which means that the flow units are of the "same type" and are interchangeable with each other. If a problem consists of multiple commodities, their respective flows are not interchangeable. This setting is easily motivated by logistic applications, because orders of some specific goods cannot be served by other goods. A **multi-commodity flow** with k commodities $K := \{1, 2, \dots, k\}$ consists of k (single-commodity) network flows $f_1, f_2, \dots, f_k$ that share the capacities of the arcs, i. e., if several commodities share arc e, they have to obey $\sum_{i \in K} f_i(e) \leq u_e$. To be useful, multi-commodity maximum flow problems always contain multiple sources and sinks, because otherwise the commodities could be merged.

Problem:	Maximum Multi-commodity Flow
Instance:	A network $\mathcal{N} = (G = (V, E), u, S^+, S^-)$, k commodities $K = \{1, 2, \dots, k\}$
Task:	Find k network flows f_i for commodities $i \in K$, such that they together form a feasible network flow with maximum value, i. e., $\sum_{i=1}^{K} f_{i,e} \leq u_e$ holds for each arc $e \in E$ and $\sum_{i=1}^{K} \lvert f_i \rvert$ is maximal.

The Maximum Multi-commodity Flow Problem can be formulated as a linear program. However, due to the size of the problem instances, we are very interested in finding a combinatorial algorithm, i. e., an algorithm not based on LP techniques, to solve the problem. Such an algorithm is not known as of yet. However, Garg and Könemann [GK98] give an FPTAS based on the path formulation of fractional network flows. The algorithm is reasonably simple to be implemented, but numerical stability and the huge number of iterations that it performs introduces severe problems in practice. However, with modifications it is possible to implement the technique in a way that is practically usable in areas such as VLSI design, which is known for its huge instances [Vyg04].

Lexicographically Maximum Flows. If a network contains several sources and sinks, for a single commodity flow, we may require certain terminals to be more important than other terminals. If we have a given rank on the terminals, we can compare two flows with respect to the inflow/outflow in/from the terminals. Because the flow is compared by the flow value of the terminals in their rank ordering, these flows are referred to as lexicographically maximum flows.

Definition 1.10 (Lexicographic Flow Order). Let $\mathcal{N} = (G = (V, E), u, S^+, S^-)$ be a network with an order $t_1, t_2, \dots, t_n$ of the sources and sinks in $S^+ \cup S^-$. For a terminal node t we define

$$|f|_t \coloneqq \begin{cases} \sum_{e \in \delta^+(t)} f(e) & t \in S^+ \text{ is a source,} \\ \sum_{e \in \delta^-(t)} f(e) & t \in S^- \text{ is a sink.} \end{cases}$$

A flow f^1 is lexicographically bigger than a flow f^2, $f^1 \geq_L f^2$ if

$$\exists\, \ell \in \{0, 1, \dots, k-1\} : \forall\, i = \{1, 2, \dots, \ell\} : |f^1|_{t_i} = |f^2|_{t_i} \wedge |f^1|_{t_{\ell+1}} > |f^2|_{t_{\ell+1}},$$

or

$$\forall\, i = \{1, 2, \dots, k\} : |f^1|_{t_i} = |f^2|_{t_i}.$$

A flow f is a **lexicographically maximum flow** if it is lexicographically bigger than any other feasible flow f'. ◁

Problem:	Lexicographically Maximum Flow
Instance:	A network $\mathcal{N} = (G = (V, E), u, S^+, S^-)$ and an order of the terminals.
Task:	Find a maximum flow f that is also lexicographically maximum, i. e., $f \geq_L g$ for all feasible network flows g.

The existence of lexicographically maximum flows for any order of the terminal nodes has been shown by Minieka [Min73]. They can be used to model specific scenarios in an evacuation setting, where flow is required to leave some nodes earlier. In this setting, Minieka also showed that lexicographically maximum flows can be used to compute an earliest arrival flow, and Hoppe and Tardos used them to compute a quickest transshipment [HT94], two dynamic flow problems which we review in the next chapter.

2 Flows over Time

In this chapter, we introduce the field of network flows over time. We consider flows in a discrete and a continuous setting with multiple commodities. Flow problems can be classified into two main groups that allow for two different optimization goals: Minimizing the time horizon to satisfy some supplies and demands, and sending as much flow as possible in a given period of time. The first objective is covered by the Quickest Transshipment Problem, while the latter is the Maximum Flow over Time Problem. We explain the technique of time expansion to solve flow over time problems in pseudo-polynomial time and see how a maximum flow over time can be obtained in networks with arc release dates and deadlines by reduction to Quickest Transshipment. Finally, we show the existence of multi-commodity earliest arrival flows. Earliest arrival flows combine both optimization objectives, and multi-commodity flows model several goods that share a network.

With static network flows it is possible to gain excellent results in static situations, for example in image processing [BK04]. Also, the original motivation by Harris and Ross [HR55], as described in Section 1.2, is an application in a static scenario. However, many processes in our immediate surroundings involve some notion of time that cannot be captured by static network flows. This motivated the definition of *dynamic network flows* by Ford and Fulkerson [FF58; FF62] already in the 1950's. We also use the term *flow over time* to distinguish such problems from tasks, where a solution has to adjust to changed input data, which are also often referred to as "dynamic" [EGI98]. What distinguishes dynamic networks from static networks is that they have an additional transit time τ_e for each arc e in the underlying graph. The term flow over time refers to the idea that flow entering the arc at any point in time θ will leave the arc after some time $\theta + \tau_e$.

In the temporal setting arc capacities represent the amount of flow that can enter an arc *per time unit* and not the total amount of flow on an arc. Ford and Fulkerson originally defined the flow per time unit as a discrete value. One discrete flow unit entering an arc travels along the arc as a whole and arrives at the head after the transit time. Later the *continuous* time model was developed as a generalization. We can add additional constraints to this basic flow model. On a network level, we may change the properties of arcs over time. On the flow level, we may assign individual properties to each flow unit (or to a group of flow units), like different transit times or different capacity usage. This leads to concepts such as flow-dependent or commodity-dependent transit times. Typical optimization goals in network flow over time problems are the maximization of flow arriving at a given point in time, or the minimization of the necessary *time horizon* to send a specific amount of flow. Additionally, one might also ask for flows minimizing additional constraints, such as costs.

The applications for flows over time in both practical settings and inside the field of combinatorial optimization are countless. We cannot give an extensive overview over all applications and will concentrate on giving some examples. Due to the obvious connection to road networks, modelling of traffic is one of the most wide spread applications, examples are given by Köhler, Langkau and

Skutella [KLS02] and Harks, König and Matuschke [HKM+14]. Köhler, Möhring and Skutella survey various techniques used for traffic modelling [KMS09]. Subsequent problems involving flows over time are connected with logistics [ADS10; RT07] and (packet) routing [PW11; HGS11]. Other application scenarios include scheduling [Ful61], which is also one of the first usages of dynamic network flows in an application, cloud data stores [WSL+12] and network structuring [SZJ09]. For a general overview, see also the survey by Aronson [Aro89].

Evacuations. Another important application of network flows over time is the field of evacuation optimization and evacuation planning. On a city-scale, an evacuation can immediately be seen as a special form of a traffic problem. But also on smaller scales, like in buildings, persons can be modelled as flow units if the network structure resembles the building with adequate precision [CFS82]. Intuitively, we would take minimization of the time horizon as optimization goal in evacuation scenarios, because in hazardous situations time is crucial. Solutions having this property are referred to as *quickest transshipments*. However, there is a model that is even better suited for the evacuation scenario: *Earliest arrival flows* do not only minimize the time horizon, but also send as much flow as possible *at every point in time*. At first glance, sending the maximum amount of flow in each time step seems like a strong requirement and existence of such flows may not be likely. However, under certain conditions, such solutions *do* exist, even for multi-commodity flows. Chapter 4 deals with situations that do not allow for an earliest arrival flow. An analysis of some practical use cases of earliest arrival flows with regard to evacuations is given in Chapter 3. A recent survey of the use cases for network flows over time in the setting of evacuations is due to Dhamala [Dha14].

From a practitioners point of view, earliest arrival flows are, ideally speaking, the best we could ask for. Even in emergency scenarios, at every point in time as many evacuees can be safe if they adhere to an earliest arrival flow during the evacuation process. Taking this view we disregard the fact that people usually do not adhere to optimal solutions. Individual aspects are taken into account by Nash flows over time [KS11]. Hamacher and Tüfekçi [HT87] added even more constraints that an optimal flow in an evacuation scenario should satisfy. They introduce additional costs and are looking for an earliest arrival flow that additionally has lowest total cost. The idea behind the extension is to minimize movement. They also define lexicographically minimum cost flows, another extension of the Earliest Arrival Flow Problem that allows *priority evacuations*. In this scenario flow should leave some areas of the network, probably more endangered parts, as quickly as possible.

Outline of the chapter. We start with the introduction of our network flow over time model in Section 2.1. We investigate the discrete and continuous model and the simple Maximum Flow over Time Problem. The important technique of time expansion to solve discrete dynamic flow problems is described in Section 2.2. Quickest flow problems are introduced in Section 2.3. We use them to compute maximum flows over time in settings with arc release dates and deadlines. In Section 2.4 we introduce earliest arrival flows. We give an overview on known results and extend existence results and algorithms to the case of multi-commodity earliest arrival flows.

2.1 Discrete and Continuous Flows over Time

The central object for studies in the field of flows over time is the **dynamic network** which we denote by $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$. A dynamic network consists of a network together with **transit**

times $\tau : E \to \mathbb{R}$. Depending on the problem at hand we additionally may be given supplies and demands b_v for all nodes $v \in V$ if we are looking for a transshipment, or a time horizon T if we are maximizing flow within a time horizon. If we are interested in feasibility, we sometimes have both. For a dynamic network, the transit times cover the dynamic effect of passing time when flow travels along an edge. Flow entering an arc e at time θ leaves the arc at time $\theta + \tau_e$.

Discrete and Continuous Time Models. A network flow over time is an assignment of flow values entering arcs for each point in time. They can be defined in two ways, discrete and continuous, and the setting only changes the underlying domain. In the discrete setting, the domain is the set $\{1, 2, \dots, T\}$ of the first T points in time. For the continuous setting we take the interval $[0, T[$ as domain. We can identify the T time steps of the discrete setting with the T intervals $[0, 1[, [1, 2[, \dots, [T-1, T[$ in the continuous setting. In most cases a flow in the discrete setting corresponds to an equal flow in the continuous setting.

We define flows over time in the broadest sense that we use in this thesis, allowing several commodities and negative travel times $\tau_e < 0$. Flows over time are then defined as follows.

Definition 2.1 (Network Flow over Time). A discrete **(multi-commodity) network flow over time** is a set of functions $f_i : E \times \{1, 2, \dots, T\} \to \mathbb{R}_{\geq 0}$ for each commodity $i \in K$, and similarly, a continuous network flow over time is a set of Lebesgue-integrable functions $f_i : E \times [0, T[\to \mathbb{R}_{\geq 0}$.

In both models it is not allowed to have flow on arcs before time 0 or after the time horizon T. Thus we have $f_i(e, \theta) \coloneqq 0$ for all $\theta \geq T - \max\{0, \tau_e\}$ and $\theta < 0 - \min\{0, \tau_e\}$ in the continuous setting and $\theta > T - \max\{0, \tau_e\}$ and $\theta \leq 1 - \min\{0, \tau_e\}$ in the discrete setting.

A feasible flow over time adheres to capacity constraints

$$\sum_{i \in K} f_i(e, \theta) \leq u_e$$

for each $e \in E$ and $\theta \in [0, T[$.

For each vertex $v \in V$ and each commodity $i \in K$ we denote the **inflow** into v as

$$\operatorname{in}_{f_i}(v, \theta) \coloneqq \sum_{e \in \delta^-(v)} \int_0^{\theta - \max\{0, \tau_e\}} f_i(e, \xi)\, d\xi,$$

and the **outflow** from v as

$$\operatorname{out}_{f_i}(v, \theta) \coloneqq \sum_{e \in \delta^+(v)} \int_0^{\theta} f_i(e, \xi)\, d\xi.$$

The amount of flow that is currently at a vertex at time θ is the **excess**

$$\operatorname{ex}_{f_i}(v, \theta) \coloneqq \operatorname{in}_{f_i}(v, \theta) - \operatorname{out}_{f_i}(v, \theta).$$

A flow over time satisfies **(weak) flow conservation**, if

$$\operatorname{ex}_{f_i}(v, \theta) \geq 0$$

holds for each intermediate node $v \in V \setminus S^+ \cup S^-$ and each point in time $\theta \in [0, T[$. The flow satisfies **strict flow conservation**, if the stronger requirement

$$\operatorname{ex}_{f_i}(v, \theta) = 0$$

also holds. If supplies and demands $b_{i,v}$ for each each commodity $i \in i \in K$ and each vertex v are given, we additionally require the flow to satisfy

$$\min\{b_{i,v}, 0\} \leq \mathrm{ex}_{f_i}(v, \theta) \leq \max\{0, b_{i,v}\}$$

for all vertices and all points in time θ. In this case, all supplies should be shipped to the sinks within the given time horizon T and

$$\mathrm{ex}_{f_i}(v, T) = -b_{i,v}$$

holds for each vertex $v \in V$.

The **value** of f at time θ is the amount of flow sent by all commodities from sources to sinks until time θ and is defined to be

$$|f|_\theta := \sum_{t \in S^-} \sum_{i \in K} \mathrm{ex}_{f_i}(t, \theta)$$

with the **total value** to be $|f| := |f|_T$.

For discrete flows over time the same definitions hold, we just have to change the integration over the continuous time interval to a sum and define the inflow

$$\mathrm{in}_{f_i}(v, \theta) := \sum_{e \in \delta^-(v)} \sum_{\xi=1}^{\theta - \max\{0, \tau_e\}} f_i(e, \xi)$$

and outflow

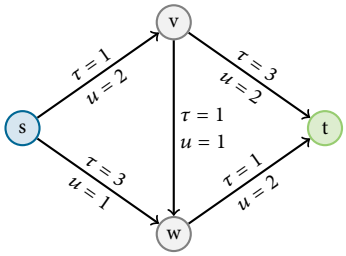
$$\mathrm{out}_{f_i}(v, \theta) := \sum_{e \in \delta^+(v)} \sum_{\xi=1}^{\theta} f_i(e, \xi)$$

with respect to the discrete domain. In the discrete setting, excess and flow value functions are only defined for the discrete points in time $\theta \in \{1, 2, \ldots, T\}$.

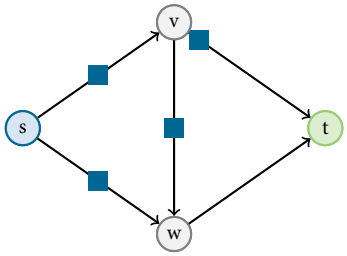
◁

Fleischer and Tardos [FT98] compared flows over time in the discrete and continuous model and showed that optimal flows for many problems have equal value in both models if transit times remain constant. Koch, Nasrabadi, and Skutella [KNS11] generalized both models to *Borel flows*, thus creating a unified model relying heavily on measure theory.

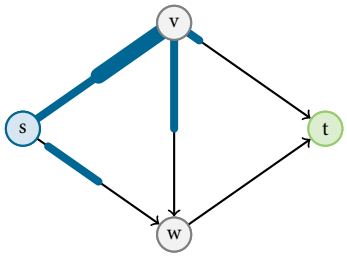
Arc Capacities in Flows over Time. Capacity constraints define an upper capacity of the *inflow* into the edge at each point in time. One may have the requirement that also the total amount of flow on an edge is limited, e. g., $\int_\theta^{\theta+\ell} f(e, \xi)\, d\xi \leq U_e$, where $U_e \in \mathbb{R}^+$ is the allowed maximal aggregated arc capacity and $\ell > 0$ is a time window. A common example that can be modelled by aggregated capacities is a bridge. The arc capacity u_e as defined above limits the inflow at each point in time. For a bridge this might be the number of lanes, for example. However, many heavy trucks on a bridge at the same time might exceed the capacity given by its structure. Therefore, flow problems having this property are sometimes referred to as "bridge flows". Aggregated arc capacities have first been studied by Melkonian [Mel07], and an FPTAS is due to Dressler and Skutella [DS11].



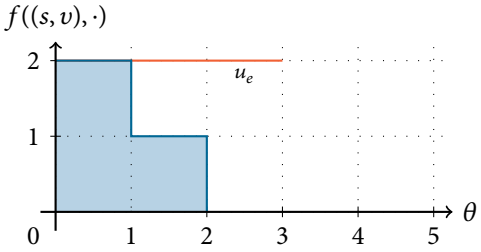
(a) A simple network instance with transit times and capacities denoted next to the arcs.



(b) The discrete flow at time $\theta = 1.5$. The flow units travel along the arcs as a whole.



(c) The continuous flow at time $\theta = 1.5$. Flow continuously travels along the arcs.



(d) The inflow rate on arc (v, w) for the whole time horizon.

Figure 2.1: A simple dynamic network and the same flow once in the discrete and once in the continuous model. We consider a time horizon of $T = 5$ and send one flow unit along the paths (s, v, t) and (s, w, t) that have a total travel time of 4 and two flow units along the faster path (s, v, w, t) with travel time 3.

Multi-commodity Flows over Time. Definition 2.1 specifies multi-commodity flows over time. Several commodities allow for a more realistic modelling of flow over time problems, especially for traffic and evacuation problems. However, they are considerably harder and introduce other complications. We will discuss difficulties that occur due to multiple commodities, whenever such difficulties arise. On the positive side, in Section 2.4 later in this chapter we show that earliest arrival flows also exist in the multi-commodity setting under certain conditions. An interesting extension in the case of several commodities are *commodity dependent transit times*. In this case the transit time $\tau_{i,e}$ for an arc e depends on the commodity. Commodity dependent transit times are discussed in detail in [Gro09]. Unfortunately, non-existence results for earliest arrival flows carry over which makes approximation as described in Chapter 4 necessary for the case of commodity dependent transit times. They also introduce the additional problem of *overtaking*, which might be desirable or not, depending on the application.

Path Flows. Similar to classical networks, we can also define a path based formulation for flows over time. Again, we denote the set of all source-sink-paths by $\mathcal{P}$. Now, a path flow assigns a flow value not only to each path, but to each path at each point in time. The flow value denotes the inflow rate into a *path* and not into an edge any more. A flow unit travels along the path without any breaks and enters an edge $e = (v, w)$ at time $\tau(P_{[s,v]})$, i. e., after travelling along all preceding edges. To make sure the flow does not exceed the arc capacities, we have to make sure that not too much flow enters an arc at the same time. For simplicity of presentation, we only consider instances with non-

negative travel times for the given arcs here and discuss the special case with negative travel times in Chapter 5. Notice that residual networks introduce arcs with negative travel times. However, we will see that these are not problematic.

Definition 2.2 (Path Flow over Time). Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network with $\tau \geq 0$ and $\mathcal{P}$ the set of all s-t-paths for any pair of source $s \in S^+$ and sink $t \in S^-$. The **length** of a path P is defined as

$$\tau(P) \coloneqq \sum_{e \in P} \tau_e.$$

For an edge $e \in E$ and a given point in time θ we define the set of paths using e at the specified time as

$$\mathcal{P}_e^\theta \coloneqq \left\{P \in \mathcal{P} \mid \tau(P_{[s,v]}) \leq \theta \wedge \tau(P_{[v,t]}) < T - \theta\right\}.$$

In the case that we have a residual network, the definition is accordingly. Here, $\overleftrightarrow{\mathcal{P}}$ defines the set of all s-t-paths in the residual network and we denote paths using an arc e, or its reverse arc at time θ by $\overleftrightarrow{\mathcal{P}}_e^\theta$ and $\overleftrightarrow{\mathcal{P}}_{\overleftarrow{e}}^\theta$, respectively.

A discrete **path flow over time** is an assignment $x : \mathcal{P} \times \{1, 2, \dots, T\} \to \mathbb{R}_{\geq 0}$, and a continuous path flow over time is an assignment $x : \mathcal{P} \times [0, T[\to \mathbb{R}_{\geq 0}$. Both flows have to satisfy the following constraints on their respective domains: There is no flow on copies of paths that arrive too late, i. e., $x(P, \theta) = 0$ for $\theta \in [T - \tau(P), T[$. Additionally, the capacity of arcs is not exceeded. That is,

$$\sum_{P \in \overleftrightarrow{\mathcal{P}}_e^\theta} x\big(P, \theta - \tau(P_{[s,v]})\big) \leq u_e$$

holds for all edges $e = (v, w) \in E$ and times $\theta \in \{1, 2, \dots, T\}$ and $\theta \in [0, T[$, respectively. We denote the **value** of a path flow over time by

$$|x| \coloneqq \sum_{P \in \mathcal{P}} \sum_{\theta=1}^{T} x(P, \theta),$$

and

$$|x| \coloneqq \sum_{P \in \mathcal{P}} \int_{\theta=0}^{T} x(P, \theta).$$

◁

Notice that we do not have to worry about flow conservation for path based flows over time, as they automatically enforce strict flow conservation. On the other hand, path based flows do not allow problem instances that require waiting in intermediate nodes without changing the network structure and adding loops. Such approaches also suffer from the introduction of non-simple paths. Observe also that it is straight forward to construct a flow over time f from a given path flow x with the same value $|f| = |x|$, while it is not so clear how a path flow can be constructed from a given edge flow.

As a starting point for the introduction of network flows over time we will consider traditional problems and techniques that have been introduced already in the 1950's. This is the most basic extension of network flows to the temporal setting.

Maximum Flow over Time Problem. The Maximum Flow over Time Problem was introduced by Ford and Fulkerson [FF58; FF62] and is the simplest dynamic flow problem. The problem consists of finding a dynamic flow that sends as much flow as possible within a given time bound and is stated as follows:

Problem:	Maximum Flow over Time
Instance:	A dynamic network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ and time horizon T.
Task:	A flow over time f with time horizon T that maximizes the flow value $\lvert f \rvert$.

By using the extended network from Definition 1.5, the Maximum Flow over Time Problem can be reduced to the single source-single sink case. The additional arcs are equipped with zero transit times. The problem has been studied only for the case of non-negative travel times so far, and two methods to solve the problem have been proposed. The first relies on time expansion, which we discuss later in this section in 2.2, the other is based on a reduction to the Minimum Cost Flow Problem in an extended network.

Notice that in the discrete setting, the Maximum Flow over Time Problem can be written as a linear program that enforces the constraints for a feasible flow over time. However, this LP would be exponential in size even for the edge based formulation as flow conservation for all T time steps has to be ensured and T is part of the input. An equivalent path based formulation is possible but obviously not smaller. It is probably surprising that a polynomial algorithm to compute a maximum flow over time not only exists, but that it is also fairly simple. For this algorithm we need some additional notation for special types of flows over time.

Temporally Repeated Flows. The encoding size of flows over time can be large due to the time horizon. By definition (of either the path or edge based formulation), it is necessary to specify a flow value not only on each arc (or path), but also for each point in time. At first glance, the super-polynomial size seems unavoidable, as T is not polynomial in the input size in general. However, nicer descriptions of flows over time are possible. If we can enforce the flow sent on paths to be constant for most time steps and only switch its value polynomially often, we can achieve a description in polynomial size.

Definition 2.3 (Temporally Repeated Flow). Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network and let x be a static path flow in the underlying network. The corresponding **temporally repeated flow** is defined as

$$f(e, \theta) := \sum_{P \in \mathcal{P}_e^\theta} x(P)$$

for each arc $e = (v, w)$ and each point in time $\theta \in [0, T[$ or $\{1, 2, \dots, T\}$ for the continuous or discrete setting, respectively.

Let $\overrightarrow{\mathcal{P}}$ again be the set of all paths in the residual network (with respect to a given flow). A path decomposition of the flow in the residual network with paths $P \in \overrightarrow{\mathcal{P}}$ is called **generalized path**

decomposition. Let x be such a general path decomposition with $P \in \overleftrightarrow{\mathcal{P}}$. The corresponding **generalized temporally repeated flow** is defined as

$$f(e,\theta) \coloneqq \sum_{P \in \overleftrightarrow{\mathcal{P}}_e^\theta} x(P) - \sum_{P \in \overleftrightarrow{\mathcal{P}}_{\bar{e}}^\theta} x(P).$$

◁

Observe that the temporally repeated flow f in fact is a feasible flow because the flow value on each edge e is limited by the flow value on e within the static network flow x. In a similar way we can define a path flow over time x' by setting $x'(P,\theta) \coloneqq x(P)$ for each time $\theta \in [0, T - \tau(P)[$.

For a generalized temporally repeated flow the same observation is not necessarily true. Notice, that by the above definition it is totally possible that the obtained flow might be negative. This happens if flow is reduced by the flow value on the reverse arc when there is no flow on the corresponding (forward) arc. Thus, whenever generalized temporally repeated flows are used, we have to show that they indeed form a feasible flow and the definition is justified.

Ford and Fulkerson Algorithm. If the travel times for an instance of the Maximum Flow over Time Problem are all non-negative, a temporally repeated flow with maximum value can be computed in strongly polynomial time [FF58]. Consider the following algorithm.

Algorithm 2.1: Ford Fulkerson Max Flow over Time

Input: $\mathcal{N} = (G = (V,E), u, \tau, S^+, S^-)$ with transit times $\tau \geq 0$ and a time horizon T.

Output: A temporally repeated flow with maximum value.

1. Create the extended network $\mathcal{N}$ by adding a super source and super sink and add an additional artificial arc (t^*, s^*).
2. Define arc costs $c_e \coloneqq \tau_e$ for $e \in E$ and $c_{(t^*,s^*)} \coloneqq -T$.
3. Compute a minimum cost circulation f in $\mathcal{N}'$.
4. Compute a path decomposition x of f.
5. Return a temporally repeated flow that sends flow along paths in x as long as possible, e. g., within the interval $[0, T - \tau(P)[$.

The idea behind the algorithm is the following: Consider a path P in the path decomposition. The contribution of this path to the value of the minimum cost circulation is exactly $\tau(P) - T = -(T - \tau(P))$, which is exactly the negative value of the period of time path P is used in the temporally repeated flow. Thus, the smaller this value the longer the path is used and the more value is contributed to the maximum flow over time. The value is minimized by a minimum cost computation. The running time of the algorithm is bounded by the running time of a static minimum cost flow computation and is thus strongly polynomial. The algorithm also proves that no flow storage at intermediate nodes is necessary to obtain an optimal solution.

In a certain sense this is the ideal algorithm. It can be implemented such that it basically works in the original network (only few arcs are added), the running time is strongly polynomial and the solution can be described in polynomial size as temporally repeated flow. We will therefore try to achieve similar algorithms for more complicated problems. However, we mostly do not find algorithms as powerful as this. Then, we will try to find approximate solutions that are simple and can be represented as temporally repeated flows.

Multi-commodity Maximum Flow over Time. Notice, that a solution of the Maximum Flow over Time Problem is a single-commodity flow. Adding additional commodities to the problem makes it weakly $\mathcal{NP}$-hard [HHS07], regardless of whether flow storage is allowed or not. Algorithm 2.1 computes a maximum flow over time that does not require flow storage in intermediate nodes. In the multi-commodity case, this is different: The possibility of flow storage in intermediate nodes may change the value of a maximum flow. Fleischer and Skutella [FS07] give an FPTAS that computes a multi-commodity flow over time in the setting with flow storage in intermediate nodes. For the case without flow storage, they present a $(2+\varepsilon)$-approximation algorithm. An extended FPTAS that approximates the optimal flow in the setting without flow storage is due to Groß and Skutella [GS12b]. The same discrepancy in the value of a maximum flow between flows that use storage, and those that do not, occurs if we allow negative travel times for the single commodity Maximum Flow over Time Problem. We discuss the implications of negative travel times in Chapter 5.

2.2 Time Expansion

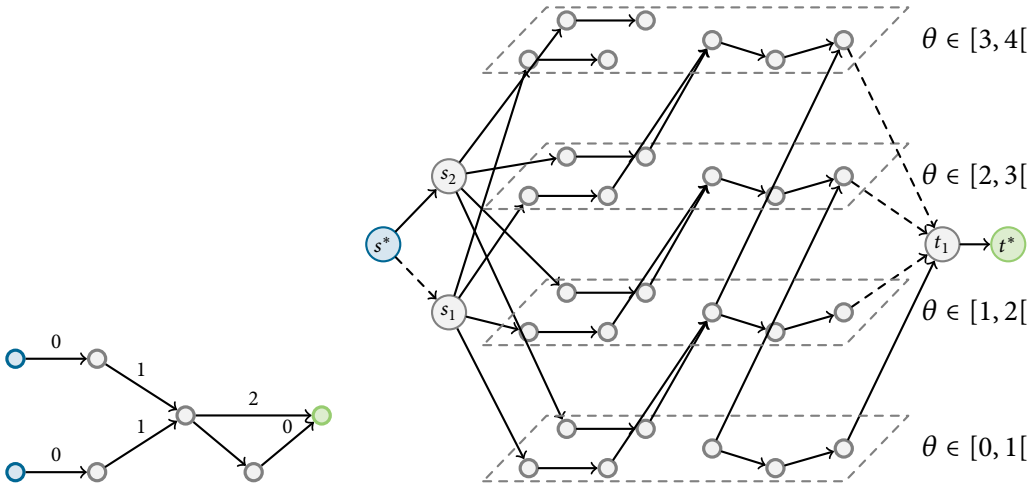
A popular way of computing flows over time is the so-called *time expansion*. The general idea of time expansion is to remove the temporal aspect of dynamic network flows by introducing copies of the (static) network structure and to identify each copy of the network with a certain point in time. As an example, assume an arc $e = (v, w)$ with travel time τ_e should be used at time θ. The flow then starts at node copy v^θ and arrives after τ_e time units at node copy $w^{\theta+\tau_e}$. Generally speaking, a copy of an edge $e = (v, w)$ exists between each copy of v and w whose time layers differ by the transit time of the edge. The following definition is for the continuous case for multiple commodities with possibly negative travel times. The time-expanded network in the discrete case is defined similarly. Notice that the definition only allows continuous flows that are constant between integral points in time.

Definition 2.4 (Time-expanded Network). Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a network with integral time horizon $T \in \mathbb{Z}_{\geq 0}$, and optionally also supplies and demands for each vertex $v \in V$ and each commodity $i \in K$, denoted by $b_{i,v}$. We denote the corresponding **time-expanded network** by $\mathcal{N}^T := (G = (V^T, E^T), u^T, S^{+T}, S^{-T})$. Its components are defined as follows. The set of nodes consists of T copies of the original network denoted by v^θ, super sources s_i^* and super sinks t_i^* for each commodity, and also the original terminal nodes, that is,

$$V^T := \{v^\theta \mid v \in V, \theta \in \{1, \dots, T\}\} \cup S^+ \cup S^- \cup \{s_i^*, t_i^* \mid i \in K\}.$$

As new sources and sinks, we use the super terminals for each commodity

$$S^{+T} := \{s_i^* \mid i \in K\}, S^{-T} := \{t_i^* \mid i \in K\}.$$



(a) A simple example with non-negative travel times.

(b) The time-expanded network contains two sources and a sink outside the temporal copies and an additional super source on the left side. The sink and super sink are on the right side.

Figure 2.2: An example of the time-expanded network for a single-commodity dynamic network with two sources and a single sink. The time-expanded network covers a time horizon of $T = 4$.

For each arc e we create $T - |\tau_e|$ copies connecting node copies on time layers θ with copies on time layer $\theta + \tau_e$, if the arrival time lies within the feasible time interval. For positive transit times, arc copies start at time 1 and stop at time $T - \tau_e$, and copies of arcs with negative transit time start at time $|\tau_e| + 1$ and end at time T. The set of all arc copies is defined as

$$\tilde{E} := \left\{ e^\theta = (v^\theta, w^{\theta+\tau_e}) \;\middle|\; \begin{array}{l} e = (v,w) \in E, \\ \theta \in \{1 - \min\{0,\tau_e\}, \dots, T - \max\{0,\tau_e\}\} \end{array} \right\}.$$

Additionally, the original terminal nodes are connected their respective copies on each layer and the super nodes with the source and sink nodes. The arcs connected to sources and their copies are defined as

$$E^+ := \cup\{(s_i^*, s) \mid i \in K, s \in S_i^+\} \cup \{(s, s^\theta) \mid i \in K, s \in S_i^+, \theta \in \{1,2,\dots,T\}\},$$

and arcs connected to sinks and their copies are denoted by

$$E^- := \cup\{(t, t_i^*) \mid i \in K, t \in S_i^-\} \cup \{(t^\theta, t) \mid i \in K, t \in S_i^-, \theta \in \{1,2,\dots,T\}\}.$$

The set of all edges in the time expanded network then is defined as

$$E^T := \tilde{E} \cup E^+ \cup E^-.$$

If node storage is desired, we additionally add **holdover arcs** between node copies belonging to intermediate nodes of the original network

$$H := \left\{ (v^\theta, v^{\theta+1}) \;\middle|\; v \in V \setminus \bigcup_{i \in K} S^+ \cup S^-, \theta \in \{1,\dots,T-1\} \right\},$$

and set $E^T := E^T \cup H$ in this case. Arc capacities are defined depending on node balances. If no node balances are given, the capacities are defined as

$$u^T_{e'} := \begin{cases} u_e & \text{if } e' \in \tilde{E} \text{ with } e' = e^\theta, \\ \infty & \text{else,} \end{cases} \quad \text{for all } e' \in E^T.$$

If node balances are given, the extended definition

$$u^T_{e'} := \begin{cases} u_e & \text{if } e' \in \tilde{E} \text{ with } e' = e^\theta, \\ b_{i,s} & e' = (s_i^*, v), \\ -b_{i,t} & e' = (t, t_i^*), \\ \infty & \text{else,} \end{cases} \quad \text{for all } e' \in E^T.$$

specifies extended arc capacities limiting the outflow of sources and inflow of sinks. ◁

Notice that this definition slightly differs from other definitions, e. g., in [Sku09]. Traditional definitions (with and without holdover) use holdover arcs between the sources and sinks. In these definitions arcs of type (s_j, s_j^θ) (t_j, t_j^θ) are replaced by arcs $(s_j^{\theta-1}, s_j^\theta)$; only the first copy s_j^0 is connected to the super source s^* and the last copy t_t^T is connected to the super sink. In most cases, these changes do not make a big difference. However, in the context of *size increasing time-expanded networks* this is not only interesting on the practical side for the implementation of algorithms on time-expanded networks, but also for approximation algorithms that we see in Chapter 4.

Definition 2.4 specifies holdover arcs with infinite capacity. The storage capacity of nodes can be limited by setting holdover capacity to any positive value. However, neither from practical nor theoretical point of view it makes a difference whether the node capacity is any positive, but finite value or zero and we will not further investigate this case.

Lemma 2.5. *Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network with integral transit times $\tau_e \in \mathbb{Z}$. For any time horizon $T \in \mathbb{Z}_{\geq 0}$, a feasible static S^{+T}-S^{-T}-multi-commodity flow f^T in the time-expanded network $\mathcal{N}^T$ yields a feasible multi-commodity flow over time f with the sources S^+ and sinks S^- in $\mathcal{N}$ with time horizon T and equal value $|f| = |f^T|$.*

Proof. We will show both directions by defining a flow in the respective model and showing that it is indeed a feasible flow (over time). We show the case with holdover arcs and refer to the fact that calculations are still valid if holdover arcs are missing.

Constructing a Flow over Time from a Static Flow. For a given multi-commodity flow f^T in the time-expanded network we define

$$f_i(e, \theta) := f_i^T(e^{\lfloor \theta \rfloor + 1})$$

for all arcs $e \in E, \theta \in [0, T[$ and all $i \in K$.

The dynamic flow f obeys capacities due to

$$\sum_{i \in K} f_i(e, \theta) = \sum_{i \in K} f_i^T(e^{\lfloor \theta \rfloor + 1}) \leq u_{e^\theta} = u_e$$

for all arcs $e \in E$ and $\theta \in [0, T[$.

For the flow conservation we consider the flow in the time-expanded network. We have zero excess at internal nodes due to (strict) flow conservation in the static case. We split the flow into flow on holdover arcs and flow on copies of the original arcs and use the fact that the inflow and outflow of all holdover arcs up to time θ cancels out.

$$
\begin{aligned}
0 &= \sum_{\xi=1}^{\theta} \sum_{e\in\delta^-(v^\xi)} f_i^T(e) - \sum_{\xi=1}^{\theta} \sum_{e\in\delta^+(v^\xi)} f_i^T(e) \\
&= \left(\sum_{\xi=1}^{\theta} \sum_{e\in\delta^-(v^\xi)\cap E^T} f_i^T(e) + \sum_{e\in\delta^-(v^\xi)\cap H} f_i^T(e) \right) \\
&\quad - \left(\sum_{\xi=1}^{\theta} \sum_{e\in\delta^+(v^\xi)\cap E^T} f_i^T(e) - \sum_{e\in\delta^+(v^\xi)\cap H} f_i^T(e) \right) \\
&= \left(\sum_{\xi=1}^{\theta} \sum_{e\in\delta^-(v^\xi)\cap E^T} f_i^T(e) \right) - \left(\sum_{\xi=1}^{\theta} \sum_{e\in\delta^+(v^\xi)\cap E^T} f_i^T(e) \right) \\
&\quad + \left(\sum_{\xi=1}^{\theta} \sum_{e\in\delta^-(v^\xi)\cap H} f_i^T(e) - \sum_{\xi=1}^{\theta} \sum_{e\in\delta^+(v^\xi)\cap H} f_i^T(e) \right) \\
&= \left(\sum_{\xi=1}^{\theta} \sum_{e\in\delta^-(v^\xi)\cap E^T} f_i^T(e) \right) - \left(\sum_{\xi=1}^{\theta} \sum_{e\in\delta^+(v^\xi)\cap E^T} f_i^T(e) \right) - f_i^T((v^\theta, v^{\theta+1})) \\
&= \sum_{e\in\delta^-(v)} \sum_{\xi=1}^{\theta-\max\{0,\tau_e\}} f_i^T(e^\xi) - \sum_{e\in\delta^+(v)} \sum_{\xi=1}^{\theta} f_i^T(e^\xi) - f_i^T((v^\theta, v^{\theta+1})) \\
&= \sum_{e\in\delta^-(v)} \int_{\xi=0}^{\theta-\max\{0,\tau_e\}} f_i^T(e^{\lfloor\xi\rfloor})\, d\xi - \sum_{e\in\delta^+(v)} \int_{\xi=0}^{\theta} f_i^T(e^{\lfloor\xi\rfloor})\, d\xi - f_i^T\big((v^\theta, v^{\theta+1})\big) \\
&= \sum_{e\in\delta^-(v)} \int_{\xi=0}^{\theta-\max\{0,\tau_e\}} f_i(e^\xi)\, d\xi - \sum_{e\in\delta^+(v)} \int_{\xi=0}^{\theta} f_i(e^\xi)\, d\xi - f_i^T\big((v^\theta, v^{\theta+1})\big) \\
&= \mathrm{in}_{f_i}(v,\theta) - \mathrm{out}_{f_i}(v,\theta) - f_i^T\big((v^\theta, v^{\theta+1})\big) \\
&= \mathrm{ex}_{f_i}(v,\theta) - f_i^T\big((v^\theta, v^{\theta+1})\big)
\end{aligned}
$$

for all nodes $v \in V \setminus S^+ \cup S^-$, times $\theta \in [0, T[$ and commodities $i \in K$. We started with a value equal to zero and due to the flow conservation constraint, we have $\mathrm{ex}_{f_i}(v,\theta) \geq 0$, and the value of the excess equals the flow value on the holdover arc from node v^θ to $v^{\theta+1}$. There is no holdover arc for time T any more, thus equality holds at the last point in time. For the case without waiting in intermediate nodes, we have no holdover arcs and therefore strict flow conservation also holds in this case. Using the same calculation and taking out arcs connecting copies of sources s^θ and sinks t^θ with the respective super vertices s and t, respectively we get

$$
\mathrm{ex}_{f_i}(s,T) = -\sum_{i=1}^{\theta} f_i^T((s,s^\theta)) \qquad \text{and} \qquad \mathrm{ex}_{f_i}(t,T) = \sum_{i=1}^{\theta} f_i^T((t^\theta,t))
$$

for each source $s \in S^+$ and sink $t \in S^-$, hence supplies and demands are satisfied.

Constructing a Static Flow from a Flow over Time. We set the flow value on copy θ of an arc e to be the total flow value that enters the arc in one time step in the continuous flow. Thus we define

$$f_i^T\left(e^\theta\right) := \int_{\theta-1}^{\theta} f_i(e,\xi)\,d\xi$$

for all arcs $e \in E$ and time steps $\theta \in \{1, 2, T - \max\{0, \tau_e\}\}$. If holdover arcs are present, we define the flow value on the arcs by

$$f_i^T\left(v^\theta, v^{\theta+1}\right) := \mathrm{ex}_{f_i}(v,\theta)$$

for intermediate nodes $v \in V \setminus S^+ \uplus S^-$ and all time steps $\theta \in \{2, \dots, T\}$. For any source $s \in S^+$ we define

$$f_i^T\left((s, s^\theta)\right) := \mathrm{in}_{f_i}(s,\theta) - \mathrm{in}_{f_i}(s,\theta-1),$$

and similar for the sinks $t \in S^-$

$$f_i^T\left((t^\theta, t)\right) := \mathrm{out}_{f_i}(t,\theta) - \mathrm{out}_{f_i}(t,\theta-1).$$

The arcs incident to the super terminals are just defined accordingly.

We split the inflow and outflow into flow on copies of arcs and flow on holdover arcs and use the definitions for inflow and excess to see that flow conservation holds at all nodes $v \in V$ and $\theta \in \{1, 2, \dots, T\}$.

$$\begin{aligned}
&\sum_{\delta^-(v^\theta)} f_i^T\left(e^{\theta-\max\{0,\tau_e\}}\right) - \sum_{\delta^+(v^\theta)} f_i^T\left(e^\theta\right) \\
&= \sum_{\delta^-(v^\theta)} \int_{\theta-\max\{0,\tau_e\}-1}^{\theta-\max\{0,\tau_e\}} f_i(e,\xi)\,d\xi - \sum_{\delta^+(v^\theta)} \int_{\theta-1}^{\theta} f_i(e,\xi)\,d\xi \\
&= \left(\left(\mathrm{in}_{f_i}(v,\theta) - \mathrm{in}_{f_i}(v,\theta-1)\right) + \mathrm{ex}_{f_i}(v,\theta-1)\right) \\
&\quad - \left(\left(\mathrm{out}_{f_i}(v,\theta) - \mathrm{out}_{f_i}(v,\theta-1)\right) + \mathrm{ex}_{f_i}(v,\theta)\right) \\
&= 0
\end{aligned}$$

The same calculation for sources and sinks shows that the value of both flows is equal. The capacities are obeyed by f^T because

$$\begin{aligned}
f^T\left(e^\theta\right) = \sum_{i\in K} f_i^T(e^\theta) &= \sum_{i\in K} \int_{\theta-1}^{\theta} f_i(e,\xi)\,d\xi \\
&= \int_{\theta-1}^{\theta} \sum_{i\in K} f_i(e,\xi)\,d\xi \le \int_{\theta-1}^{\theta} u_e\,d\xi = u_e.
\end{aligned}$$

□

Using the lemma it is easy to see that a maximum flow over time can be computed in the time-expanded network. Any flow in the time-expanded network yields a flow over time of the same value, thus a maximal flow over time corresponds to a maximum flow in the expanded network.

The size of the time-expanded network is not necessarily polynomial in the original input size for the dynamic problem any more because T is not bounded. Thus, algorithms basing on the technique of time expansion are only pseudo-polynomial.

Dynamic Problems in Time-expanded Networks. Many flow problems with dynamic counterparts can be solved by applying the discrete algorithm on the (static) time-expanded network. This is basically true for all problems, in which the state of an arc at one point in time does not affect its state at other points in time. For other problems, it might not be as obvious. An example are *interdiction* problems. In such problems, a network is given and the goal is to remove arcs (probably with costs) such that in the remaining network the value of a maximum flow is either minimized or zero.[1] In a temporal version of such problems, the interdiction of an arc at time θ would be equal to interdiction in the time-expanded network for all times $\theta' \geq \theta$ which is not the same problem any more. However, these issues do not occur in the problems we consider in this thesis. As a starting point for literature on interdiction problems see for example [CSM04; MM70].

An example of a more complex flow problem that can be solved using time-expanded networks is the Minimum Cost Flow Problem. Similar to our proofs for Lemma 2.5, one can show that a static flow in the time-expanded network induces a dynamic flow with the same value and vice versa [Sku09]. However, in opposition to the static case (and to the Maximum Flow over Time Problem) there is no polynomial algorithm for the problem, unless $\mathcal{P} = \mathcal{NP}$. The $\mathcal{NP}$-hardness has first been observed by Klinz and Woeginger [KW04]. They were also the first to show that it is $\mathcal{NP}$-hard to find a temporally repeated flow with minimum cost.

Other problems that can be solved using time-expanded networks are the Quickest Transshipment Problem and the Earliest Arrival Flow Problem, as we see in the remainder of this chapter.

Cyclically time-expanded Networks. One way to generalize time-expanded networks are **cyclically time-expanded networks**. These networks have additional arcs. Consider an arc $\left(v^\theta, w^{\theta+\tau_{(v,w)}}\right)$ where the copy for time $\theta+\tau_{(v,w)} > T$ would be outside the time horizon. In this case, using the modulo operation node v^θ is connected with copy $w^{\theta+\tau_{(v,w)} \equiv T}$. Solutions in such networks can be used to be repeated, e. g., a transshipment may be sent every day when the time horizon equals the length of a day. This technique has been applied to the location routing problem by Harks et al. [HKM+14] and Köhler and Strehler [KS10; KS12] used the same technique to optimize traffic lights.

Shrinking the Network Size. A problem using time-expanded networks in practice is their typically huge size. There exist several techniques to shrink the network size without changing the resulting flow values too much and allowing for theoretically efficient fully polynomial-time approximation schemes. The so-called **condensed time-expanded network** networks were introduced by Fleischer and Skutella [FS07] to approximate the Minimum Cost Flow over Time Problem. In condensed time-expanded networks, all travel times are divided by the same constant Δ and the capacity is in turn scaled by the same factor. If all travel times are multiples of Δ, the same flows can be computed in smaller networks. If not all travel times are multiples of Δ, an FPTAS can be gained if the rounding factor is chosen carefully.

Fleischer and Skutella also introduced a more advanced version of time condensation, the **geometrically condensed time-expanded networks**. These networks do not scale transit times by a single constant Δ, but by several geometrically increasing constants. Thus it is possible to achieve a more

[1] The Minimum Cut Problem problem can be seen as a very restricted variant of network interdiction problems and in fact the original task described by Harris and Ross [HR55] asks for an interdiction of the Soviet rail network at cheapest costs.

detailed approximation for earlier time steps. Geometrical condensation allows for an FPTAS for the Earliest Arrival Flow Problem [FS07]. We apply this type of networks in Section 4.4 to establish a fully polynomial-time approximation scheme to compute instance optimal value-approximate earliest arrival flows. Even more generalized are the *elastic time-expanded networks* introduced by Wang et al. [WSL+12]. These networks allow different condensation factors for given time intervals $[T_i, T_{i+1}[$.

Groß and Skutella use **sequence rounded time-expanded networks**, another variant of time condensation that does not only round travel times of arcs, but rounds *sequences* of arcs to a single arc in the expanded network [GS12b]. The technique has use cases if optimal solutions require non simple paths. For many problems, including the above mentioned Maximum Flow over Time Problem, the Quickest Transshipment Problem and the Earliest Arrival Flow Problem optimum solutions do not need waiting in intermediate nodes. Such solutions correspond to paths in the time-expanded network that visit at most one node copy for any node $v \in V$. For problems like the Maximum Multi-commodity Flow over Time Problem, it can be beneficial for flow to use cycles if waiting in intermediate nodes is not allowed. If such paths are transferred to (geometrically) condensed time-expanded networks, the loss due to the rounding can no longer be bounded by a factor of ε. In these cases sequence rounding still allows to get an FPTAS.

2.3 Quickest Transshipments

If a problem contains a temporal dimension, minimizing the necessary time becomes an additional objective for optimization besides maximizing the value. For flows over time we consider a temporal variant of the Transshipment Problem. Similar to the static case, we have supplies for sources and demands for sinks, and a transshipment sends the necessary flow from sources to sinks. What is different in the dynamic case is that the arc capacities only limit the inflow. As a consequence, more flow can be sent as a static minimum cut in the network allows by using paths more than once.

Definition 2.6 (Dynamic Transshipment)**.** Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network, and let $b : V \to \mathbb{Z}$ supplies and demands for sources and sinks, respectively.

If a flow over time f with a time horizon of T satisfies the supplies and demands, f is a **dynamic transshipment** or **transshipment over time** with time horizon T, e. g., for all commodities $i \in K$ the flow values satisfy $-\operatorname{ex}_{f_i}(s, T) = b_s$, and $\operatorname{ex}_{f_i}(t, T) = -b_t$ for sources s and sinks t, respectively. ◁

Feasibility of Transshipments. The first question that arises is whether the given supplies and demands can be satisfied by *some* transshipment over time. To answer the question, the total capacity (which is the value of a minimum cut) of the network is no bound for the supplies any more. If there is a source-sink-path, it can be used several times. Thus, in the temporal setting, we can decide feasibility by ignoring capacities which leads us to the following observation.

Observation 2.7. *For a given network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ and supplies b_v, we can decide whether there exists a $T < \infty$ such that a feasible transshipment within time T exists by a single static flow computation in the network with capacities $u'_e \equiv \infty$.*

Proof. For a feasible transshipment, we need to decide for each pair of source and sink how much flow shall be sent from the source to the respective sink in total. This assignment must be such that if any positive amount of flow is supposed to be sent from a source to a sink, then there has to be a

path in the network that connects them. Since we can compensate a small capacity by a larger time horizon the capacity of this path is not important. A static transshipment in the network with infinite capacities u'_e is such an assignment. If the transshipment needs more capacity on some arcs, in the first step as much flow as possible is sent, continued with the second step and so on. At some finite point in time T all supplies have been shipped and a feasible transshipment over time exists. □

Quickest Transshipments. If a feasible transshipment exists, we are interested in the necessary time horizon to ship all supplies. This motivates the Quickest Transshipment Problem which asks for a transshipment satisfying given balances in the shortest period of time.

Problem: Quickest Transshipment

Instance: A dynamic network $\mathcal{N} = (G, u, \tau, S^+, S^-)$, supplies and demands for the sources and sinks $b : V \to \mathbb{R}$.

Task: Compute a feasible transshipment over time within the network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ with minimal time horizon T.

In the case of a single source and single sink (but with a finite demand), the problem is also called Quickest Flow Problem. If several sources are present but only a single sink, the problem is also called Evacuation Problem.
If arcs are equipped with costs $c_{e,i}$, we might ask for a Quickest Transshipment with **bounded costs**, i. e., the transshipment should also satisfy

$$\sum_{e \in E} c_{e,i} \int_0^T f_i(e, \xi)\, d\xi \leq C_i$$

for cost bounds C_i for each commodity $i \in K$.

Because the solution of the Quickest Transshipment Problem is a transshipment with lowest possible time bound, a natural application is modelling evacuation scenarios [CFS82; CHT88]. For these scenarios we mostly use special instances which only contain a single sink that is the safe target for all flow units. In the case of building evacuation, we assume this single sink to be "outside". However, the problem is not very suitable to model an evacuation scenario. If a catastrophe arises, we are typically not only interested in minimizing the time until the last evacuee is safe but want to evacuate as many evacuees as early as possible. In a feasible solution of the Quickest Transshipment Problem it is possible for flow to travel around a cycle before entering the sink if the necessary time horizon is is not increased by travelling trough the cycle. We will see in Section 2.4 that earliest arrival flows are better suited for the evacuation scenario and avoid solutions containing cycles.

The Quickest Flow Problem can be solved in pseudo-polynomial running time by solving a static Transshipment Problem within a time-expanded network. With known lower and upper bounds of the time horizon a polynomial algorithm for the Transshipment over Time Problem gives a polynomial algorithm for solving the quickest transshipment problem by using it in a binary search framework. Burkard, Dlaska, and Klinz [BDK93] have shown that this approach is relevant in practice if the binary search is improved by some heuristics. However, even with the improve-

ments the worst case running time does not become strongly polynomial. Instead of using a simple binary search, the optimal time horizon for a feasible transshipment can be computed by using Megiddo's search framework ([Meg79], see Theorem 1.1). Burkard, Dlaska, and Klinz were the first who showed how using the parametric search framework results in a strongly polynomial algorithm for the Quickest Flow Problem.

For the most efficient use of the binary search framework we are interested in good lower and upper bounds for the time horizon of a dynamic transshipment. As from any source at least some flow has to be sent to any sink, a lower bound can be computed easily by using shortest paths. For the upper bound we have to take the demands into account as we do not know how much flow can be sent using shortest paths.

Observation 2.8. *Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be an instance of the Quickest Transshipment Problem, $B^+ := \sum_{s \in S^+} b(s)$ the total supplies, and let $\operatorname{dist}(v, w)$ denote the length of a shortest path between two vertices v and w. Let*

$$SP^- := \max\{\min\{\operatorname{dist}(s,t) \mid t \in S^-\} \mid s \in S^+\}$$

be the length of the longest shortest path from any source to some sink and let

$$SP^+ := \max\{\operatorname{dist}(s,t) \mid s \in S^+, t \in S^-\}$$

be the length of the longest shortest path between any pair of source and sink.

Then, SP^- is a lower bound for the time horizon for a quickest transshipment. An upper bound for the time horizon is given by

$$SP^+ + \frac{B^+}{\min_{e \in E} u_e}.$$

Proof. From any source s, a positive amount of flow has to be sent to some sink t. Thus, the lower bound is obvious.

We will show the upper bound by induction on the number of sources. Let s be the only source. We can send all B^+ flow units along shortest paths whose length are bounded by SP^+. The capacity of the path is at least the minimum capacity which allows us to send all flow units in time $SP^+ + \frac{B^+}{\min_{e \in E} u_e}$.

For the multiple source case we take out a source s with supply b_s from the set of sources and solve the instance with the smaller number of sources. Let x be a solution of this instance. The upper bound is by induction $SP^+ + \frac{B^+ - b_s}{\min_{e \in E} u_e}$.

We have to sent additional b_s units of flow along paths whose lengths are bounded by SP^+. In the worst case, all of those paths are blocked by the solution x. In that case, the first of the remaining flow units arrives directly after the old flow needing additionally $\frac{b_s}{\min_{e \in E} u_e}$ time units. In total the upper bound is

$$SP^+ + \frac{B^+ - b_s}{\min_{e \in E} u_e} + \frac{b_s}{\min_{e \in E} u_e} = SP^+ + \frac{B^+}{\min_{e \in E} u_e}.$$

□

We call an instance of the Quickest Transshipment Problem **balanced** if there exists a finite time horizon such that all supplies and demands can be balanced. We have seen, that we can easily

detect balanced instances and Observation 2.8 gives an upper bound for the time horizon. For implementation purposes, the existence of reasonably good lower and upper bounds is enough. The decision problem belonging to the Quickest Transshipment Problem raises the question whether a given time horizon allows for a feasible transshipment.

Problem:	Transshipment over Time
Instance:	A dynamic network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$, supplies and demands for the sources and sinks $b : V \to \mathbb{R}$, and a time horizon T.
Task:	Decide whether there exist a transshipment over time satisfying the demands and supplies in the given time horizon T, or not.

A condition to detect whether an instance is not only balanced, but also feasible for a given time horizon $T < \infty$ has first been observed by Klinz [Kli94]. To explain the condition, we need to introduce an abbreviation for the maximum flow that can be sent from a subset of terminals to the others. Let $b(M) \coloneqq \sum_{s \in M} b_s$ be the total supply of a subset $M \subseteq S^+ \cup S^-$. The maximum amount of flow that can be sent from sources in $S^+ \cap M$ to sinks in $S^- \setminus M$ is denoted by $o(M)$. The following Theorem gives the feasibility condition by Klinz. A similar observation has been made in the continuous flow model by Fleischer and Tardos [FT98].

Theorem 2.9 (Klinz [Kli94]). *An instance $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ with supplies and demands $b : V \to \mathbb{R}$ has a feasible solution within time horizon T if and only if $o(A) > b(A)$ for every subset $A \subset V$.*

On the basis of this observation Hoppe and Tardos [HT94] have developed a strongly polynomial algorithm for the Transshipment over Time Problem and also subsequently for the Quickest Transshipment Problem. They use generalized temporally repeated flows to show that polynomial size solutions exists. However, the algorithm strongly relies on submodular function minimization and again on Megiddo's parametric search framework. Due to this rather complicated methods no implementation is known so far and most likely an implementation will never be made. For practical purposes, algorithms working in the time-expanded networks are good enough.

2.3.1 Arc Release Dates and Deadlines

Hoppe also studied a special case of transshipments that allow arcs to have release dates and deadlines [Hop95]. Hoppe calls arcs with such restrictions *mortal* arcs. An arc cannot be used by flow before its release date and after its deadline.

Definition 2.10. Arc **release dates** and **deadlines** are two functions $r : E \to \mathbb{R}_{\geq 0}$ and $d : E \to \mathbb{R}_{\geq 0}$, respectively, that define the time an arc is available. A flow over time f is a flow that **respects release dates and deadlines** if

$$f(e, \theta) = 0 \qquad \text{for all } \theta < r_e \text{ and } \theta \geq d_e - \tau_e.$$

◁

Without loss of generality, we will assume $0 \leq r_e < d_e \leq T$. If the deadline is not larger than the release date, we can just erase the arc and if release date and deadline are too early, or too late, respectively, it just does not change the flow value compared to a flow without these requirements. Within this setting we define the corresponding problems for the Maximum Flow over Time Problem and the Quickest Transshipment Problem.

Problem:	Maximum Flow over Time Problem with Release Dates and Deadlines
Instance:	A dynamic network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$, time horizon T, release dates $r : E \to \mathbb{R}^+$ and deadlines $d : E \to \mathbb{R}^+$.
Task:	Find a flow over time f respecting release dates and deadlines with maximal value $\lvert f \rvert$.

If $d \equiv T$ and $r \not\equiv 0$ we also speak of the Maximum Flow over Time Problem with Release Dates, and if $r \equiv 0$ and $d \not\equiv T$ we speak of the Maximum Flow over Time Problem with Deadlines.

If the release dates $r_e \equiv 0$ are zero and the deadlines $d_e \equiv T$ are not restricting the flow, the problem reduces to the Maximum Flow over Time Problem.

Problem:	Quickest Transshipment Problem with Release Dates and Deadlines
Instance:	A dynamic network $\mathcal{N} = (G, u, \tau, S^+, S^-)$, supplies and demands for the sources and sinks $b : V \to \mathbb{R}$, non-negative release dates $r : E \to \mathbb{R}^+$ and deadlines $d : E \to \mathbb{R}^+$.
Task:	A transshipment over time with minimal time horizon T respecting release dates and deadlines.

Notice, that the Quickest Transshipment Problem with release dates and deadlines can be solved in a time-expanded network. For an arc e, we just have to remove the copies $e_1, \dots, e_{\ell_e - 1}$ and $e_{d_e}, \dots, e_T$. However, this solution is not preferable from the theoretical point of view as it requires pseudo-polynomial running time.

Hoppe [Hop95] observed that instances of flow over time problems with release dates and deadlines can be reduced to instances without release dates and deadlines by introducing (a small number of) additional pairs of sources and sinks. The term "a small number" refers to the fact that the introduction of the additional nodes and arcs does not increase the network size too much, such that it remains polynomial in the input size, in contrast to time expansion.

We briefly describe the reduction and prove that the Maximum Flow over Time Problem with release dates and deadlines and the Quickest Transshipment Problem are in fact equivalent. Note that the reduction necessarily relies on the possibility of (unlimited) holdover. Hoppe pointed

out that already Klinz observed that the problem becomes $\mathcal{NP}$-hard if waiting in intermediate nodes is not allowed [Kli94].

We show that it is possible to remove all mortal arcs by introducing few new vertices and additional arcs. Thus, we create a new network which allows to compute a network flow of equal value on all sources and sinks in the original network. The arcs are replaced by the gadget depicted in Figure 2.3b. More formally, we construct a new network $\overline{\mathcal{N}}$ as follows.

Definition 2.11. Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network, and r_e and d_e release dates and deadlines for arcs $e \in E$. We define the corresponding dynamic network without release dates and deadlines $\overline{\mathcal{N}} = (\overline{G} = (V, \overline{E}), \overline{u}, \overline{\tau}, \overline{S^+}, \overline{S^-}, T)$ as follows.

$$\overline{V} \coloneqq V \cup \{s_e, t_e, v', w' \mid E = (v, w) \in E\},$$

$$\overline{E} \coloneqq \{(v, v'), (v', t_e), (w', v'), (s_e, w'), (w', w) \mid e = (v, w) \in E\},$$

$$u_{(v,v')} \coloneqq u_{(v',t_e)} \coloneqq u_{(w',v')} \coloneqq u_{(s_e,w')} \coloneqq u_{(w',w)} \coloneqq u_e \qquad \forall e = (v, w) \in E,$$

$$\tau_{(v,v')} \coloneqq 0, \qquad \tau_{(w',v')} \coloneqq 0,$$

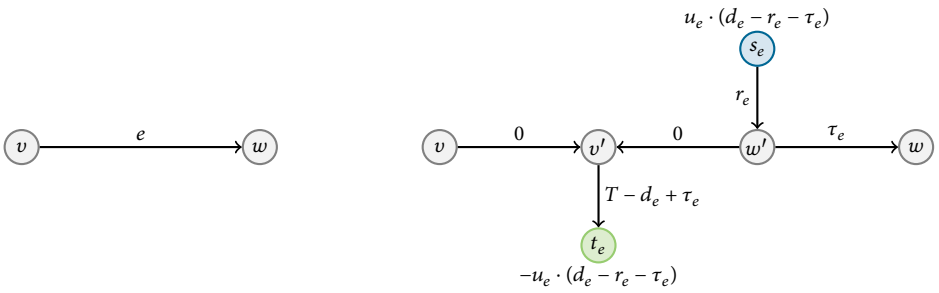
$$\tau_{(v',t_e)} \coloneqq T - d_e + \tau_e, \qquad \tau_{(s_e,w')} \coloneqq r_e, \qquad \tau_{(w',w)} \coloneqq \tau_e \qquad \forall e = (v, w) \in E,$$

$$\overline{S^+} \coloneqq S^+ \cup \{s_e \mid e \in E\},$$

$$\overline{S^-} \coloneqq S^- \cup \{t_e \mid e \in E\},$$

$$b_v \coloneqq \begin{cases} u_e \cdot (d_e - r_e - \tau_e) & v = s_e, \\ -b_{s_e} & v = t_e, \\ b_v & v \in V, \\ 0 & v' \in \overline{V} \setminus (V \cup \overline{S^+} \cup \overline{S^-}). \end{cases}$$

◁



(a) An arc with release date r_e, deadline d_e, and transit time τ_e.

(b) A gadget adding a new pair of source and sink to replace the release date and deadline as described in Definition 2.11.

Figure 2.3: The construction used to reduce instances with release dates and deadlines to instances without mortal edges. Each arc $e(v, w)$ is replaced by the given gadget.

We show that for any given flow in the original network $\mathcal{N}$ we can construct a flow with the same value on the original terminal nodes that also satisfies all supplies and demands of the additional nodes. Also, we see that from any flow in $\overline{\mathcal{N}}$ that satisfies the demands for all terminal nodes and that we can construct a flow in the original network that has the same value.

Lemma 2.12. *Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network with arc release dates r_e and deadlines d_e and f be a feasible flow in $\mathcal{N}$. Then there exists a flow f' in $\overline{\mathcal{N}}$ with the same value $|f| = |f'|$. If f does not require waiting in intermediate nodes, then also f' does not require waiting in intermediate nodes.*

Proof. We define f' by setting the value for each arc $e = (v, w)$ as follows.

$$
\begin{aligned}
f'((v, v'), \theta) &:= f(e, \theta) \\
f'((v', t_e), \theta) &:= \begin{cases} u_e & \theta \in [r_e, d_e - \tau_e[\\ 0 & \theta \in [0, r_e[\cup [d_e - \tau_e, T[\end{cases} \\
f'((s_e, w'), \theta) &:= \begin{cases} u_e & \theta \in [0, d_e - r_e - \tau_e[\\ 0 & \theta > d_e - r_e - \tau_e \end{cases} \\
f'((w', v'), \theta) &:= \begin{cases} u_e - f(e, \theta) & \theta \in [r_e, d_e - \tau_e[\\ 0 & \theta \in [0, r_e[\cup [d_e - \tau_e, T[\end{cases} \\
f'((w', w), \theta) &:= f(e, \theta)
\end{aligned}
$$

The capacities of f' are feasible and lie within the interval $[0, u_e]$. Because $f'((v, v'), \theta) = f(e, \theta)$ and $f'((w', v), \theta) = f((v, w), \theta)$, flow conservation in v and w carries over from f.

All arcs have flow value 0 before the release date and after the deadline. We have flow conservation at v' for $\theta \leq d_e$ because

$$
\begin{aligned}
&\int_0^\theta f'((v, v'), \xi)\, d\xi + \int_0^\theta f'((w', v'), \xi)\, d\xi \\
&= \int_{r_e}^{\theta - \tau_e} f((v, v'), \xi)\, d\xi + \int_{r_e}^{\theta - \tau_e} u_e - f((w', v'), \xi)\, d\xi \\
&= \int_{r_e}^{\theta - \tau_e} u_e\, d\xi = \int_0^\theta f'((v', t_e), \xi)\, d\xi.
\end{aligned}
$$

For node w' we have

$$
\begin{aligned}
&\int_0^\theta f'((w', v'), \xi)\, d\xi + \int_0^\theta f'((w', w), \xi)\, d\xi \\
&= \int_{r_e}^\theta u_e - f(e, \xi)\, d\xi + \int_{r_e}^\theta f(e, \xi)\, d\xi \\
&= \int_{r_e}^\theta u_e\, d\xi = \int_0^{\theta - r_e} f'((s_e, w'), \xi)\, d\xi,
\end{aligned}
$$

thus flow conservation holds on all nodes which shows the statement of the lemma. □

We will show a similar lemma for the opposite direction, i. e., creating a feasible flow from a given flow in the extended network $\overline{\mathcal{N}}$. However, in this case it is not possible to construct a flow that does not need waiting in intermediate nodes. This is because it is possible that another source than s_e sends flow to t_e in $\overline{\mathcal{N}}$. Consider the replacement of an arc $e = (v, w)$. The capacities on the new source and sink and the transit time allow us to limit the flow through the gadget to the same maximum value $u_e \cdot (d_e - r_e - \tau_e)$, however a feasible flow in $\overline{\mathcal{N}}$ can use the arc from the beginning and after the deadline. An example of such a situation is depicted in Figure 2.4. Because the total flow through the construction is limited by the actual flow value we construct feasible flows that are "drawn out". In the following lemma we show that it is possible to "compress" the flow again and use the arc only at valid times by storing early flow for later usage.

Lemma 2.13 ([Hop95]). *Let f' be a feasible flow in the network $\overline{\mathcal{N}}$. Then, there exists a feasible flow in the dynamic network $\mathcal{N}$ with release dates r_e and deadlines d_e that may require waiting in intermediate nodes regardless of whether f' requires waiting in intermediate nodes.*

Proof. We define the new flow f by setting

$$f(e, \theta) \coloneqq u_e - f'((w', v'), \theta)$$

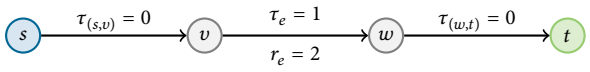
for all arcs $e = (v, w) \in E$ and times $\theta \in [0, T[$. Obviously, f respects capacities. Also, observe that the total amount of flow on e satisfies $\int_0^T f(e, \xi)\, d\xi = \int_0^T f'((v, v'), \xi)\, d\xi = \int_0^T f'((w, w'), \xi)\, d\xi$. To show that (weak) flow conservation holds, we have to show that at each point in time the amount of flow assigned to e is not more than the amount that leaves v to v' in f' and at least as much as enters w from w'. Before time r_e, there is no flow on e in f, and also no flow entering node w, so flow conservation holds. Consider now a point in time $\theta > r_e$.

$$\begin{aligned}
&\int_0^\theta f'((v, v'), \xi)\, d\xi = \int_{r_e}^\theta f'((v, v'), \xi)\, d\xi \\
\geq &\int_{r_e}^\theta u_e - \int_{r_e}^\theta f'((w', v'), \xi)\, d\xi = \int_{r_e}^\theta u_e - f'((w', v'), \xi)\, d\xi \\
\geq &\int_{r_e}^\theta f'((w', w), \xi)\, d\xi = \int_0^\theta f'((w', w), \xi)\, d\xi.
\end{aligned}$$

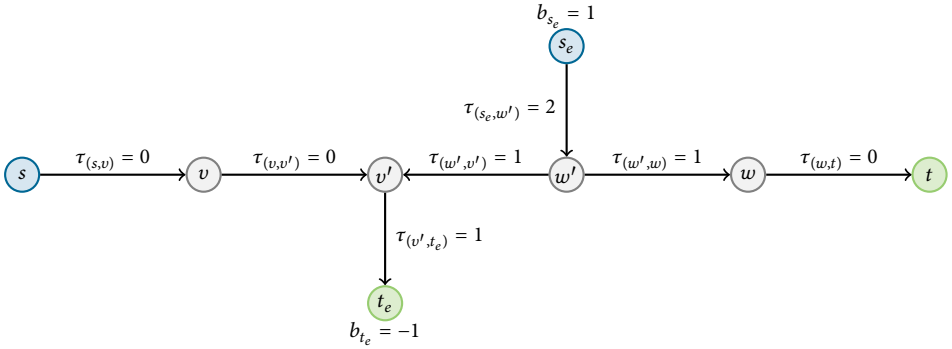
Observe that the statement follows because the second equation equals exactly the flow on e in f by definition. At time T, we have equality because both flows send the same amount of flow in total and thus flow conservation holds. Notice, that we only have weak flow conservation due to the inequalities in the above calculation. Strict flow conservation is violated if and only if $f((v, v'), \theta) \neq 0$ for some $\theta < r_e$. □

With these two lemmas we can show that it is possible to compute a maximum flow over time with release dates and deadlines in a network with supplies and demands. We include the construction in a search framework to find the exact value of the maximum flow over time.

Theorem 2.14. *Let $\mathcal{N} = (G = (V, E), u, \tau, s, t)$ be a dynamic network and T a time horizon. The* Maximum Flow over Time Problem *with arc release dates and deadlines can be solved in (strongly) polynomial time. The resulting maximum flow may require waiting.*



(a) Within time horizon $T = 4$ one unit of flow can be sent.



(b) A feasible flow in the network containing the gadget may send flow on path (s, v, v', t_e) at time zero *before* the release date $r_e = 2$.

Figure 2.4: An example of the computation in Lemma 2.13 that may require waiting. The network containing edge $e = (v, w)$ with positive release date consists of only one path (s, v, w, t).

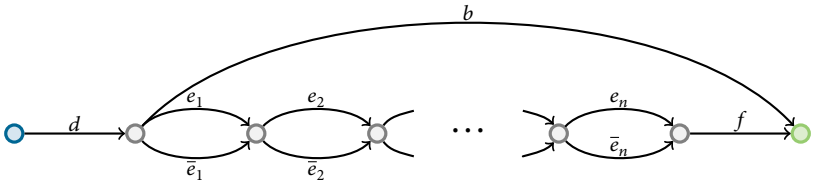
Proof. Create a network $\overline{\mathcal{N}}$ by replacing each arc with the structure defined in Definition 2.11. For an arbitrary $b \in \mathbb{N}$ we assign supplies and demands $b_s := b$ and $b_t := -b$ to the original source and sink, respectively. It is possible to send b units of flow in the original network if the quickest transshipment in $\overline{\mathcal{N}}$ has a time horizon of at most T. The flow can be transformed back using Lemma 2.13. Trying different values for b within a binary search framework solves the maximum flow problem. The algorithm has strongly polynomial running time if Megiddo's framework for parametric search is used. □

The proof of Lemma 2.13 shows, that the computed flow over time may require waiting in intermediate nodes. Accordingly, the maximum flow coming out from Theorem 2.14 uses waiting. It is not possible to find a maximum flow that does not use waiting in polynomial time, unless $\mathcal{P} \neq \mathcal{NP}$.

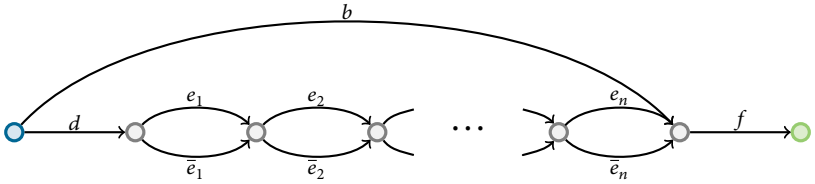
Theorem 2.15 ([Kli94]). *If waiting in intermediate nodes is not allowed, the* Maximum Flow over Time Problem *with deadlines and release dates is $\mathcal{NP}$-hard.*

Proof. We reduce the flow over time problem with release dates to Partition if waiting is not allowed. Let $\{a_1, a_2, \dots, a_n\}$ be a partition instance. For the reduction we create a network as depicted in Figure 2.5a. The instance consists of $n+1$ nodes $v_1, v_2, \dots, v_{n+1}$ and consecutive nodes v_i and v_{i+1} are connected with two parallel edges e_i and $\bar{e}_i$. There is an additional source node s and a sink t. These nodes are connected with the other nodes via the arcs $d = (s, t)$, $b = (v_1, t)$ and $f = (v_{n+1}, t)$. All arcs have unit capacity. For $n \in \{1, 2, \dots, n\}$ we set $\tau_{e_k} := a_n$ and $\tau_{\bar{e}_k} := 0$. The other arcs have zero travel time. All arcs except b and f have no release date and are immediately available. For the remaining arcs we set $r_f := T$ and $r_b := 1$.

A solution of Partition is encoded by a path using either arcs e_k or $\bar{e}_k$. We use the interpretation that flow using e_k means taking element a_k into the solution and using $\bar{e}_k$ means that a_k is not taken.



(a) Reduction with release dates. All arcs are present at time 0 except $r_b = 1$ and $r_f = T$.



(b) Reduction with deadlines. All arcs are available until the time horizon $T + 1$ except $d_b = T$ and $d_d = 1$.

Figure 2.5: The graphs used by the reduction of the Maximum Flow over Time Problem with release dates or deadlines to partition. All capacities are unit capacities, and travel times are $\tau_{e_n} := a_n$, and $\tau_{\bar{e}_n} := \tau_b := \tau_d := \tau_f := 0$.

The partition instance is a `yes`-instance if and only if there exist a path using arcs e_k and $\bar{e}_k$ with travel time $T := \frac{\sum_{i=1}^{n} a_i}{2}$.

For each time $\theta > 1$ flow can be sent along (d, b). In the interval $[0, 1[$ flow can be sent along another path using arc f if and only if it arrives at tail(f) at time T, when the edge becomes available. Thus, within a time horizon of $T + 1$, we can always send T units of flow along path (d, b). Another flow unit can be sent if and only if the partition instance is a `yes`-instance.

Figure 2.5b shows a similar graph used to reduce the flow over time problem with deadlines to Partition if waiting is not allowed. Arc d limits the inflow in the lower paths representing the partition instance. Arc b can be used to send T units of flow up to time T. The additional unit of flow can be sent if and only if the partition instance is a `yes`-instance because then arc f can be used in the time interval $[T, T + 1[$. If the instance is a `no`-instance flow arrives to early at f and would block flow coming along arc b. □

2.4 Earliest Arrival Flows

The Maximum Flow over Time Problem as defined in Section 2.1 and the Quickest Transshipment Problem as defined in Section 2.3 can both be generalized in the following way. For the maximization variant, we are interested in computing a flow over time that sends as much flow as possible within a given time horizon T. For the variant with supplies and demands, we are interested in finding a minimum time horizon to satisfy them. Both problems focus only on the result at the time horizon. However, we can additionally ask for a flow that sends flow as early as possible in addition to the other requirements. We say that a flow over time has the *earliest arrival* property if

as much flow as possible arrives at the sinks at each time $\theta < T$. For the Quickest Transshipment Problem we can limit ourself to satisfy as much of the total demand as possible.

Definition 2.16 (Earliest Arrival Pattern). Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network, and f be a feasible network flow that respects supplies and demands if b_v is given.

A **pattern** is a function $p : [0, T[\to \mathbb{R}_{\geq 0}$ in the continuous setting and $p : \{1, 2, \dots, T\} \to \mathbb{R}_{\geq 0}$ in the discrete setting. A flow has the **arrival pattern** p_f if

$$p_f(\theta) = |f|_\theta \qquad \text{for } \theta \in [0, T[, \text{ or } \theta \in \{1, 2, \dots, T\}, \text{ respectively.}$$

Let f^*_θ be a maximum flow with time horizon θ, possibly subject to supplies and demands if b_v is given. The **earliest arrival pattern** for $\mathcal{N}$ is then defined as

$$p^*(\theta) \coloneqq |f^*_\theta| \qquad \text{for } \theta \in \{1, 2, \dots, T\}, \text{ or } \theta \in [0, T[.$$

Observe that two flows f^*_θ and $f^*_{\theta'}$ may be completely different for $\theta \neq \theta'$. In particular, for $\theta' > \theta$ it is totally possible to have $f^*_\theta(e, \xi) \neq f^*_{\theta'}(e, \xi)$ for an arc $e \in E$ (for any of the first time steps $\xi \leq \theta$).

A network flow over time f for a given time horizon T is an **earliest arrival flow** if its arrival pattern equals the earliest arrival pattern, e. g., $|f|_\theta = p_f(\theta) = p^*(\theta)$ holds simultaneously for all points in time. A dynamic transshipment adhering to the earliest arrival pattern is an **earliest arrival transshipment**. ◁

2.4.1 Single-commodity Earliest Arrival Flows

The requirements for a flow to have the earliest arrival property is quite high, as it should meet the pattern at *all* points in time. Flows having the earliest arrival property are also known as **universally maximal flows** or **universally quickest flows**. Existence of earliest arrival flows in the case of a single source and a single sink has first been observed by Gale [Gal59] and Philpott [Phi90] in the discrete and continuous setting, respectively. Gales proof was non-constructive and relied on the Supply-and-Demand-Theorem also proven by Ford and Fulkerson [FF62]. First pseudo-polynomial algorithms have been presented by Minieka [Min73] and Wilkinson [Wil71].

We discuss maximum earliest arrival flows now and earliest arrival transshipments after that. Notice that there are also variations of earliest arrival flows that we do not discuss. So-called *latest departure flows* satisfy a similar property as earliest arrival flows, but this property regards the departure time. For single source-single sink networks, there exist flows that have both the earliest arrival and the latest departure property. A relaxed version of the earliest arrival property is proposed by Stiller and Wiese [SW10]: *Multiple deadline flows* require only that the earliest arrival pattern is met for some points in time.

Problem:	Maximum Earliest Arrival Flow
Instance:	Dynamic network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ with transit times $\tau \geq 0$ and a time horizon T.
Task:	Compute a maximum flow over time f with time horizon T such that the flow pattern $p_f = p^*$ is an earliest arrival flow.

The Maximum Earliest Arrival Flow Problem can be solved by computing a static lexicographically maximal flow in the time-expanded network [Min73]. As lexicographical order of sinks, the copies of the original sink are taken for each time point $\{1, 2, \dots, T\}$. More formally, let t^θ and $t^{\theta'}$ be two copies of a sink at times θ and θ'. Then t^θ has higher priority than $t^{\theta'}$ if and only if $\theta < \theta'$. The resulting flow can then easily be transformed into a dynamic flow. An earliest arrival flow in the time-expanded network relates to a minimum cost flow by Lemma 2.5. To see this, we take transit times as costs and notice that flow arriving earlier incurs less total costs. In an earliest arrival flow, at each point in time the maximum amount of flow arrives and it is not possible to send flow earlier which would reduce costs even further. Thus, any earliest flow with value $|f|$ equals a minimum cost flow of the same value in the time-expanded network.

The lexicographical order and the minimum cost approach can be combined to get the Earliest Arrival Flow Algorithm 2.2. It uses the Successive Shortest Paths Algorithm 1.1 to send flow units. Minieka [Min73] first observed that it is possible to compute this flow also without using time expansion. The algorithm runs the Successive Shortest Paths Algorithm on the residual network belonging to the original graph and sends flow along the paths in a temporally repeated way, but using backward arcs. Minieka was the first to show an earliest arrival flow can be computed in the original network by using a generalized path decomposition.

Algorithm 2.2: Maximum Earliest Arrival Flow Algorithm

Input: $\mathcal{N} = (G = (V, E), u, \tau, s, t)$ with transit times $\tau \geq 0$ and a time horizon T.

Output: A generalized temporally repeated flow over time with time horizon T.

1. Initialize f be the zero flow in the static network $\mathcal{N} = (G, u, s, t)$ and let x_P be the corresponding generalized path decomposition.
2. Compute a shortest s-t-path P in the residual network $\mathcal{N}_f$ where the arc lengths are equal to the transit times. If $\tau(P) \geq T$, return the generalized temporally repeated flow according to path flow x.
3. Let δ be the residual capacity of P. Increase $x_P := x_P + \delta$, and update f accordingly. Continue with 2.

Algorithm 2.2 is executed on the original network, however it still does not become polynomial due to the pseudo-polynomial running time of Algorithm 1.1. This also raises doubt whether a polynomial algorithm for Earliest Arrival Flow can exist because the earliest arrival pattern may contain exponentially many break points and thus would be even hard to store in polynomial time. $\mathcal{NP}$-hardness was finally shown by Disser and Skutella [DS15]. They show that it is $\mathcal{NP}$-hard to obtain the average arrival time of flow of an earliest arrival flow. Notice, that an s-t-flow minimizes its average arrival time if and only if it is an earliest arrival flow [JR82].

The early algorithms work in the discrete setting and the existence of similar continuous algorithms with the same running time has been shown by Fleischer and Tardos [FT98]. An FPTAS for the Maximum Earliest Arrival Flow Problem is due to Hoppe and Tardos [HT94].

Earliest Arrival Transshipments. Notice that any earliest arrival transshipment necessarily also is a quickest transshipment. We will now consider the case with bounded supplies and demands in the sinks, which makes the problem harder in the following sense: It is not true any more in this

setting, that earliest arrival flows exist necessarily. By non-existence we mean that there is no single flow over time that meets the earliest arrival pattern for all points in time simultaneously. This gives rise to the question of approximating the feasibility, e. g., to ask for a flow over time that is as good as possible and is not "too far off" of the pattern. We will answer this question in Chapter 4. In particular, if multiple sinks are present, we are only interested in the total amount of flow that has arrived at the sinks but not at which specific sink flow has arrived.

Problem: Earliest Arrival Transshipment

Instance: A dynamic network $\mathcal{N} = (G, u, \tau, S^+, S^-)$, supplies and demands for the sources and sinks $b : V \to \mathbb{R}$.

Task: Compute a transshipment over time sending all supplies from sources such that all demands are satisfied and the resulting transshipment over time satisfies $|f|_\theta = p(\theta)$ for $\theta \in [0, T^*[$, where T^* is the minimum necessary time horizon.

If all the transit times $\tau \equiv 0$ are zero, we speak of the Earliest Arrival Transshipment with zero travel times.

Algorithms for the case of a single sink and a single source can be built on the aforementioned algorithms for the maximization case. The successive shortest path algorithm can be executed with an increasing time horizon until the demands are satisfied by the computed solution. The case with multiple sources *and* sinks is considered in Chapter 4 since the existence of earliest arrival transshipments is no longer guaranteed. The case with multiple sources and a single sink, however, still allows for earliest arrival transshipments.

An earliest arrival transshipment cannot be computed in the static network because the supplies are limited and any temporally repeated flow will exceed the supplies eventually. However the technique to use successive shortest paths from Algorithm 2.2 can be applied in a time-expanded network. Richardson and Tardos [RT02] also observed that Minieka's existence proof based on lexicographically maximal flows can be extended to networks with several sources and a single sink with appropriate demands. The problem with the limited capacities dissolves in this case, because the arc capacities of the super sources in Definition 2.4 limit the inflow into each source.

Algorithm 2.3: Earliest Arrival Transshipment

Input: $\mathcal{N} = (G, u, \tau, S^+, t)$ with transit times $\tau \geq 0$, balances b_v, and a single sink t.

Output: An earliest arrival transshipment f.

1. Build the time-expanded network $\mathcal{N}^T$ with supplies and demands with a sufficiently large time horizon $T \geq T^*$, e. g., by using the bound from Observation 2.8.
2. Compute a min cost flow with respect to transit times as costs in $\mathcal{N}$ using the Successive Shortest Path Algorithm 1.1.
3. Return the flow over time corresponding to the static flow constructed by the procedure from Lemma 2.5.

Baumann and Skutella [BS09] give an algorithm that computes an earliest arrival transshipment in running time bounded by the size of the input and *output* size. However, the algorithm heavily relies on submodular function minimization and is thus not practically applicable. Practical implementations, such as we consider in Chapter 3, rely on time expansion. An algorithm running in polynomial time in the time horizon and the total supply at the sources is due to Tjandra [Tja03]. Fleischer and Skutella [FS07] present an FPTAS that sends the maximal amount of flow possible at time θ a bit later at time $(1+\varepsilon)\theta$ for each time $\theta \geq 0$. For the case of series-parallel graphs, Ruzika, Sperber and Steiner [RSS11] present a polynomial algorithm.

Zero Travel Times. The Earliest Arrival Transshipment Problem becomes easier in the case of zero travel times. As such we consider instances for which $\tau_e \equiv 0$ for all arcs. It is possible to compute an earliest arrival transshipment with zero travel times in polynomial time [HO84]. Fleischer gives an algorithm with improved running time [Fle01].

2.4.2 Multi-commodity Earliest Arrival Flows

Multi-commodity earliest arrival flows have not been studied so far, to the best of the authors knowledge. In the Multi-commodity Earliest Arrival Transshipment Problem we again take the point of view motivated by the evacuation problem and try to maximize the total flow sent until a given point in time. For this purpose, we do not weight commodities or enforce any precedence constraints.

Problem:	Multi-commodity Earliest Arrival Transshipment
Instance:	A dynamic network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$, together with a set of commodities K.
Task:	A multi-commodity flow over time f sending the maximal possible amount of flow $\lvert f\rvert_\theta = p^*(\theta)$ for each $\theta \in [0, T^*[$, or $\{1, 2, \dots, T^*\}$ where T^* is the optimal time horizon of a quickest transshipment.

Multi-commodity Earliest Arrival Transshipments. Before we prove that multi-commodity arrival transshipments exist in the case of a single sink, we observe that it is possible to reduce the multi-commodity case to single commodities in the single source-single sink case. If all commodities start in the same source and have the same sink as destination, we can simply combine the commodities to one big commodity, compute an earliest arrival transshipment and split the flow up into the commodities. The same splitting technique then provides the algorithm for multiple sinks.

Observation 2.17. *Let $i \neq i' \in K$ be two commodities with the same sources $s_i = s_{i'}$ and sinks $t_i = t_{i'}$. The commodities can then be combined to a new commodity $\hat{\imath}$ and any feasible flow on the new instance with commodities $K \setminus \{i, i'\} \cup \hat{\imath}$ can be transformed to a feasible flow in the original instance and vice versa.*

Thus, all instances which only use different commodities on the same pair of sources and sinks can be solved using any algorithm solving the Earliest Arrival Transshipment Problem by re-

duction. Without loss of generality we now assume that multiple commodities use different source and sink pairs. Notice that the simple example with three commodities on the network depicted in Figure 4.10 shows that earliest arrival transshipments do not necessarily exist in the case of multi-commodity flows with multiple sinks, even if the transit time is zero. However, if the commodities use the same single sink but different sources, we can use Algorithm 2.3 to solve instances by combining the commodities before the earliest arrival flow computation and splitting the flow afterwards. This leads to the following algorithm.

Algorithm 2.4: Multi-commodity Earliest Arrival Transshipment Algorithm

Input: Dynamic network $\mathcal{N} = (G, u, \tau, S^+, t)$ with a single sink t and commodities K.

Output: A multi-commodity earliest arrival transshipment f.

1. Define new supplies and demands b' by

$$b'_v := \begin{cases} \sum_{i \in K} b_{i,v} & \text{if } v \text{ source in } \bigcup_{i \in K} S_i^+, \\ \sum_{i \in K} b_{i,t} & \text{if } v = t \text{ is the sink,} \\ 0 & \text{otherwise.} \end{cases}$$

2. Compute an earliest arrival transshipment for the instance defined by the network $\mathcal{N} = (G, u, \tau, \bigcup_{i \in K} S_i^+, t)$ with path decomposition x using Algorithm 2.3.

3. For each commodity $i \in K$ split the path flow into commodity dependent paths:
   - Select a source $s \in S_i^+$ with positive supply $b_{i,s} > 0$,
   - let P be an s-t-path with positive flow value $x_P > 0$,
   - set flow value $x'_{i,P} := \min\{x_P, b_{i,s}\}$,
   - update $b_{i,s} := b_{i,s} - x'_{i,P}$ and $x_P := x_P - x'_{i,P}$,
   - continue until no source with positive supplies in S_i^+ exists any more.

4. Return edge flow f belonging to x'.

Theorem 2.18. *In dynamic networks with non-negative travel times, $k \geq 1$ commodities and a single sink earliest arrival transshipments exist. They can be computed by Algorithm 2.4 with a running time that is polynomial in k and the size of an earliest arrival transshipment in the same network in which all commodities are combined to one commodity.*

Proof. We show that the construction in step 3 of Algorithm 2.4 creates a feasible solution of the Multi-commodity Earliest Arrival Transshipment Problem. By definition of the new demands b'_v for each source, we have $\sum_{P \in \mathcal{P}^{st}} x_P = \sum_{i \in K} b_{i,v}$. The definition of $x'_{i,P}$ as minimum makes sure that no path's capacity is exceeded and no more supply is sent than possible. Thus, for any path P, the flow values are reduced until the remaining capacities are 0 and the constructed flow respects capacities due to $\sum_{i \in K} x'_{i,P} = x_P$. Hence, $x_{i,P}$ is a feasible flow satisfying all demands.

The constructed flow is an earliest arrival flow in the single-commodity setting. Thus, there cannot

be a multi-commodity flow sending more flow at any time, as such a flow would induce a single-commodity flow of higher value.

The algorithm computes a path decomposition of an earliest transshipment in $\mathcal{N}$. For each of the commodities some paths are selected to define a path flow x', which is possible in polynomial time in the size of the path decomposition. Finally, an edge flow is built and returned. □

Flow Dependent Transit Times. An interesting example for evacuation modelling and for traffic modelling in general is the usage of *flow dependent transit times*. They model the fact that the speed on a path, i. e., a lane of a street or a crowded room, becomes slower when the amount of flow units gets higher [CFS82]. It is computationally challenging to compute such flows. However, they can be approximatively modelled in networks in the following way: A single arc with flow dependent transit time is replaced by several arcs with constant travel time. The capacities of the new arcs are smaller than the capacity of the original arc which is equal to the sum of travel times. The travel times of the new arcs are increasing. Such a construction forces flow to use the fast arc first to arrive quicker. The more flow uses the arc, the more of the longer arcs have to be used, thus approximating the effect of time depending travel times. The technique was first described by Köhler, Langkau, and Skutella [KLS02] and is referred to as *bow graph*. Note that this model may lead to undesired effects, such that flow using an arc copy with short travel time later may overtake flow units taking the slower arc copy.

3 Evacuation Simulation and Optimization

The two research areas of evacuation *simulation* and evacuation *optimization* have so far only been considered separately: Tools used in practice only provide simulation algorithms whereas approaches based on network flows over time have been mostly evaluated in theoretical research only. In the following we describe a simple conversion framework to generate different network flow instances from rasterized building geometry automatically. Such rasterized data can be derived from the cells of a *cellular automaton*. Apart from that we combine simulations and optimized network flows, such as *earliest arrival transshipments*, to further improve evacuation results. We provide choices for exit assignments of evacuees within a building to improve their egress time and we validate the automatically generated model by comparing it to a test evacuation of a 22-storey office building.

Publication Remark: The results in Section 3.2 are joint work with Daniel Dressler, Martin Groß, Timon Kelter, Joscha Kulbatzki, Daniel Plümpe, Gordon Schlechter, Melanie Schmidt, Martin Skutella, and Sylvie Temme and have been published in [DGK+10]. The results of Section 3.3 have been presented at the 2nd International Conference on Evacuation Management 2012 (without proceedings).

Coping with disaster scenarios as well as possible has always been an important task. This includes answering the question how long it will take until all people residing in an endangered area are safe and optimizing the evacuation process as well as evacuation management. It was common in the pre-digital era, that calculation and optimization of egress times have been performed by experts using a lot of expertise, intuition and experience. Since then, computer simulation models have been developed and are widely used in practice nowadays. Many of the existing software tools are commercial, but there also exist free and open source software tools. Two of these are MATSim [MAT14] and ZET [zet14], both of which are well suited for academic usage and research. While MATSim is specifically designed for large scale[1] evacuation situations, ZET is especially designed for modelling building evacuations. Existing approaches for building evacuation simulation (and similarly for nautical vessels and other comparatively small structures) employ *microscopic models*. Such models allow a very detailed modelling of both the structure of the situation and also of the behaviour of the evacuees. Due to their size, large scale evacuations usually are simulated in a *macroscopic setting* which broadly abstracts from individual evacuees and detailed structures.

In parallel to the development of better simulation models, combinatorial models to address evacuations have evolved. Those models finally led to the theory of network flows over time and especially earliest arrival flows as described in Chapter 2. Network flow models have been applied to realistic building evacuation scenarios occasionally [Fra81; CFS82; CHT88; HT01; Tja03]. These works employ network flows primarily in simple models that largely abstract from the actual building

[1] By large scale we mean for example the evacuation of a city with thousands of evacuees.

structure. The results show that network flows can be used to get meaningful results for evacuation scenarios even if they abstract from many specific individual properties such as different walking speeds. However, in the mentioned works much effort was put into setting up the network structure and parameters such as transit times.

So far optimization approaches using network flows (over time) and simulations were developed in parallel and have not been combined. However, a combination of both approaches can be fruitful in different ways. While simulation results mainly show expected behaviour (if the model is well calibrated), we hope to derive decisions to improve egress times from optimization results. If an evacuation situation is modelled by a dynamic network, simulation results can also be used to automatically compute transit times on arcs. The authors of [CFS82] include a detailed discussion how the correct transit times and capacities on arcs modelling staircases should be set. Such decisions can be made automatically without complicated reasoning by using results of pedestrian simulation if the simulation model is well calibrated. Finally, simulations can be used to compare quality of network-based decisions.

Exit Distributions. The first and most basic decision in an evacuation scenario that can be influenced is the exit decision. Most existing evacuation plans give predefined evacuation routes, sometimes more than a single one for a given room. Depending on the building structure, exit capacity and utilization, optimal exit choices may differ. We develop two new strategies based on network flows to improve egress time due to better exit choices and compare the results by using a cellular automaton simulation algorithm. We compare this solution the most basic exit assignment based on shortest paths. To complement the results with another approach we also employ a game theory based strategy to derive exit assignments.

Automatic Model Generation. To be useful in practice, e. g., for implementation in existing software tools, it must be possible to generate appropriate network flow structures for buildings (or larger scenarios) automatically. The network model should be precise enough to generate meaningful results but should on the other hand not be too detailed. In this case the network sizes increase significantly if time expansion is applied and it is easy to reach the limits by means of running time and required memory space even on modern machines. An early model by Chalmet, Francis and Saunders [CFS82] uses only very few nodes due to this limitations. More detailed models use one node for each room connected via arcs going through doors. While one node can be appropriate for some rooms like offices, this introduces inaccuracy for other rooms such as hallways. An approach to generate finely graded instances based on semantic analysis of the building structure is presented in [Sch11]. As such an approach requires careful reasoning we propose a method to automatically generate networks with reasonable level of detail by subdividing the building geometry into rectangular areas of variable size. Our approach works completely automatically and can be based on the discrete structure of a cellular automaton. Hence, implementations can be easily added to existing simulation software based on cellular automata.

Outline of the Chapter. In Section 3.1 we describe our evacuation model which is implemented in the software tool ZET. From a given scenario we derive two consistent models for simulation and optimization that can be generated automatically. We then use the generated network models to compute exit assignments in Section 3.2. To verify the effectiveness of the different approaches we compare them on simple test instances representing situations commonly occurring in evacu-

ations. In Section 3.3 we describe a test evacuation of a 20-storey building that we have observed and compare the measured data with results from earliest arrival flow in the network model and the simulation results.

3.1 Modelling of Evacuation Scenarios

In order to show practical relevance of network flow optimization, our approaches are implemented within the framework provided by the ZET software[2]. In the following we will briefly describe the main features of the evacuation software that we use for our experiments. After a short discussion of the underlying evacuation model in the software, we give an introduction into evacuation simulation and into the process of creating network flow models from an evacuation scenario. The approaches in the remainder of the chapter base on the interfaces provided by ZET and the network flow algorithms published in the OPEN NETWORK FLOW LIBRARY[3] and are available as plugins.

3.1.1 The ZET evacuation tool

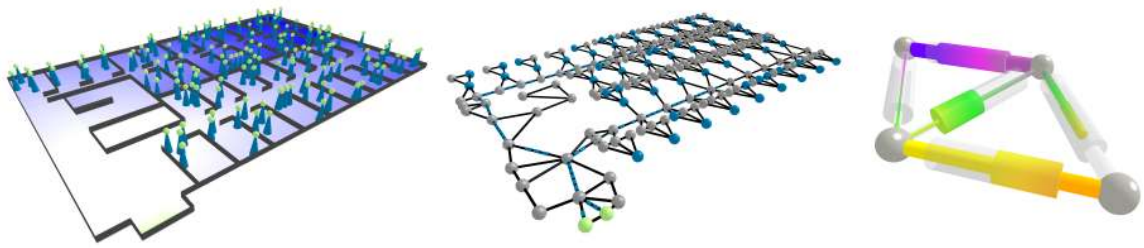
ZET consists of three main components. A use case typically begins with using the *editor* to model an evacuation scenario by providing building structure and specifying positions of evacuees. The final result can be presented using the *visualization* component. Version 2 supports visualization of cellular automata, flows over time in classical networks and Nash flows over time in the queuing model as described by Koch and Skutella [KS11]. An example for each of the visualization types is given in Figure 3.1. Figures 3.1a and 3.1b depict the ground floor of a highly populated office building as cellular automaton and network, respectively. The cellular automaton provides a floor field visualizing the distance from the exit in different shades of blue and evacuees as cone shaped objects. The network visualization distinguishes sources, sinks and intermediate nodes by different colours. Sources are depicted in blue and sinks in green. Flow units using an arc are represented as blue cylinders around the arcs. As another example Figure 3.18 at the end of this chapter shows a visualization of the building used for the practice study. In the Nash flow visualization depicted in Figure 3.1c edges are extended by transparent queues that can be filled by flow units that queue up behind a bottleneck. Flow units that arrive at the same time at the sink have the same colour. An elaborate introduction into Nash flows is due to Koch [Koc12].

Simulation and Optimization. However, the last component is most important for our experiments: The algorithmic part contains algorithms for simulations and optimization as well as algorithms converting the evacuation scenario into instances for the other algorithms. ZET provides a powerful interface that allows to extend the basic algorithms already included by additional plugins. All algorithms described in the remainder of this chapter are implemented as plugins and are available from the project page.

Algorithms

Efficient network data structures and algorithms are the core ingredient for a connection of network flows with simulation to optimize evacuations. The OPEN NETWORK FLOW LIBRARY implements

[2] `http://zet-evakuierung.de/en`
[3] `http://onfl.zetool.org`



(a) Cellular automaton model. (b) Network with flow over time. (c) Nash flow over time.

Figure 3.1: Examples of different evacuation models using the visualization of ZET.

many network flow algorithm and aims at two goals: Implemented algorithms should be fast such that large instances of flow over time problems can be solved. Furthermore, the algorithms should provide easy to use interfaces to implement further algorithms easily.

Data Structures. The network flow library provides all necessary data structures to implement both static and dynamic network flow algorithms. The networks can be equipped with additional parameters that are modelled as functions. This way it is easy to switch from costs in a network to lengths in a shortest path algorithm. The classic data structures are complemented with dynamic variants such as time-expanded networks and flows over time, both as path and edge flow. An interesting variant of time-expanded networks is the increasing time-expanded network which is introduced in Chapter 4.

Basic Algorithms. The library provides a basis of classical algorithms together with many network flow over time algorithms used in the context of evacuation. We briefly list the most important ones. Search algorithms such as breadth first search as well as different shortest paths algorithms have been implemented as basic algorithms. Dijkstra's algorithm supports non-negative rational arc lengths [Dij59], for instances with general arc lengths without negative cycles the algorithm by Moore, Bellman and Ford is implemented [Bel58; For56; Moo59]. Also, not so common algorithms such as the all pair shortest path algorithm for undirected graphs by Shoshan and Zwick [SZ99] and an implementation of Yen's algorithm to find k shortest paths (for $k \in \mathbb{N}$) [MP03] are contained.

Static Network Flows. We implemented several algorithms solving the MAXIMUM FLOW OVER TIME PROBLEM and the MINIMUM COST FLOW PROBLEM. For the former problem we implemented two frameworks, an augmenting path framework for classical algorithms and a push-relabel framework. Within the push-relabel framework we implemented the algorithm provided by Goldberg with the highest label strategy and the global labelling and gap relabelling heuristics proposed by Cherkassky and Goldberg [CG97]. The implementation is fast enough to run fast on large instance such as time-expanded networks. These implementations are complemented by a column generation approach linked with Gurobi and an implementation of Fujishige's maximum flow algorithm for fractional flows [Fuj03].

To solve the minimization variant we implemented the MINIMUM MEAN CYCLE CANCELLING ALGORITHM [Kle67], which is a simple strongly polynomial algorithm [GT89] and the SUCCESSIVE SHORTEST PATH ALGORITHM 1.1.

Flow over Time Algorithms. The main contribution of the implemented algorithms lies in the variety of flow over time algorithms that are available. The Maximum Flow over Time Problem can be solved in polynomial time by using the Minimum Mean Cycle Cancelling Algorithm on the extended network (see Definition 1.5) or by one of the maximum flow algorithms in the time-expanded network. The Quickest transshipment Problem can be solved using a binary search framework to calculate the optimal time horizon and then computing a static transshipment in a time-expanded network. In the same way, the earliest arrival flow algorithm can be solved. However, these approaches are not feasible in practice because they have very large running time.

We have implemented several algorithms solving the Earliest Arrival Flow Problem even on large instances. The natural approach simply uses the Successive Shortest Paths Algorithm Problem in a time-expanded network. The approach by Tjandra [Tja03] that also augments flow on shortest paths, but stores flows implicitly, is also implemented. Both of these approaches allow also instances where arcs are equipped with release dates and deadlines, however the software does not take advantage of such features yet. We also implemented value-approximate earliest arrival flows, for a discussion of those we refer to Chapter 4. The implementations use transit time transformation based on reduced costs as described in Chapter 5 to minimize the necessary time horizon. The implementations are fast enough to solve the instances in the remainder of this chapter.

3.1.2 Evacuation Model

The evacuation model consists of two parts: The first contains the building geometry with additional semantic information. The second part of the model consists of the specification of evacuees in the scenario and their properties.

Building Structure. The building geometry consists of a set of rooms and areas with specific semantics within the rooms. A **room** r is a closed polygon specified by n line segments. Rooms are simple, i. e., they do not self intersect and two rooms r_1 and r_2 do only intersect at the boundary. Thus, two rooms may share a single point or some line segments. The border (as sequence of line segments) is considered to not be passable, however any sequence of line segments shared by two rooms can be defined to be passable and is then called a **door**. An evacuation scenario may consist of different floors, each of which contains several rooms. Passages between two floors can be realized by specifying doors between rooms on different floors.

Any room may contain several areas, each of which itself is again a simple polygon. Areas impose additional semantics to the part of the room they cover and have therefore additional parameters. The most important areas are as follows.

- **Assignment area**. An assignment area specifies a region where evacuees reside at the beginning of the evacuation process. As a parameter, assignment areas store the number of evacuees they contain, such that these can be distributed randomly within the area.

- **Evacuation area**. An evacuation area denotes a place that is the target for all evacuees in the evacuation scenario. It will be typically modelled as a sink in networks. In a simulation model evacuees will be taken out of the simulation process when they reach such an area.

- **Stair area**. In the model we consider rooms to be flat. Stair areas can be used to model height differences within the building. Such areas contain two disjoint sequences of line segments on

their border that we denote as upper- and lower part and the stair is considered to go upwards from the lower part in direction of the upper part. The stair also has two speed factors as attribute to reflect changes in the movement speed along the stairs for both directions. Notice that this only makes sense if the lower and upper bounds are of equal orientation and are on opposite endings of the area. However, intermediate angles can be approximated in the conversion process by using the scalar product of the angle. In accordance to Weidmann [Wei93] the walking speed on indoor stairs downwards is scaled by a factor of 0.36 while the speed on stairs upwards is scaled by a factor of 0.45.

- **Inaccessible area**. Such an area simply models inaccessible regions that may occur as pillars or balconies. They do not need to have a non-zero surface area. If it is zero, they model a simple barrier.

Evacuees. The actual evacuation scenario consists in positioning evacuees in the building together with deciding some of their properties. These properties are for example their age, speed and knowledge of the given building. The evacuees can be of different types such that it is possible to model heterogeneous groups of inhabitants in a building. The values of the parameters are distributed at random according to given distributions and the evacuees are positioned randomly in the assignment areas. It is possible to have overlapping assignment areas such that different densities can be modelled.

3.1.3 Evacuation Simulation

Evacuation scenarios in buildings are typically small enough to apply microscopic models, i. e., models that allow to simulate each evacuee individually. Such models then can be divided further into discrete and continuous models. In the latter type, the evacuees can move freely within the building geometry and the trajectories during the evacuation process may be arbitrary (with respect to possible human movement). On the contrary, discrete models allow evacuees only to obtain positions in certain discrete positions and accordingly limit possible movement directions. These scenarios are often implemented on a *lattice*. Continuous models are often computationally challenging and are therefore not as widely used as discrete models. One of the most prominent examples of continuous models is the so-called *social forces model* proposed by Helbing et al. [HFM+02]. In the remainder of this chapter we only use discrete lattice-based models.

Cellular Automata

The concept of cellular automata goes back to early work by von Neumann [Neu66]. While working on the development of self-reproducing robots he described the concept of a lattice-based cellular automaton.[4] The concept turned out to be very powerful and has many applications, one of which is traffic and pedestrian modelling. Nagel and Schreckenberg modified the model for the case of single-lane traffic simulation [NS92]. Their model was able to show many of the observable properties of single-lane traffic. Consequently, the model has been extended further to more complex cases including several lanes and pedestrian traffic. A limited two dimensional cellular automaton model

[4] Von Neumann finally found a model of a machine reproducing itself on a grid after long research on this topic. The model was then simplified by Conway, who invented the very well known *Game of Life* cellular automaton [Gar70]. Despite its simplicity, the model turns out to be powerful enough to simulate a universal Turing machine.

Figure 3.2: A model of a lecture hall within the editor of ZET. The building geometry is visualized by black lines. Assignment areas are denoted in blue, green represents evacuation areas and stair areas are depicted in yellow.

for pedestrian motion is due to Blue and Adler [BA99]. The possible movement was further refined by the introduction of floor fields by Burstedde et al. [BKS+01]. They also introduced dynamic floor fields that can be used to generate behaviour such as lane formation and contraflow at bottlenecks, which before was only possible to generate in more complex models. A detailed overview on cellular automaton based simulation of pedestrian flows is due to Klüpfel [Klü03].

Evacuation Cellular Automaton. Due to its simplicity and at the same time powerfulness, the cellular automaton model is a natural candidate to be used in our combination of simulation and optimization. The concept is easy to implement, fast and well understood. An **evacuation cellular automaton** consists of a two dimensional grid C of **cells**, a **neighbourhood function** $N : C \to \mathcal{P}(C)$, a set of **exits** $\mathcal{E}$, and **potentials** $\text{pot}_e : C \to \mathbb{R}_{\geq 0}$ for each exit. A subset of cells $C' \subseteq C$ is **connected**, if for any two cells $c_1, c_n \in C'$ there exists a sequence $c_1, c_2, \ldots, c_n$ with $c_i \in N(c_{i-1})$ for $i = 2, 3, \ldots, n$. An exit $e \in \mathcal{E}$ is a connected subset $e \subseteq C$ of the cells such that two exits are disjoint, i. e., $e_1 \cap e_2 = \varnothing$ for two exits e_1 and e_2. For any exit e the according potential defines the distance of any cell to the exit satisfying $\text{pot}_e(c) = 0$ if and only if $c \in e$. We shortly write $CA = (C, N, \mathcal{E}, \text{pot}_e)$ for a cellular automaton.

For a given evacuation scenario we are given a set $\mathcal{I} = \{1, 2, \ldots, n\}$ of evacuees. The **state** s of a cellular automaton then is defined by an injective mapping $M_s : I \to C$ of the individuals to positions and some state dependent properties $p(i)$ of a finite state space. For example, the state covers the information if the individual is safe, its current speed or the destination exit.

The simulation is performed by the evacuation **cellular automaton algorithm** that starts with the initial state s_0 and changes the state in each iteration until the evacuation is complete. An evacuation simulation is considered complete if all evacuees within the simulation are safe (or otherwise taken

out of the simulation). For any evacuee i on cell c, the algorithm applies actions defined by a given **rule set** $\mathcal{R}$ taking into account the evacuees state $p_s(i)$ and computes a new state s'. A valid rule set has to contain a **movement rule** that changes the position $M_s(i)$ of evacuee i based on some potential pot_e. The general simulation framework is given in Algorithm 3.1.

Algorithm 3.1: General Evacuation Cellular Automaton Algorithm

---

Input: Cellular automaton $CA = (C, N, \mathcal{E}, \text{pot}_e)$, evacuees $\mathcal{I}$, and a rule set $\mathcal{R}$, and an initial state s_0.

Output: A sequence of states $s_0, s_1, \ldots, s_k$.

1. Set $t \coloneqq 0$.
2. For every evacuee $i \in \mathcal{I}$:

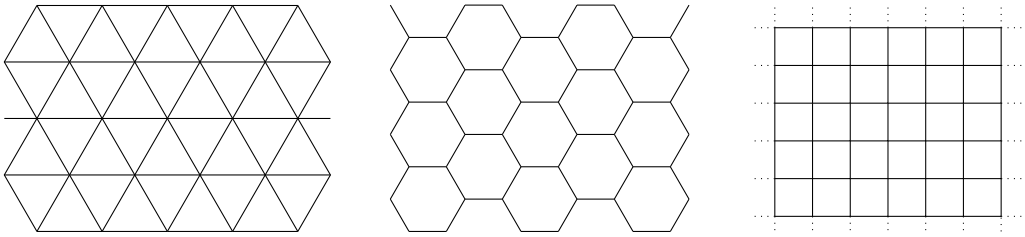
   For every rule $r \in \mathcal{R}$:

   - Perform actions specified by r on cell $M_{s_i}(i)$ taking into account neighbour cells $N\big(M_{s_i}(i)\big)$ and change state of the cellular automaton to s'.
3. Update the state $s_{t+1} \coloneqq s'$ to the current state.
4. If there are still evacuees active in the cellular automaton, set $t \coloneqq t+1$ and continue with step 2.

Notice that the algorithm may not terminate, if the rule set is not defined carefully. Also, the rules are executed one after another for the evacuees within the simulation. This may lead to problems, if the order in which the evacuees are iterated over in step 2 is badly chosen. The ideal concept would be a parallel execution, i. e., the rule set is applied for all evacuees in parallel. This best resembles reality, in which people also do not move one after another. However, such a procedure can create *collisions* if two evacuees shall move to the same cell. If such a parallel update is desired, an additional resolving step can be inserted after step 2. For a detailed introduction to parallel updates see for example [Klü03]. If the rules are applied to evacuees sequentially, the order plays an important role. The two worst case situation is a line of evacuees standing on consecutive cells. If the rules are applied for the evacuees from the first to the last evacuee (with respect to their desired walking direction), they can all walk at the same time in one step, thus resembling marching soldiers. The opposite happens if the order is reversed. Then in the first step only the first evacuee can move, while the others have to wait. It is therefore good practice to shuffle the order of the evacuees randomly before each iteration.

Cell Shapes and Neighbourhoods. The lattice of the cellular automaton should fill the two dimensional Euclidean space, thus allowing triangles, squares[5], and hexagons as possible shapes. The triangle is not very suitable, because it allows only few direct neighbours and its shape does not fit well to human evacuees. The square and hexagon have different advantages and disadvantages

[5] In fact, every parallelogram can be used. However, simulation is more accurate if the possible walking directions are distributed equally and the distance differences to neighbour cells are minimized. Therefore only squares and rectangular cells serve as quadrilateral shapes.



(a) Triangle pattern. (b) Hexagonal tiles. (c) Squares.

Figure 3.3: Possible cell shapes of a cellular automaton.

and are both used in practice. Hexagonal cells have equal distance to the six neighbours and fit the shape of human beings reasonably well. However, on buildings with two rectangular main axes it is not possible for evacuees in the simulation to walk straight in both of the directions. In the case of squares, allowing movement along the four direction of the horizontal and vertical axis is a very strong limitation. Allowing diagonal movement, too, however elongates the distances to four of the eight neighbour cells by a factor of $\sqrt{2}$. This neighbourhood is called **Moore neighbourhood**. Figure 3.4 depicts some common neighbourhoods. The cellular automaton implemented within ZET uses square cells with the Moore neighbourhood.

Movement. We will briefly describe how movement of evacuees is realized. In any step an evacuee can move from a cell c to one of its neighbour cells $N(c)$. The cell change is executed immediately. To realize different walking speeds and to compensate the longer walking distance in diagonal directions of the Moore neighbourhood, the evacuees make a compulsory break after each step. This break is called waiting period. To derive the directions of movements we employ the potentials. We assume that each evacuee follows a potential pot_e for some exit e. The potential assigns a real value describing the approximate distance to the exit fo each cell. The direction (i. e., which of the neighbour cells is used) is selected based on the potential difference. A neighboured cell $c' \in N(c)$ is selected with probability

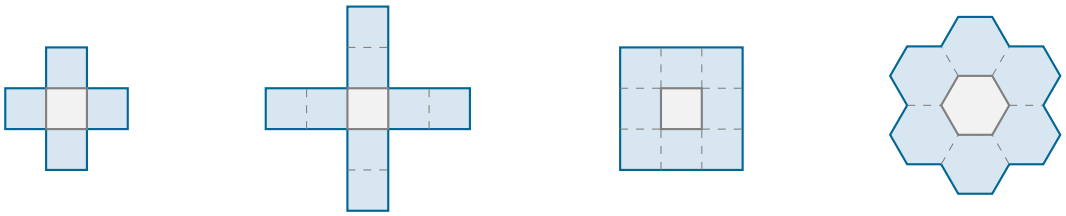
$$p(c') := \frac{e^{\text{pot}_e(c') - \text{pot}_e(c)}}{C},$$

where

$$C := \sum_{c' \in N(c)} e^{\text{pot}(c') - \text{pot}(c)}$$

is a normalization constant. Thus, an individual walks in direction of decreasing potential values but can deviate from the optimal path with some probability. Notice that in the above discussion we assumed that all neighbour cells are free. If a cell already contains an evacuee, the cell it stands upon is simply excluded from the neighbourhood.

Potential Values. The potential values for a potential according to exit e using the Moore neighbourhood are computed as follows. The potential is initialized with $\text{pot}_e(c) = 0$ for exit cells $c \in \mathcal{E}$ and with $\text{pot}_e(c) = \infty$ for all other cells $c \in C \setminus e$. The potential values of the other cells is then computed in a breadth first search manner. First all cells $e \in e$ are inserted into a queue. Then,



(a) Von Neumann neighbourhood.

(b) Extended von Neumann neighbourhood.

(c) Moore neighbourhood.

(d) Neighbourhood for hexagons.

Figure 3.4: Common neighbourhoods in design of cellular automata. If we denote the grey cell with c, the neighbourhood $N(c)$ are the blue cells.

until the queue is empty, a cell c is extracted and for all cells $c' \in N(c)$ the potential is updated to $\min\{\text{pot}_e(c'), \text{pot}_e(c) + \text{dist}(c, c')\}$ where dist equals to 1 if the cells are horizontally or vertically aligned, and $\sqrt{2}$ else. For hexagonal cells all neighbour cells have a distance of one.

Swapping. A special case occurs, if two persons want to switch positions. The simulation algorithm implemented in ZET can detect such situations and allows the two evacuees to swap positions. As the space for the swap is very limited, the swap will lead to a longer waiting period before the next move can happen. Swapping is necessary for some of the experiments we conduct in the next section.

3.1.4 Deriving Network Flow Models from an Evacuation Scenario

The first step in any sort of simulation or optimization is the transfer of the building geometry into a model that is digestible by the respective algorithm. The process should work fully automatically without manual intervention and ideally creates "similar" models for different methods such that the results are comparable. We therefore decide to use a raster of 40cm × 40cm square cells for the cellular automaton. This value allows for a maximum density of 6.25 persons per square meter and seems to be quite reasonable in most cases according to [Wei93]. The building is rasterized so that it can be represented by the cells and therefore the model is discrete in space. Based on this discretization we also define a network flow model. For the conversion we assume that all vertices of polygons are positioned on a grid and have coordinates that are multiples of the cell size. Furthermore we assume that all line segments are either vertical or horizontal, i. e., either the x- or y-coordinates of their respective endpoints are equal. Such geometries allow native conversion into cellular automatons. For the conversion into a network flow model we describe a variant that directly converts such a rasterized geometry into a network flow model.

Rasterization. If the building geometry does not suffice the above assumptions, a rasterization preprocessing step is necessary. First, the geometry has to be rotated such that it best fits into the requirement of having horizontal and vertical lines only. This can be done efficiently using principal component analysis. For details see specific literature, for example [Jol02]. In a second step all polygon coordinates have to be moved to the closest multiple of the underlying grid size. In order to optimize results beforehand all points may be translated. Assume C is the grid size and we have a total amount of n points (x_i, y_i) with positive coordinates for $i \in \{1, 2, \ldots, n\}$. Then we want to

minimize the sum of the distances $\sum_{i=1}^{n} C \cdot (\lfloor \frac{x_i}{C} \rfloor + \lfloor \frac{y_i}{C} \rfloor)$. Observe that by shifting all points in one direction we improve the sum for k points and worsens the sum for the remaining $n - k$ points if the position was not optimal. After the points lie on the given raster we have to rasterize also lines that are not vertically or horizontally aligned. This can be done for example by replacing them with a sequence of steps, e. g., by using Bresenham's algorithm [Bre65].

Creating a Cellular Automaton Model

The rasterization naturally defines a cellular automaton based on square cells. Let an evacuation scenario with a set of k rooms $\{r_1, r_2, \dots, r_k\}$ be given. The rooms are rasterized with a grid size C using the technique above, i. e., all line segments of the polygons are axis aligned and the coordinates are multiples of C. Let c_{ij} be a square cell defined by

$$c_{ij} \coloneqq \{(x, y) \in \mathbb{R}^2 \mid C \cdot i \leq x \leq C \cdot (i + 1), C \cdot j \leq y \leq C \cdot (j + 1)\}.$$

We then define the set of cells as

$$\mathcal{C} \coloneqq \left\{ c_{ij} \;\middle|\; \exists 1 \leq \ell \leq k : c_{ij} \text{ lies completely within } r_\ell \text{ and is not inacessible} \right\}.$$

The neighbour cells

$$N(c_{ij}) \coloneqq \left\{ c_{i-1\,j-1}, c_{ij-1}, c_{i+1\,j-1}, c_{i-1\,j}, c_{ij}, c_{i+1\,j}, c_{i-1\,j+1}, c_{ij+1}, c_{i+1\,j+1} \right\} \cap \mathcal{C}$$

of cell c_{ij} is the Moore neighbourhood consisting of the 8 surrounding cells (if they exist) and the cell itself. We additionally include the original cell to allow an evacuee not to move at all if all neighbour cells are already occupied.

For each of the exit areas defined in the evacuation scenario we add a potential to the cellular automaton. Evacuees are distributed randomly in the regions defined by the assignment areas. In order to keep the simulation results comparable with the network flow output, each person is assigned the same speed. Reaction times, movement speeds, and exit selection are controlled by according rules. The rules also make sure that the walking speed is adjusted on stairs by the factors specified in the model.

Creating a Network Flow Model

Modelling networks for evacuation scenarios almost always comprises some kind of balancing between level of detail and accuracy of the results. We want to achieve the following goals in our automatic conversion process. First of all, any resulting optimal earliest arrival flow should not take longer than an actual evacuation. Ideally, we want to get most accurate results out of a network flow model. If some inaccuracy is introduced, at least we do not want to over estimate the evacuation time because an earliest arrival flow is the best possible outcome and we expect it to be a lower bound on the actual evacuation time. Second, the computation should not take too long, as the process should be included in the workflow of existing evacuation tools.

The first network flow models for evacuation modelling are due to Chalmet, Francis and Saunders [CFS82]. They present a rather rough model for an eleven-floor building and show how dynamic network flows can be used to detect bottlenecks within an evacuation of that building. They use three to four nodes per floor in addition to a small number of temporary nodes. Capacities of

nodes and arcs are computed manually. Based on this model we formalize the first very simple network flow model for building evacuation. Examples of the rasterization approaches we consider are depicted in Figure 3.5. Additional examples of the rectangle given in Figure 3.10 and Figure 3.20.

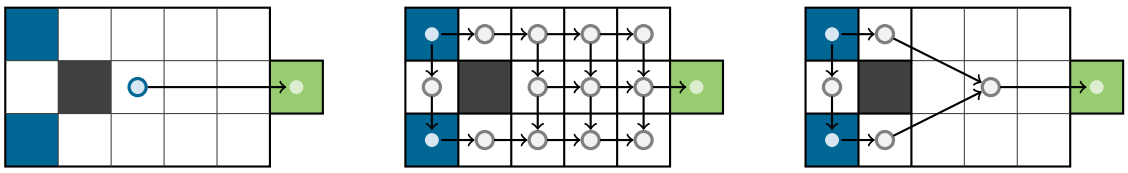
Simple Modelling. Let a building geometry with n rooms $R = \{r_1, r_2, \dots, r_n\}$ and k exits be given. In a simple network model, the set of vertices $V \coloneqq R \cup \{t_1, \dots, t_k\}$ consists of a node for each room and a sink t_i for each of the k exits. Any room that contains evacuees will become a source. We create an edge (r_i, r_j) for each door that connects rooms r_i and r_j and also an edge (r_i, t_j) if exit j is present in room r_i. The capacities of the arcs are estimated by the capacities of the door width and the node capacities by the area of the room they represent. To have a more realistic model we may add additional nodes and arcs for stairways and scale their capacity, as proposed by Chalmet, Francis and Saunders [CFS82].

Raster-based Network Models. For our purposes we use a more detailed approach that is based on the given rasterized structure for the cell size 40cm × 40cm. Generally, each cell can be represented by a node which leads to very large models. To generate more tractably sized instances we combine several (rectangular) clusters of cells together to one node. Assume we are given the cells of the cellular automaton $\mathcal{C}$ as defined above. We create one vertex for each of the cells and use $V \coloneqq \{v_{ij} \mid c_{ij} \in C\}$ as the set of vertices. We connect nodes that that lie next to each other vertically or horizontally and therefore define the set of edges to be

$$E \coloneqq \left\{(v_{ij}, v_{k\ell}) \;\middle|\; v_{ij}, v_{kl} \in V, |i-k| = 1 \text{ and } j = \ell, \text{ or } i = k \text{ and } |j-\ell| = 1\right\}.$$

Thus, we get a very dense network in which any vertex has at most 4 adjacent vertices that directly corresponds to the cellular automaton. It allows detailed flow computations with the drawback of huge instance sizes which makes the model impracticable in practice. We do not add diagonal arcs because it reduces the accuracy. Each cell should model one evacuee standing on it, however more arcs on the same space increases density modelled by a flow over time. Furthermore the ability to model arc lengths of 1 and ≈ 1.4 is only possible with smaller step sizes leading to higher throughput.

Rectangle Raster. We decrease density by aggregating some cells to rectangular areas and use one node for each area. We assume that the nodes are positioned in the centre of their associated rectangle and nodes are connected if the respective rectangles meet at their boundary. Because evacuees will not take detours via the centre of a rectangle in reality we want to avoid that the rectangles deviate too far from being square. More formally, a rectangle with width w and height h should satisfy $|w - h| \leq 1$.



(a) Simple modelling. (b) Raster-based network model. (c) Rectangle raster.

Figure 3.5: Three possibilities to model a room with a single exit, two evacuees and an obstacle as a network flow model.

Optimal Rectangle Positioning. The question of how to place the rectangles within the area gives rise to an interesting research question of its own. The problem of finding a minimum number of rectangles covering an orthogonal polygon is already $\mathcal{NP}$-hard, even if the rectangles are allowed to overlap. However, these problems tend to behave well in the sense that it is easy to compute solutions with small gaps to the optimum using primal dual approximation algorithms [HL07]. We approach the problem using a simple heuristic. We start in the upper left corner of the orthogonal polygon and try to extend the first free cell by a rectangle that is as big as possible (and still obeys the above constraint regarding its size). We then iterate through all fields of the polygon and skip fields that are already covered by a rectangle. More formally, the following algorithm 3.2 describes the process. Notice that the actual implementation within the software tool ZET is more sophisticated and takes areas with different semantics such as assignment areas and stairs into account. The former contain evacuees in the beginning and should cover a well defined region to place sources while the latter changes walking speed which leads to longer transit times on arcs.

Algorithm 3.2: Rectangular network

Input: Raster nodes $V = \{v_{11}, \dots, v_{mn}\}$ belonging to one room r of the building geometry.

Output: Partitioning of the geometry into rectangular areas belonging to nodes.

Initialize $M_{ij} \coloneqq 0$ for all $i = 1, \dots, m$, $j = 1, \dots, n$.

For $i \in \{1, 2, \dots, m\}$:

For $j \in \{1, 2, \dots, n\}$:

- If $M_{ij} \neq 0$ continue with the next iteration for $j + 1$.
- Search an $n \in \mathbb{N}_0$ maximally such that $M_{i+k,j+\ell} = 0$ for $k, \ell \in \{0, 1, \dots, n\}$.
- If the $n \times n$ square can be extended to the right such that $M_{i+k,j+n+1} = 0$ for $k \in \{0, 1, \dots, n+1\}$ set $w \coloneqq n + 1$, other wise set $w \coloneqq n$.
- If the $n \times n$ square can be extended below such that $M_{i+n+1,j+k} = 0$ for $k \in \{0, 1, \dots, n+1\}$ set $h \coloneqq n + 1$, other wise set $h \coloneqq n$.
- Create node v and assign the rectangle from $\big(C \cdot (i-1), C \cdot (j-1)\big)$ to $\big(C \cdot (i+h-1), C \cdot (j+w-1)\big)$ as area(v).
- Set $M_{i+k,j+\ell} \coloneqq 1$ for $k \in \{0, 1, \dots, h\}, \ell \in \{0, 1, \dots, w\}$ and $|w - h| \leq 1$.

Transit Times. Nodes are placed in the centre of their respective rectangles and connected via an arc if the rectangles meet at the boundary. The transit times are set to the rounded euclidean distances. The walking speed changes on stairs (depending of the direction). We scale the transit times by the above mentioned factors of 0.36 and 0.45 for downward and upward arcs, respectively.

Reaction Times. The evacuation scenario allows to define a reaction time for persons. In our network flow model this time is modelled by the introduction of an additional single source node for each flow unit. This node is connected with the original source with a single arc having a transit time such that the flow unit arrives at the starting node not earlier as the specified reaction time. The travel time of each source-sink path is increased by the time equivalent to the reaction time of the evacuee corresponding to the flow unit in the source. Due to increased path length it is possiblethat the time horizon increases drastically. However, we can neglect such effects on the path length because due to the Reduced Costs Transit Time Conversion Algorithm 5.1 described in Section 5.3, we can assume without loss of generality that the shortest flow carrying path has a travel time of 0 and that the time horizon is decreased accordingly. Thus the influence on the running time due to modelling reaction times in a network flow model can be kept small if reaction times are chosen reasonably.

3.2 Exit Assignments

The easiest possibility to influence the course of an evacuation is to assign exits to the evacuees and let them choose their paths freely. The idea behind the approach is, that evacuees typically will not divide in optimal way but rather behave ineffectively. We expect most evacuees in real life scenarios decide to take the exit they were coming from or just follow some other evacuees. It is also likely that they choose the shortest exit. In this section we define exit assignments for evacuees and analyse changes in the egress time with simulations in simple scenarios.

Definition 3.1 (Exit assignment). Let $\mathcal{E}$ be the set of exits of an evacuation model and $\mathcal{I}$ be the set of evacuees. An **exit assignment** is a function $a : \mathcal{I} \to \mathcal{E}$ that assigns the exit each evacuee should take in an evacuation scenario. ◁

Shortest Paths Assignments. A natural variant to assign evacuees to exits is to assign each of them to a nearest exit.We can define a nearest exit assignment in two possible ways: If a network flow model is given, shortest paths in the network can be used to assign sources to sinks; in a cellular automaton the potential values also define a nearest exit for each cell. Shortest paths assignments a_{SP} based on a rough network flow model where one node may cover a larger area can be imprecise because all evacuees in one node have to take the same exit. We give the algorithm to compute network shortest paths exit assignments as introductory example but use cellular automaton potentials in the final experiments. In Section 3.2.2 we briefly compare the two approaches.

Algorithm 3.3: Shortest Paths Exit Assignments

Input: Network Flow Model $\mathcal{N} = (G = (V, E), u, \tau \geq 0, S^+, S^-)$.

Output: Shortest Paths Exit Assignment a_{SP}.

1. For each source in $\mathcal{N}$:
   a) Compute a shortest path tree.
   b) Let $t \in S^-$ be the sink that minimizes $\text{dist}(s, t)$.
   c) For each evacuee i belonging to s, set $a_{SP}(i) := t$.
2. Return a_{SP}.

Minimum Cost Flow Assignments. The minimum cost flow assignment a_{MC} uses a more advanced method than simply using shortest paths but still works on small static networks. The idea is that we only want to assign evacuees to exits ignoring the actual network structure and therefore use a bipartite graph consisting only of the sources and sinks of the network model.

We define supplies for the sinks that distribute the evacuees among the exits proportionally to the estimated capacity. By capacity we think of the amount of flow that can reach an exit in each step. The relative capacity of the exits (compared to each other) is computed by maximum flow computations in the static network. We also have to define costs for the arcs in the bipartite graph. An arc connecting nodes $s \in S^+$ and $t \in S^-$ should represent the time necessary for a flow unit to travel from the given source to the given exit. We therefore use the shortest path distances as costs and an infinite capacity.

Algorithm 3.4: Min Cost Flow Exit Assignments

Input: Network Flow Model $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$.

Output: Exit Assignment a_{MC}.

1. Compute shortest paths distances $\text{dist}(s,t)$ between all source-sink pairs $(s,t) \in S^+ \times S^-$.
2. For each sink assigned to an exit t_e compute $\text{cap}(t_e) := |f|$ for a maximum flow f in the network $\mathcal{N} = (G, u, S^+, t_e)$.
3. Define a bipartite graph $G' := (V', E')$ with nodes $V' := S^+ \dot\cup S^-$ and arcs $E' := \{(s,t) \mid s \in S^+, t \in S^-\}$. Define costs $c' : E' \to \mathbb{N}_0$, capacities $u' : E' \to \mathbb{N}_0$, and balances $b' : V' \to \mathbb{Z}$ by setting

   $$c'(e) := \text{dist}(s,t),$$
   $$u'(e) := \infty,$$

   for all arcs $e = (s,t) \in E'$ and

   $$b'(v) := \begin{cases} b(v) & \text{if } v \in S^+, \\ \frac{\text{cap}(v)}{\sum_{w \in S^-} \text{cap}(w)} \left(\sum_{w \in S^+} b(w)\right), & \text{else.} \end{cases}$$

   for all nodes $v \in V'$.
4. Compute a minimum cost transshipment f on G'.
5. Extract the exit assignment from the computed transshipment.

First, we calculate a maximum flow for every sink in the network to estimate the capacities of the sinks. The network for the following minimum cost computation uses these estimated capacities for the sinks and has infinite capacity on all edges. More precisely, our procedure is as follows: Let $(G = (V, A), u, \tau, b)$ be a network, with S^+ and S^- denoting the sources and sinks, respectively, as defined by the balance function. The following Algorithm 3.4 describes the procedure to generate an exit assignment by minimum cost capacities.

Earliest Arrival Assignments. Following our discussion in Chapter 2, exit assignments based on earliest arrival flows seem to be a rather good choice. We denote the exit assignment that assigns each evacuee the exit that it would reach in an earliest arrival transshipment by a_{EAT}. In the multi-commodity case earliest arrival transshipments do not necessarily exist. However, due to the definition based on an actual earliest arrival transshipment a_{EAT} still allows for an earliest arrival flow. While an assignment based on earliest arrival flows should be optimal in the evacuation, this may not be the case in practice. The pattern is only achieved by a flow that follows the correct paths through the network. If evacuees do not follow these optimal paths, they may arrive with delay.

Algorithm 3.5: Earliest Arrival Exit Assignments

Input: Network Flow Model $\mathcal{N} = (G = (V, E), u, \tau \geq 0, S^+, S^-)$.

Output: Earliest Arrival Exit Assignment a_{EAT}.

1. Compute an integral earliest arrival transshipment in $\mathcal{N}$ with Algorithm 2.3.
2. Let x be a path decomposition of the earliest arrival transshipment
3. For each path P with positive flow $x(P) > 0$:
   a) Let $(s, \dots, t, t^*)$ be the nodes visited by P, i. e., it starts at a source s and uses a sink t as last but one node before reaching the super sink t^*.
   b) Assign exit t to $x(P)$ evacuees belonging to the area of s.

Best Response Assignments. Other concepts than network flow models can be used to define exit assignments, too. Ehtamo et al. [EHH+] propose a game theoretic approach that is based on response functions. They model the exit selection as an N-player game where the strategy of the evacuees consists of choosing an exit. The evacuees try to choose the strategy that gives the largest *payoff* value in response to the strategies chosen by all other evacuees. The strategy selection is iterated several times and converges to a *Nash equilibrium*. For more details on general game theoretic concepts and the existence and computation of Nash equilibria see for example the text books by Nisan et al. [NRT+07], Osborne [Osb04] or Fudenberg and Tirole [FT91].

Ehtamo et al. [EHH+] give a basic instruction on how to compute the payoff value. We concretize the calculation by defining the following formula. The payoff value of a selected exit ideally should be equal to the evacuation time, which consists of a weighted sum of the moving time and the queueing time at the exit. Any evacuee tries to minimize the evacuation time and thus minimizes its payoff. As we do not have the exact numbers, we estimate both, the moving time and the queuing time to compute the estimated evacuation time. For the moving time we take $\frac{\text{pot}_e(c(i))}{\text{speed}(i)}$, where $c(i)$ denotes the cell on which evacuee i is standing and speed(i) refers to its speed. Thus $\text{pot}_e(c(i))$ denotes the distance of an evacuee i from exit e. Notice that the estimate is rather rough as the speed may not be obtained in crowded areas. To estimate the queuing time we use $q_a(i)$ as the number of evacuees that are "in front" of a given evacuee i following the exit assignment a. Using cap(e) as the estimated capacity of an exit per time (as already used for minimum cost flow exit assignments), we define the estimated queuing time to be $\frac{q_a(i)}{\text{cap}(e)}$. For our experiments, we weight both parts of the evacuation time equally strong because it is not immediately clear which of the values should be preferred. When the method is applied for any evacuation scenario in practice, the values may be adjusted to the given

situation. It might very well be the case that the area around exits is safer and thus queuing at exits may be preferred over walking through an hazardous building. On the other hand big crowds at exits create the risk of panic if people wait too long. This discussion leads to the following Algorithm 3.6 for computing an exit assignment based on best responses.

Let K be the set of exits and $k \in K$ be one of the exits. Let the distance (in means of floor potential value) for evacuee i to k be denoted by $d_k(i)$, the number of evacuees heading to exit k by q_k, the capacity of the exit by c_k and v_{max} the speed of the evacuees. Then the payoff for the selected exit k for evacuee i is defined as

$$p(k,i) := \frac{1}{2}\left(\frac{q_k}{c_k} + \frac{d_k(i)}{v_{max}}\right)$$

Each evacuee chooses the exit $k^*(i)$ with $p(k^*,i) = \min\{p(k,i) \mid k \in K\}$ that minimizes the value for $p(k,i)$ and the process is iterated. It turns out that the process converges very fast, after very few iterations a Nash equilibrium is reached.

Algorithm 3.6: Best Response Exit Assignments

Input: Building Model.

Output: Exit Assignment a_{BR}.

1. Initiate exit assignment a_{BR} arbitrarily, e. g., randomly or by setting $a(i)_{BR} := e$ such that $\text{pot}_e(c(i))$ is minimal for all exits $e \in \mathcal{E}$.
2. For all evacuees $i \in \mathcal{I}$: Let $q_{a_{BR}}(i) := |\{i' \in \mathcal{I} \mid a_{BR}(i) = a_{BR}(i')\}|$. Set
$$a'_{BR}(i) := \arg\min_{e \in \mathcal{E}} \left\{\frac{1}{2}\left(\frac{q_{a_{BR}}(i)}{\text{cap}(e)} + \frac{\text{pot}_e(i)}{\text{speed}(i)}\right)\right\}.$$
3. If $a'_{BR}(i) = a_{BR}(i)$ for all evacuees $i \in \mathcal{I}$, return a'_{BR}, else set $a_{BR} := a'_{BR}$ and continue with step 2.

3.2.1 Experiment Scenarios

We compared the four propositions for exit assignments described above on a set of instances that can be problematic. Each instance focuses on a situation that can occur within a building. They are designed in such a way that they are as simple as possible and still cause the problem to avoid introduction of additional dependencies. All instances basically consist of a long stretched room with exits at either of the short sides. The common concept is that all evacuees are crowded in a single area somewhere in between the exits. Any optimal evacuation strategy just has to assign an optimum proportion of the evacuees to both sides. For each of the scenarios there is a corresponding figure with exits in green and an assignment area in blue.

Overestimation of Exit Capacities. The first instance depicted in Figure 3.6 highlights the influence of exit capacities on the evacuation. The crowd of evacuees is positioned exactly in the middle between the two exits, one of which is very wide while the other is rather small. An optimum evacuation plan would send more than half of the evacuees to the larger exit.

Figure 3.6: Test scenario with two exits (in green) of different capacity. The evacuees reside exactly in in between the exits in the blue area.

Figure 3.7: Two exits at very different distances. The exit on the left is much closer to the evacuees.

Exit Distances. Not only capacities, but also different distances of exits impose difficulties on the exit selection. The scenario in Figure 3.7 addresses this situation. Both exits are very wide and have the same capacity which is not limiting outflow, but all evacuees are very near to one of the exits. An algorithm creating good exit assignments should send all evacuees to the near exit. We expect that the scenario will be difficult for minimum cost exit assignments, because Algorithm 3.4 divides the evacuees proportionally.

Obstacles. Earliest arrival transshipments should be able to compute the optimal assignment in general. However, the transshipment takes into account the different routes through the network. This information is lost if evacuees are only given the information which exit to take. The instance depicted in Figure 3.8 is designed to test this phenomenon by using some pillars as obstacles. On one side the pillar forces all evacuees to take a long way, on the other side there is a narrow path through the obstacles. The exit behind the narrow path is closer to the evacuees, such that a good exit assignment should balance between sending more people on the short way and thereby risking a jam and sending more evacuees to the distant exit.

Shared Bottlenecks. As last scenario we consider shared bottlenecks as depicted in Figure 3.9. Again, the evacuees are located in the middle but now each side has two exits. On the left side each of the exits has a very limited capacity while on the right side there is a bottleneck that limits the inflow into both of the exits. The instance models a situation, where after an emergency staircase the available routes split into two actual exits. This scenario should be very well suited for exit assignments based on earliest arrival flows, as the optimal flow sends only few people in either of the exits sharing a common bottleneck. Approaches based only on the capacity of the exit have problems if they do not detect that the two exits on the right side share a common bottleneck.

Figure 3.8: Two exits with different kinds of obstacles in front of them. The pillars on the right side allow for a narrow path such that evacuees will not take the detour around the obstacles.

Figure 3.9: Four exits on two opposite sides. The exits on the left side each have a unique entry while the two exits on the right side share a single bottleneck.

3.2.2 Computational Study

All of the exit assignments are implemented using the network flow algorithms provided by ONFL and are made available as plugins to the software tool ZET. For the above scenarios we computed the optimal assignment, performed 200 simulation runs and measured the arrival curve for each run. For the simulation we want to exclude as many influences as possible and thus use simple rules for the simulation. Basically, any evacuee moves with the same speed and gets a fixed potential assigned for the exit the evacuee is supposed to go to. However, the fixed assignment to an exit creates blockage in front of the exits for the scenario comprising shared bottlenecks because the two exits are very near to each other. To avoid the blockages it is crucial that the movement rules allow the evacuees to swap as described above in Section 3.1.2.

Comparison of Simulation Runs. For all simulation runs of a given scenario we compute the average arrival curves and compare them with each other. The comparison based on the average arrival curve is justified, because the variance in the simulation runs is not very big in most cases. In all of our experiments the average arrival curve and the median arrival curve lie remarkably near together. Also, the significant area containing 90% of all arrival curves is very narrow. The scenario with the widest significant area is the shared bottlenecks scenario which has a higher variability due to the mentioned blockage. Figure 3.11 depicts maximum, minimum, average, median and significant areas of this worst-case scenario (cf. Figure 3.9) and the scenario with exits of different capacity (cf. Figure 3.6), which we use as example for a common scenario.

Assignment of Potentials. When we have computed an exit assignment we have to map the evacuees in the evacuation that belong to a source to actual positions in the rectangular area corresponding to the source. Placing evacuees arbitrarily in nodes covering a big area may induce the following

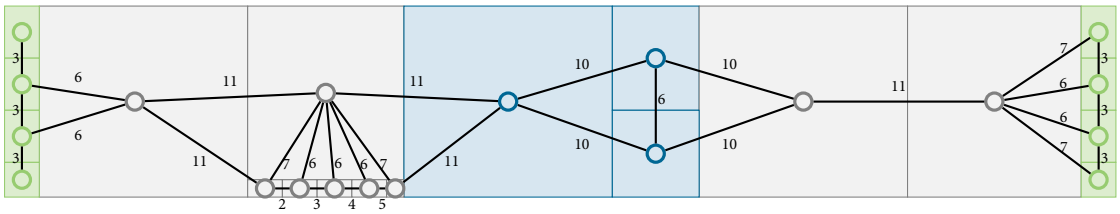


Figure 3.10: An example rasterization of a long corridor with two exits on either sides. The corridor is the network flow model version of the instance used for overestimation of exit capacities in Section 3.2 depicted in Figure 3.6.

problem (especially on our test instances that are long but narrow): If half of the evacuees assigned to a source should head right and the other half heads left, it is wise to place the individuals heading left in the left half of the node area and vice versa for the right half.

Shortest path exit assignments computed by a network induce another problem. In contrast to network flows that specify a target sink for each flow units, a shortest path defines a single sink for *all* evacuees starting in a sink. As an example consider the generated graph for the first scenario evaluating exits with different capacities. The generated graph for this scenario is depicted in Figure 3.10. It contains three sources, one of which covers a very large area containing the majority of evacuees. It is easy to see that a fraction of the evacuees in the area should go to the left exit, however, due to the shortest path computation all of them head to the right exit which is the shortest exit for all three sources. A finer graded distinction is achieved if we use potential values to define shortest paths as they define a shortest path for each *cell*. The difference between the two approaches is depicted in Figure 3.12. Due to this results we also use the potential values to simulate shortest paths exit assignments because we do not want to penalize the shortest path approach.

Results

We will briefly discuss the results of the simulation runs of the proposed approaches with respect to the given scenarios. It always improves evacuation times if an approach that not only takes the closest exits into account is used in the simple scenario that only incorporates different exit capacities (see Figure 3.15). This is not surprising as the shortest path distances ignore exit capacities. The minimum cost approach does not only consider the exits but also computes the width of the bottleneck when reaching an exit from a particular starting point. Thus it can not only handle examples as described in Figure 3.6 but also buildings that include structures such as staircases. The game theoretic best response dynamics approach also competes quite well and is mostly as good as the better flow approaches. The approach suffers slightly in the scenario with distant exits (Figure 3.7) and in the difficult scenario with shared bottlenecks (Figure 3.9).

Minimum Cost Flow Exit Assignments. The minimum cost approach does not care about the distances between evacuees and exits and thus may fail if there are wide exits very far away. The same problem arises if a lot of exits share the same bottleneck. This is because the algorithm does not take into account that evacuees heading to one of these exits may meet at the bottleneck and create a jam. Figure 3.16 reveals that minimum cost flow assignments perform worse than shortest paths for distant exits. These effects are especially serious in this example because the exits have

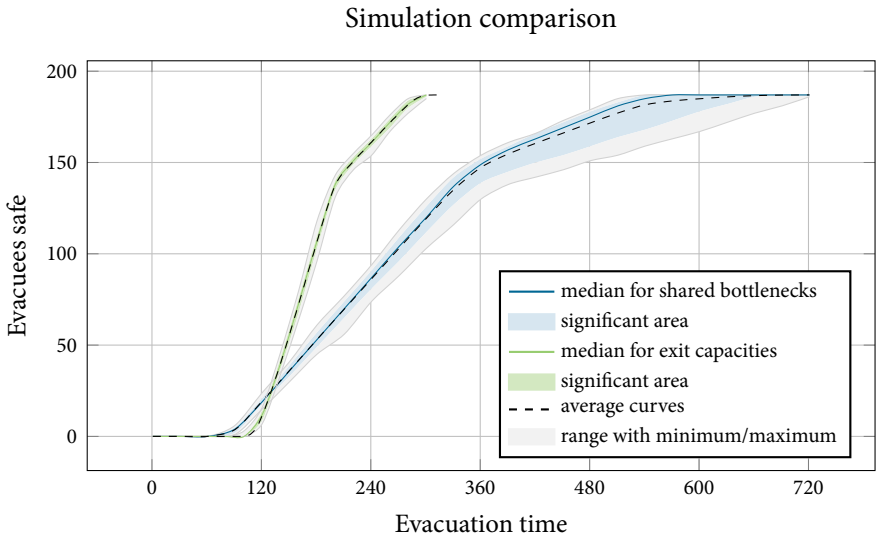


Figure 3.11: Arrival metrics for two scenarios for shortest path based exit assignments. As a typical scenario we use the simple scenario with exits of different capacity, the worst case scenario is given by shared bottlenecks.

practically no capacity limit. Minimum cost exit assignments should behave better if the exits are narrower.

Earliest Arrival Exit Assignments. The earliest arrival approach uses an optimal flow over time and thus does not suffer from these problems. By taking the temporal aspect into account it detects bottlenecks and exactly balances between the capacity and the distance of an exit. Figures 3.16 and 3.14 show the superior performance of the earliest arrival approach in the situations described as critical for the minimum cost approach. Notice also that the earliest arrival approach is the only approach dealing correctly with the shared bottleneck.

A possible error of assignments computed by an earliest arrival flow is more subtle. While the earliest arrival transshipment is optimal for the given network model, evacuees in the simulation need to follow exactly the routes of a path decomposition of the earliest arrival flow. However, the only information the agents in our simulation receive is the exit they are supposed to take.

Thus, we see a different behaviour: In the cellular automaton model (and probably also in reality) people try to get to their desired exit as fast as possible and choose the shortest path to get there. If the earliest arrival transshipment has decided that a lot of people should go to an exit reachable by many different ways of different length, all people will take the shortest one and jam somewhere at a bottleneck on the path. Subsequently, the evacuation takes significantly longer than the earliest arrival transshipment expected. Figure 3.13 shows that the minimum cost approach should be preferred for such situations and that the earliest arrival approach does not significantly perform better than shortest paths.

Discussion of Exit Assignments. We have seen that network flows greatly help to compute good exit assignments but the approaches are still in an early state of development and thus have some

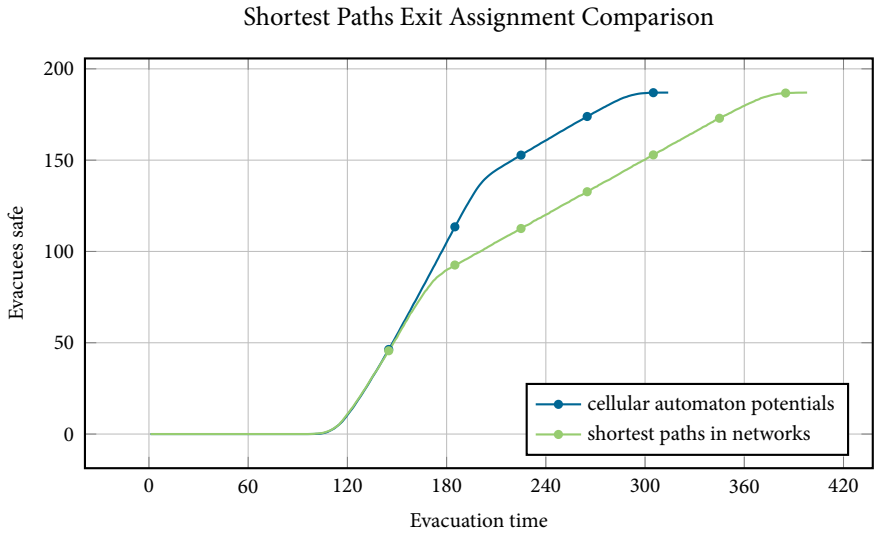


Figure 3.12: Exit assignments only based on network models of evacuation scenarios send evacuees to wrong exits.

disadvantages. We can achieve best results if all approaches are applied and the best exit assignment is chosen by testing all assignments in a simulation. Further research can be done in the direction to develop an algorithm preceding the best approach, e. g., by considering the network structure.

The gained exit assignments can also be used as initial distributions for iterative heuristics, like the best response dynamic approach, to decrease the number of necessary iterations. The performance of our approaches can also be improved by using network flow techniques to gain gain better approximations of the real capacity of exits. By using the values $\text{cap}(t_e)$ gathered by a maximum flow computation in the best response exit assignments the results improve significantly. It would also be the task of further research to improve the solutions based on network algorithms. We have seen that the algorithms have different weaknesses, and it is an interesting question whether the approaches can be combined to reduce their disadvantages. Furthermore, especially for earliest arrival transshipments, it would be nice to force the algorithm to avoid routes that evacuees are unlikely to follow.

Personal Escape Plans. Even more interesting is the question whether a more direct way to derive evacuation plans from network flows helps to develop a better evacuation plan compared to the case of exit assignments. To improve the egress time we may force evacuees to use specific routes. This may not seem feasible at first glance, however, due to the more common availability of devices such as smartphones, this approach may become practicable in the near future. It can easily be applied in city wide evacuation scenarios where evacuees should use their own cars. However, as people usually do not behave perfectly, additional ideas are needed to fully use the strength of network flow computations. Within upcoming intelligent response systems, i. e., systems that react to the actual situation *during* the evacuation process, personal routes and also exit assignments can be changed. The existing approaches so far only compute optimal flows. It might be a good idea to try to respect human behaviour and use Nash flows over time as introduced by Koch and Skutella [KNS11].

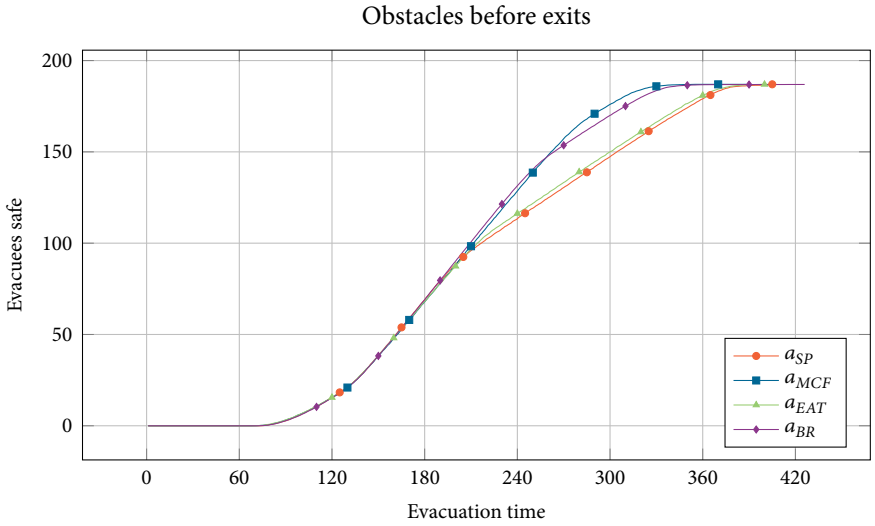


Figure 3.13: Comparison of the exit assignments on the instance depicted in Figure 3.8. The instance measures the effectiveness of the algorithm if there exits paths to exits of different lengths, but with limited capacity.

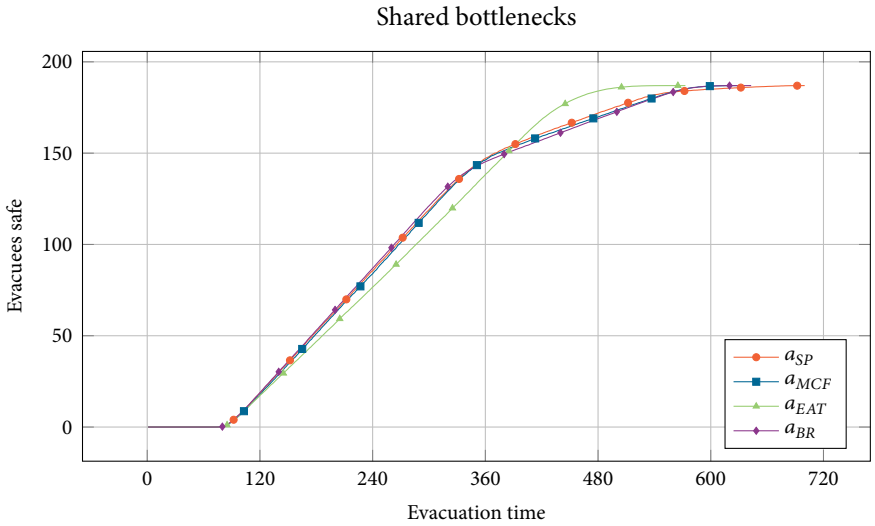


Figure 3.14: Comparison of the exit assignments on the instance depicted in Figure 3.6. The instance measures how the algorithms cope with exits that lie behind a shared bottleneck (effectively limiting the exit capacity to the capacity of the bottleneck). The earliest arrival approach is slightly worse in the beginning, however, it is the single best approach in the last part of the evacuation process.

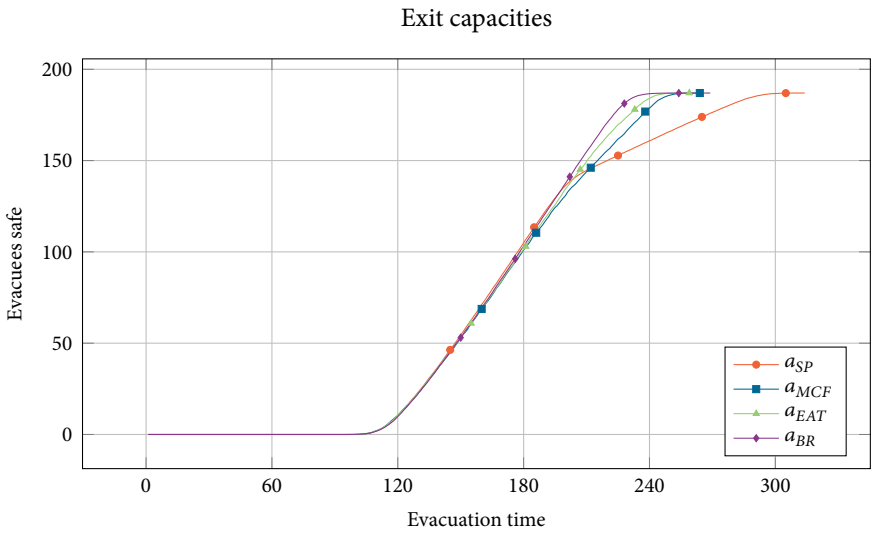


Figure 3.15: Comparison of the exit assignments on the instance depicted in Figure 3.6. The instance measures how the algorithms cope with exits of different capacity.

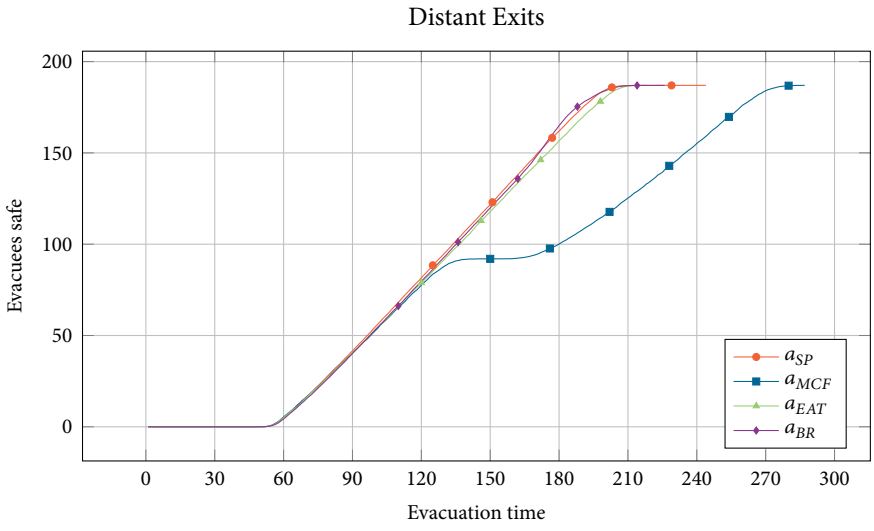


Figure 3.16: Comparison of the exit assignments on the instance depicted in Figure 3.7. The instance measures how the algorithms cope with exits having different distances to evacuees.

Evacuations would benefit from such approaches when for a given instance the earliest arrival flow equals (or at least: does not deviate much from) a Nash flow. It is still an open question on which instances these flows are equal. Instances can then be changed over time such that only paths that are also optimum paths in an equilibrium are available, e. g., by blocking. Such a procedure resembles toll strategies in algorithmic game theory and first research into this direction is due to Bhaskar, Fleischer and Anshelevich [BFA11].

3.3 Practice Example

The department for fire safety of the Technische Universität Berlin arranges fire drills for one of the larger buildings, the so-called *Telefunkenhochhaus*. We used two of those sessions in 2009 and 2014 to collect real data of an evacuation scenario to compare it with our result based on simulation in the cellular automaton and the result of an earliest arrival flow computation. The results are used to prove the accuracy of the simulation approach and to verify that instances generated by the the automated conversion described in Section 3.1.4 allow for reasonable results.

The Evacuation Setting. To compare the results of our approaches with measured results we first need to collect data. To allow for reasonable results we need both, the outcome of the evacuation and the starting data. We briefly describe our methods and show where we have to make reasonable guesses as result of limited resources. The *Telefunkenhochhaus* is a 22-storey building consisting mainly of offices and seminar rooms inhabited by university students and scientific research staff. In an evacuation scenario the inhabitants are evacuated using two main stairwells on the eastern and western outer faces of the building. The main entrance is only reachable from the first floor and by elevators and is thus not used in an evacuation process. We collected data similar to the approach taken by Chalmet, Francis and Saunders [CFS82]. Outside of the building we measured arrival times for each of the exits, times for different exits are synchronized by the launching of the fire drill. Each of the evacuees was assigned a number when leaving the building such that further information could be retrieved in an interview later without disturbing the evacuation process. Unfortunately, it was not possible to provide each person with their number, especially when larger groups arrived at the exits. When it was possible the evacuees were asked for the room where they started and its floor in the building. However, many evacuees did not remember the room they were coming from so that we could not use this information. In the end the starting floor of 80% of the evacuees was known together with the information which of the two emergency exits they had chosen. We didn't have enough personal resources to collect starting positions in the second fire test evacuation.

Defining the Model. The evacuation scenario is modelled in the ZET software suite. The building geometry was modelled after plans that were provided by the department for fire safety. The starting positions of the evacuees are distributed randomly depending on the number of inhabitants on each floor. Due to the fact that we do not know the starting position of some evacuees, we had to come up with an estimation. We did this by assuming such evacuees started in the same floor as evacuees arriving approximately at the same time. The surveyed and our final floor distribution is depicted in Figure 3.17. We expect the evacuees to have some reaction time until they start with the evacuation process. This time consists of realizing that a fire alarm has started and preparing for evacuation, e. g., by following the official guidelines to close windows and doors or to pack personal belongings.

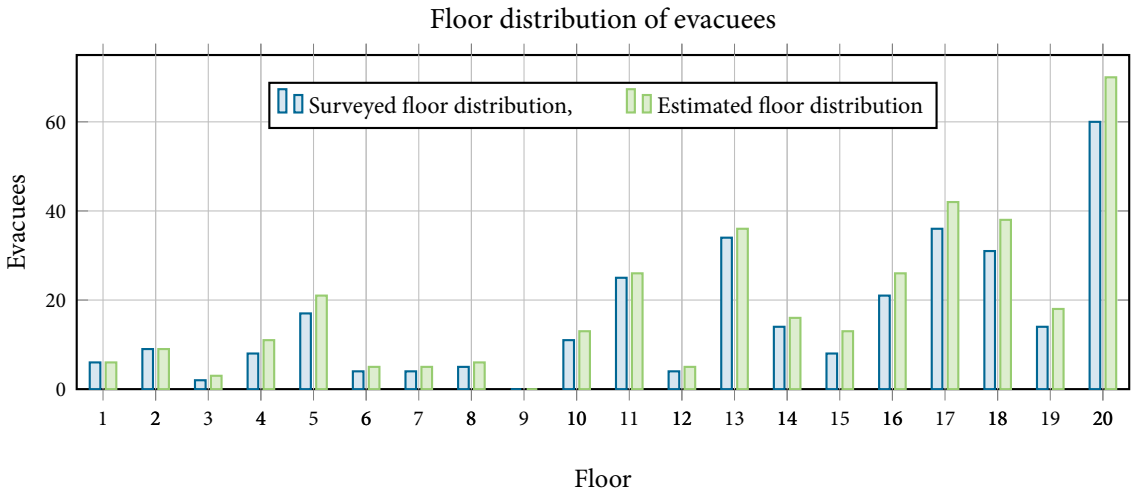
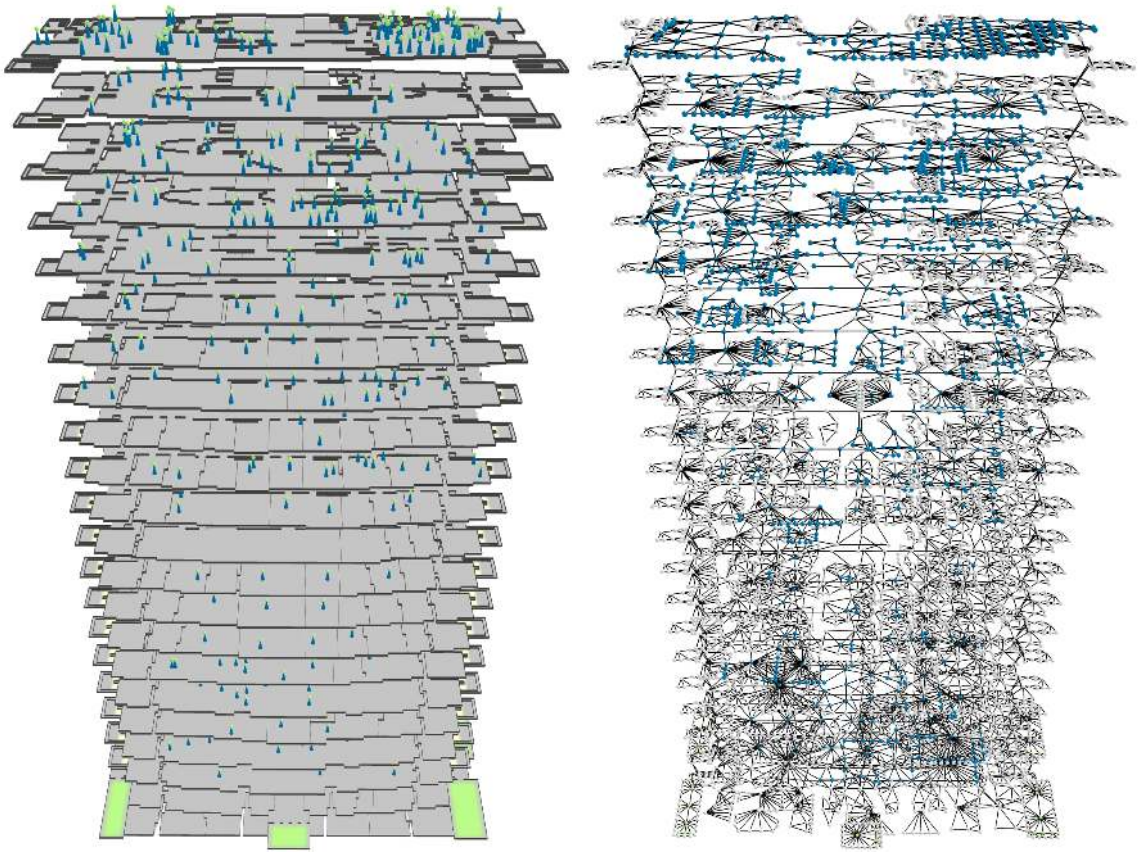


Figure 3.17: Comparison between the surveyed floor distribution and the estimated distribution used in the simulation/optimization runs.

We also expect colleagues in the same office to wait for each other. This is in accordance with our observation that the first evacuees needed more than a minute to arrive at the exit and then arrive clustered into small groups. For each evacuee we draw a normally distributed random reaction time in seconds with mean $\mu = 85$ and variance $\sigma^2 = 30$ which fits best to the measured arrival times and does not seem too unrealistic. Our experiments revealed, that it does not influence the results much, if the speeds of persons are derived by the age, because the variability of speeds is not very high for the test population. We therefore use a fixed speed of 1.51 m s^{-1} for each evacuee using data provided by Weidmann [Wei93].

Simulation and Optimization. The evacuation model was transformed into a cellular automaton and into a network model using rectangle rasterization. The complete cellular automaton model including an example distribution of the evacuees is depicted in Figure 3.18a. As for the experiments in Section 3.2 the simulation was repeated 200 times. In each repetition, evacuees were distributed according to the floor distribution and exit selection in the observed test evacuation.

The automatically generated network flow model consists of 6465 nodes and 23 170 including the individual start nodes and double edges to model that flow units can move in both directions of an arc. The model is depicted in Figure 3.18b. The final size of the time-expanded network depends on the transit times which is correlated with the speed of flow. Although both models are derived from the same evacuation model, they differ and cellular automaton steps do not directly relate with time steps in the discrete time-expanded network. We compare the influence of the modelling by an artificial scenario that only involves a single evacuee residing at the very top of the building. In the cellular automaton, the individual covers a distance of 570 m while a flow unit from the same position only travels 467 m (within the errors induced to rounding). To make the models comparable we decided to increase all transit times by 10 percent. The evacuees in the simulation have an average speed of 1.51 m s^{-1} and therefore flow needs approximately 0.265 s to travel along one arc with unit transit time.



(a) Cellular Automaton model. (b) Graph model.

Figure 3.18: The test evacuation instance as cellular automaton and network flow model within the visualization of the ZET software.

Using the above transit time scaling, the optimal time horizons for all our optimization runs lie in the interval [1588, 1619]. To estimate the time horizon beforehand, we use the bounds derived in Observation 2.8. After the transit time transformation, the estimated time horizon lies in the interval [1200, 1600] for all our runs. Notice, that the rough upper bound does not influence the running time if algorithms are implemented carefully. This is the case for Tjandra's implicit successive shortest path algorithm [Tja03] and the successive shortest path algorithm in the increasing time-expanded network. We need to reserve space for the larger time-expanded network. This was no problem for the instance depicted here. Using the bounds the time-expanded networks of instances for the *Telefunkenhochhaus* typically comprise 10 344 000 nodes and 36 840 300 edges.

Results. Both observed arrival curves, the simulated result and the obtained earliest arrival curve are depicted in Figure 3.19. The evacuation runs took place at 2nd September, 2009 and 29th April, 2014 and involved 369 and 365 inhabitants leaving the two emergency exits, respectively. The total evacuation time for the first evacuation run is 8.5 min while the second run was about 20 s faster.

The simulation generally follows the measured arrival curves, however simulation results are too

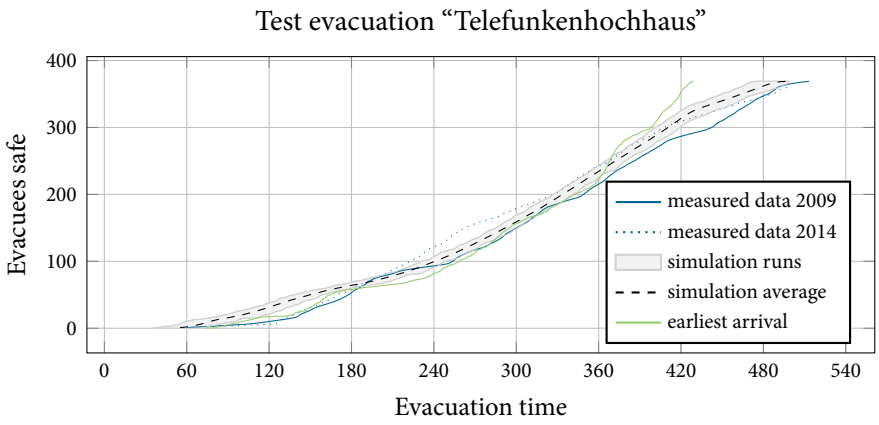
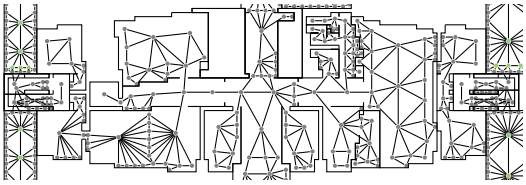


Figure 3.19: Results of a test evacuation, simulation using the cellular automaton model and comparison to with the earliest arrival flow.

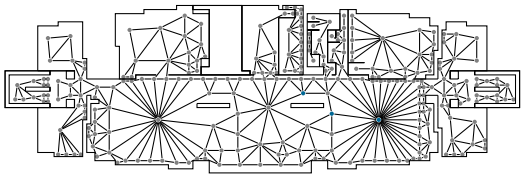
fast at the beginning in the second and third minute of the evacuation process. Afterwards, the simulation results lie in between the two measured outcomes. We consider the results very good, compared to the fact that we did not know the actual distribution of evacuees into rooms and the actual reaction times. The simulation also is a bit too fast in the last minute of the evacuation process. From personal observation of the evacuation process, this might be the result of few evacuees having a very large reaction time. They only start evacuation after request from staff of the department of fire safety and such behaviour is not covered by the simulation.

The earliest arrival flow (after adjusting travel times as mentioned above) gives similar results in the beginning. In the beginning, the results of the earliest arrival flow are even better than the results of the simulation runs. However, after 6 min, the earliest arrival pattern grows much faster than the actual and the simulated pattern such that the total evacuation time is 1 min better than the measured time. This discrepancy might have the same reason as for the simulations.

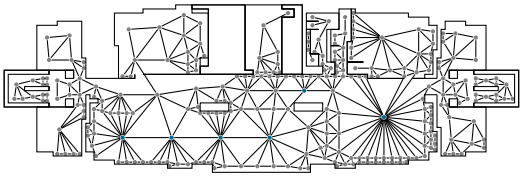
Discussion. The results of the simulation and optimization runs show that both are able to give accurate results even for larger evacuation scenarios. The automatic room conversion proves to be practical. Besides the accuracy the algorithms are fast on huge instances. It is also easy to implement the rasterization on top of existing evacuation tools that use cellular automata. The algorithms are also not limited to square cells, but also work on hexagonal cells. However, as a further direction of research a more clever rasterization process may be desired. Another improvement is the estimation of the actual transit times with much higher accuracy by using a cellular automaton itself.



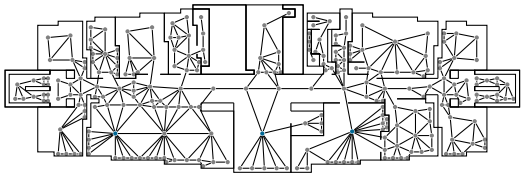
(a) 1st (ground) floor.



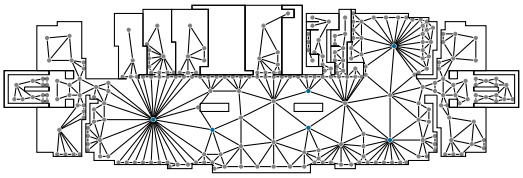
(b) 2nd floor.



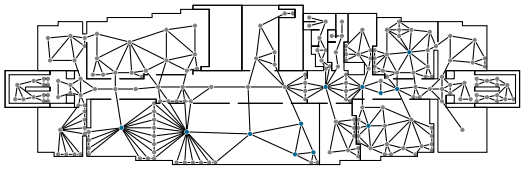
(c) 3rd floor.



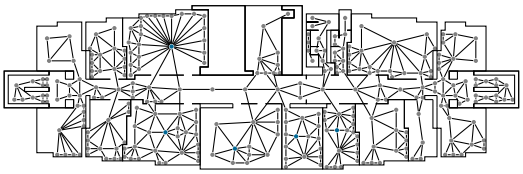
(d) 4th floor.



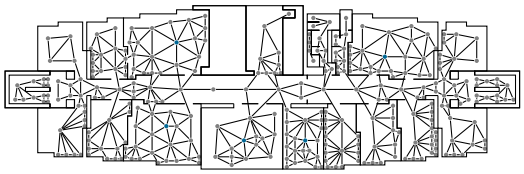
(e) 5th floor.



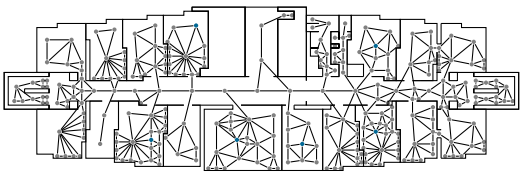
(f) 6th floor.



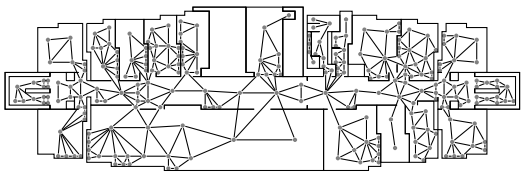
(g) 7th floor.



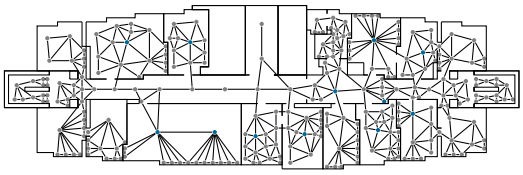
(h) 8th floor.



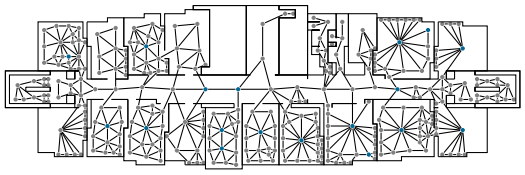
(i) 9th floor.



(j) 10th floor.

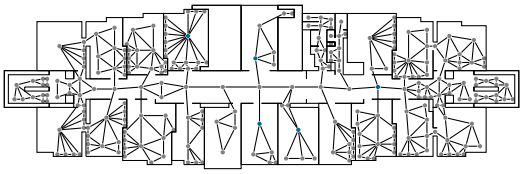


(k) 11th floor.

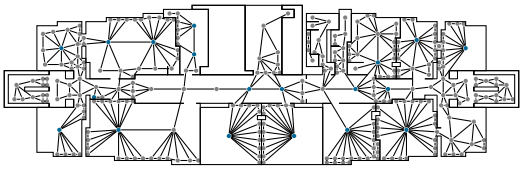


(l) 12th floor.

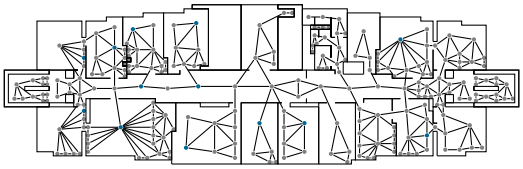
Figure 3.20: ZET automatically generated floor plan.



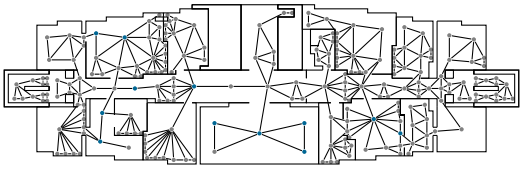
(m) 13th floor.



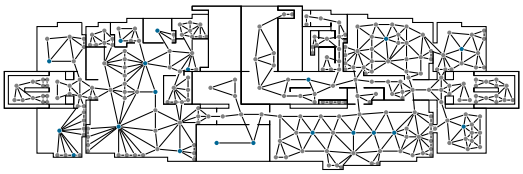
(n) 14th floor.



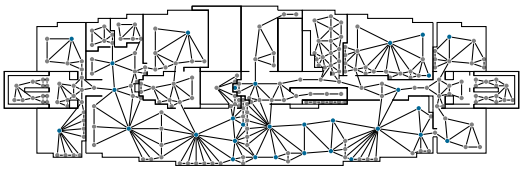
(o) 15th floor.



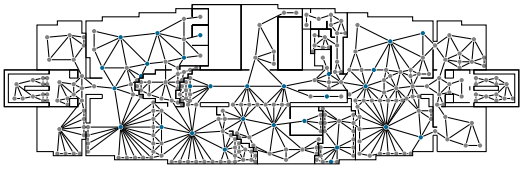
(p) 16th floor.



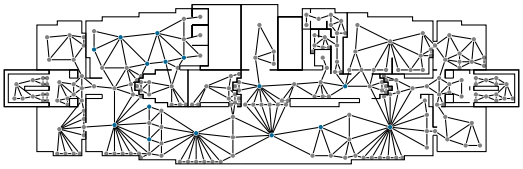
(q) 17th floor.



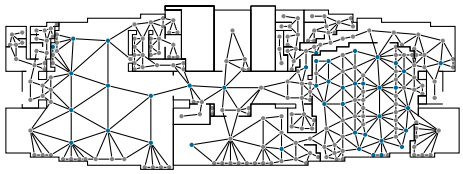
(r) 18th floor.



(s) 19th floor.



(t) 20th floor.



(u) 21th floor.

Figure 3.20: ZET automatically generated floor plan.

4 Approximating Earliest Arrival Flows

It is well known that earliest arrival transshipments do not exist in general on instances containing more than one sink. Approximation algorithms that approximate feasibility of earliest arrival flows have only rarely been studied so far. In this chapter we analyse the effectiveness of time-approximate earliest arrival flows and value-approximate earliest arrival flows. In the former, for flow it is allowed to arrive a certain factor later, while in the latter model the amount of flow arriving in each point in time is approximated, which we consider more useful in evacuation scenarios in general. We present a general approximation framework for this case and study implementations for the classical setting where we show that the algorithm allows good approximations in both the discrete and continuous setting and present a constant approximation-algorithm with tight lower and upper bounds. We also extend the results to multi-commodity flows over time.

Publication Remark: Some of the results in this chapter are joint work with Martin Groß, Daniel R. Schmidt, and Melanie Schmidt and have been published in [GKS+12]. The publication was awarded *Best Student Paper* at the European Symposium of Algorithms 2012.

In transportation models that are based on network flow over time models, it is a natural question to ask for earlies arrival flows, not only in the special case of evacuation scenarios. The general idea behind earliest arrival is the following: If some flow has to be sent (either within a given time horizon, or the demand is given specifically), it is often better if flow arrives earlier. This can be motivated by several reasons. In the evacuation scenario the available evacuation time is not specified, and it is better if at each point in time more evacuees are safe. But in general transportation problems it is just as natural to ask for the earliest arrival property. From the economic point of view, transportation creates cost and it may be cheaper, if goods arrive earlier. Also, consider a big online mail order company. If deliveries are assigned to routes in such a way that they arrive as early as possible, the customers are likely to be happy.

Following from our discussion in Section 2.4, we know already that earliest arrival flows exist and that they can be computed exactly in pseudo-polynomial time in many cases. For non-negative travel times, earliest arrival flows exist in general, if only a single sink is given [Fle01; BS09]. For zero travel times the result can be extended in such a way that earliest arrival flows exists in all networks containing a single source *or* a single sink. This is, because in instances with zero travel times the arc directions can be inverted without changing the resulting arrival pattern. A complete characterization of the class of graphs allowing for an earliest arrival transshipment, regardless of the arc capacities and supplies and demands is due to Schmidt and Skutella [SS14].

Rating of Arrival Patterns. By non-existence of earliest arrival flows we mean that there is no feasible single flow for a given instance that is maximum at each point in time. Notice, that obviously for each point in time there exists a flow that sends as much as possible *for this point in time*. For a

given flow over time we are interested in measuring its quality. This is not only useful in cases where no earliest arrival flow exists, but also in other settings. Examples for these kind of approximations are given in [HT94; FS07; GS12a; BK07].

The quality of a given arrival pattern can be measured in different ways. We consider two measurements: Time-approximations allow flow arrive too late; value-approximations approximate the flow value at each point in time. Notice that (motivated by our application of evacuation) we consider worst-case approximation factors. On a more general level one may also ask for cumulated approximation measurements. Consider two flows, the first sends one unit of flow less at almost every point in time, while the second sends two units of flow less, but only in a very small time interval.

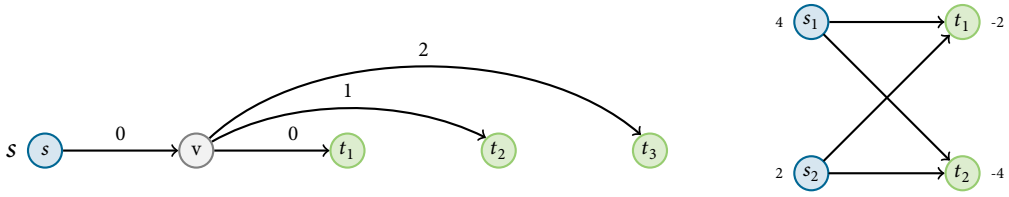
Outline of the Chapter. In Section 4.1 we begin with an introduction to quality measurements of earliest arrival flows, give some examples and show simple facts. We continue with discussing time-approximations in Section 4.2, show existence of a constant 4-approximation factor and a lower bound of 2 for this case. In Section 4.3 we establish a framework for value approximate flows. We then use the framework to derive constant approximation algorithms for standard network flows and multi-commodity flows. For the special case of zero travel times we show how the algorithm can be implemented in polynomial time. We conclude this chapter by showing the existence of an algorithm that approximates the optimal earliest arrival pattern for a given *instance*, which can be better than the constant bound guaranteed by the algorithm from Section 4.3, in Section 4.4. This is important because if the instance is not a worst-case scenario the real approximation factor might be much better than the bounds we establish in Section 4.3.

4.1 Approximate Earliest Arrival Flows

It is easy to see that earliest arrival flows do not exist in graphs containing several sinks. The reason is fairly simple: It may very well be that flow on paths arriving at a sink t at time θ blocks all paths arriving at another sink t' at the same time. If the earliest arrival time 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 pattern at the optimal arrival time for t'. The *fan graph* of Example 4.1 is a simple network having this property. We will use it throughout the chapter as counter example to derive lower bounds. In the scenario with zero-transit times the situation is not so clear. All paths have total zero travel time, hence the above situation cannot occur as easily. When two paths use a common arc, the capacity of the shared arc serves as a bottleneck also for the maximum flow and thus flow that has to wait does not arrive too late. In fact, for zero travel times earliest arrival transshipments exist always in instances having a single source. As an introduction to instances that do not allow for an earliest arrival flow, we consider two examples.

Example 4.1 (Fan graph)**.** *As an example we consider the* ***fan graph*** *with k sinks. The graph contains two additional nodes s and v and the edges (s, v) and (v, t_i) for all $i \in \{1, 2, \dots, k\}$. The travel time on arc (v, t_i) to sink i is defined as $i - 1$. Thus, in the example it is possible to send i units of flow up to time θ. However, flow that should arrive at sink t_θ at time θ must use arc (s, v) at time 0 making it impossible to maximize flow at two points in time. Figure 4.1a depicts an instance of the fan graph for $k = 3$. The general example and the pattern are depicted in Figure 4.5.*

While the fan graph requires non-zero travel times on the arcs, similar instances exist for the case



(a) The fan graph for $k = 3$. The transit times are as specified in the figure.

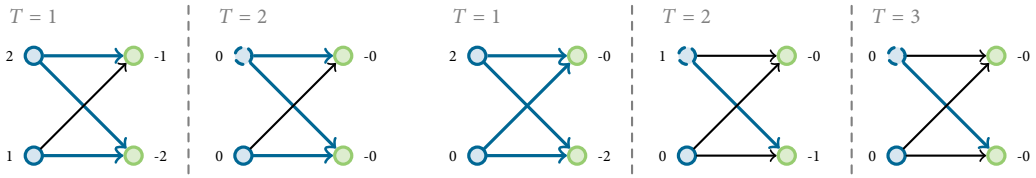
(b) A simple instance with zero travel times.

Figure 4.1: Examples that do not allow for an earliest arrival flow. The capacities are unit capacities.

of zero-travel times. However, in this case it is necessary to have multiple sources and sinks, and the smallest example needs exactly two terminals of each type.

Example 4.2. *We consider the instance with two sources s_1 and s_2 and two sinks t_1 and t_2 as depicted in Figure 4.1b. All arcs have a transit time of 0 and a capacity of 1. From both sources it is possible to send one unit of flow to each sink in one time step. The existence of an earliest arrival flow in the network depends how the supplies and demands are distributed among the terminal nodes. Consider the case $b_{s_1} = -b_{t_2} = 4$ and $b_{s_2} = -b_{t_1} = 2$. It is possible to satisfy the balances within two time steps. A quickest transshipment sends three units of flow at time $\theta = 1$ and $\theta = 2$ as depicted in Figure 4.2a. However, this pattern does not send as much flow as early as possible. By sending greedily as much flow as possible an additional unit of flow can be sent along arc (s_2, t_1). Thus, four units of flow arrive at time $\theta = 1$. Because the supplies and demands of s_2 and t_1 are satisfied after sending a maximum flow in the first time step, the last two flow units have to be sent along arc (s_1, t_2) which requires another two time steps. To satisfy the earliest arrival property, we have to send four units of flow in the beginning, but also have to send six units of flow within time $\theta = 2$. This is a contradiction and thus no earliest arrival flow exists in the instance.*

Approximation of Earliest Arrival Flows. The non-existence of earliest arrival transshipments in the case of multiple sinks gives rise to the question, whether earliest arrival transshipments can be approximated. We consider two kinds of approximation. The first relaxes the time, i. e., we are allowed to send the flow by a factor later than the optimal pattern requires. The second approximation relaxes the value, i. e., we only have to send a given factor of the maximum flow at each given



(a) A quickest transshipment within time horizon $T = 2$.

(b) A transshipment within time horizon $T = 3$ sending as much flow as possible in the first time step.

Figure 4.2: Two feasible solutions in the discrete setting for the zero travel time instance depicted in Figure 4.1b.

point in time.

Definition 4.3 (α-time-approximation)**.** Given an earliest arrival pattern p^* for a time horizon T, a flow over time f is an α-**time-approximation**, if at every point in time $\theta \in [0, T[$ we require

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

i. e., at least as much flow is sent to the sinks until θ as was possible at time $\frac{\theta}{\alpha}$. If the flow is additionally delayed by a constant $c > 0$, i. e.,

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

we say that f is a c-**delayed** α-**earliest arrival transshipment**. ◁

Time-approximation allows flow to be late. This notation has proven to be useful for finding approximation algorithms for the earliest arrival transshipment, for example the $(1 + \varepsilon)$-algorithm by Fleischer and Skutella [FS07]. However, we are more interested in the *value* at each point in time. The information "*After two hours, at least as many persons are secure, as were possible after one hour*" does not give all the information that we wish for in evacuation scenarios. This leads to the following definition.

Definition 4.4 (β-value-approximation)**.** Given an earliest arrival pattern p for a time horizon T, a flow over time f is a 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 maximum possible flow is sent until θ. ◁

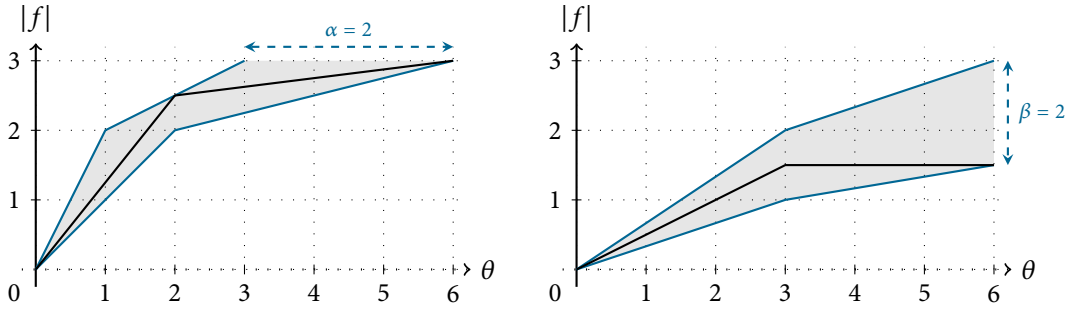
Value approximations allow similar approximation algorithms for the Earliest Arrival Transshipment Problem relaxing the feasibility. Value approximation algorithms have been presented by Hoppe and Tardos [HT94].

By specifying a desired approximation factor α or β, we have a fixed corridor in which feasible approximate flows can lie. If a flow f is both, an α-time and a β-value-approximation, we also say the flow is an (α, β)-**time-value-approximation** of the earliest arrival pattern. An example of both approximation objectives is depicted in Figure 4.3.

In the remainder of the chapter we will analyse the possible values for α and β by giving approximation algorithms and also lower bounds. Most of the results hold for both, the discrete and the continuous flow models. We will therefore first discuss some properties that allow us to relate results for the two flow models.

Continuous and Discrete Approximate Earliest Arrival Flows. Any flow in the continuous time model induces a flow in the discrete time model that has an identical flow value at each integer point in time. The discrete flow can be computed by sending the cumulated continuous flow of one time step as a whole:

Lemma 4.5. *A network that does not allow for an α-time-approximate (β-value-approximate) earliest arrival transshipment in the discrete time model does also not allow for an α-time-approximate (β-value-approximate) earliest arrival transshipment in the continuous time model.*



(a) An $\alpha = 2$-example. (b) A $\beta = 2$-example.

Figure 4.3: Examples of earliest arrival approximations. The grey area is valid for the given approximation factor. Any flow whose arrival curve lies within the area is a feasible approximation. The upper curve equals the pattern.

Proof. Let $\mathcal{N}$ be a network that does not allow for an α-time / β-value-approximate earliest arrival transshipment in the discrete time model. We assume there is an α-time / β-value approximate earliest arrival transshipment f in $\mathcal{N}$ in the continuous time model with time horizon T. Thus, the flow value satisfies $|f|_\theta \geq p^*(\frac{\theta}{\alpha})$ or $|f|_\theta \geq \frac{p^*(\theta)}{\beta}$, respectively, for all points in time. In particular this holds for all $\theta \in \{1, \dots, T\}$.

We define a flow f in the discrete time model by cumulating the flow between the integral points in time and define

$$f'(e,\theta) := \int_0^\theta f(e,\xi)\,d\xi - \int_0^{\theta-1} f(e,\xi)\,d\xi = \int_{\theta-1}^\theta f(e,\xi)\,d\xi$$

for all $\theta \in \{1, \dots, T\}$ and $e \in E$. The flow equals the total flow on an arc e within a time interval $[\theta - 1, \theta[$.

f' is feasible in the discrete time model and satisfies $|f'|_\theta = |f|_\theta$ for all $\theta \in \{1, \dots, T\}$ since

$$\sum_{\tau=1}^{\theta} f'(e,\tau) = \sum_{\tau=1}^{\theta} \int_0^\tau f(e,\xi)\,d\xi - \int_0^{\tau-1} f(e,\xi)\,d\xi = \int_0^\theta f(e,\xi)\,d\xi - 0$$

holds for all $e \in E$. Notice also that the earliest arrival pattern p^* in the continuous model is at least as large as the earliest arrival pattern $\hat{p}^*$ for the discrete time model as any discrete flow induces a continuous flow with the same value. This gives

$$|f'|_\theta = |f|_\theta \geq p^*\left(\frac{\theta}{\alpha}\right) \geq \hat{p}^*\left(\left\lfloor \frac{\theta}{\alpha} \right\rfloor\right)$$

and

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

for the time- and value-approximation, respectively.

Thus, f' is an α-time / β-value approximate earliest arrival flow in the discrete time model, which is a contradiction to our initial assumption. □

We now prove a lemma showing that also upper bounds can be transferred from the discrete model to the continuous model. However, we will lose a factor of ε in the time-approximation.

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

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

Connections Between the Approximation Objectives. The time- and value-approximation models differ and we investigate relations between them. Ideally, we would like to establish a result that directly relates α-time and β-value transshipments. For example, any β-value-approximate flow can be sent β-times to send enough demand to establish an α-approximation with time horizon βT^*, where T^* is the optimal time horizon. However, it is not clear that the β-value-approximate flow satisfies a $\frac{1}{\beta}$-fraction of the demands of each terminal, such that the flow can be sent repeatedly. If we additionally require a structure on the convexity of the earliest arrival pattern, we can establish such a relation. Before stating the results, we will briefly recall some basic definitions.

Definition 4.7 (Concavity and convexity). Let $f : \mathbb{R} \to \mathbb{R}$ be a function. Then f is **concave** if

$$f(\xi x_1 + (1-\xi)x_2) \geq \xi f(x_1) + (1-\xi) f(x_2)$$

holds for all $x_1, x_2 \in \mathbb{R}, \xi \in [0,1]$. f is **convex** if

$$f(\xi x_1 + (1-\xi)x_2) \leq \xi f(x_1) + (1-\xi) f(x_2)$$

holds for all $x_1, x_2 \in \mathbb{R}, \xi \in [0,1]$. ◁

Observe, that if an arrival pattern is concave, flow has to arrive from time 0 on, i. e.,, the network must contain a shortest path of zero travel time. We will now show that any α-time-approximate flow with a concave arrival pattern is also a value approximate flow and that any β-value-approximate flow with a convex arrival pattern is also a time-approximate flow.

Lemma 4.8. *Let p^* be an earliest arrival pattern and f be a feasible transshipment. Then, if p^* is concave and f is an α-time approximation, it is also a $\beta = \alpha$-value-approximation, and if p^* is convex and f is a β-value approximation, it is also an $\alpha = \beta$-time-approximation.*

Proof. For the first statement, let $\alpha > 0$. First we show that for any concave non-decreasing function $p : \mathbb{R}_{\geq 0} \to X \subseteq \mathbb{R}_{\geq 0}$ with $p(0) \geq 0$ then holds

$$\frac{p(x)}{\alpha} \leq p\left(\frac{x}{\alpha}\right) \leq p(x).$$

By definition of concavity

$$\frac{p(x)}{\alpha} \leq \frac{p(x)}{\alpha} + \left(1 - \frac{1}{\alpha}\right)p(0) \leq p\left(\frac{x}{\alpha} + \left(1 - \frac{1}{\alpha}\right)\cdot 0\right) = p\left(\frac{x}{\alpha}\right) \tag{4.1}$$

holds. The second inequality follows from the monotonicity of p. Hence, Equation 4.1 holds and the given flow pattern p_f also is $\beta := \alpha$-value-approximate.

Let now the pattern be concave and $\beta > 0$. Any convex function $p : \mathbb{R}_{\geq 0} \to X \subseteq \mathbb{R}_{\geq 0}$ with $p(0) = 0$ satisfies

$$p\left(\frac{x}{\beta}\right) \leq \frac{p(x)}{\beta} \leq p(x).$$

Again, by definition of convexity we have

$$p\left(\frac{x}{\beta}\right) = p\left(\frac{x}{\beta} + \left(1 - \frac{1}{\beta}\right) \cdot 0\right) \leq \frac{p(x)}{\beta} + \left(1 - \frac{1}{\beta}\right) p(0) = \frac{p(x)}{\beta}.$$

The last equality holds for any arrival pattern because there is no flow arriving up to time 0. Hence, a given β-value-approximate pattern f is also $\alpha := \beta$-time-approximate.

□

The lemma states a transfer of upper bounds, but it can be used in the inverse direction to establish lower bounds, too. If a concave instance does not allow for a β-value-approximation, there also does not exist an $\alpha = \beta$-time-approximation. Notice that for the first statement of the lemma, there must be flow reaching a sink at time zero to apply the lemma, because otherwise at the point in time when the first flow arrives at a sink, concavity is violated.

4.2 Time-approximate Earliest Arrival Flows

We will first discuss the existence of time approximations. Only few results exist, that show the existence of a time-approximation. Based on time condensation Fleischer and Skutella [FS07] gain an FPTAS for the Quickest Flow Problem, and Groß and Skutella [GS12a] use the technique for computing *generalized flows over time*. Baumann and Köhler [BK07] compute a 4-time-approximation for networks with flow dependent transit times. The technique is applied by Groß to the multi-commodity setting with commodity dependent transit times [Gro09]. However, despite these already existing results, time-approximation is not a really useful tool for the scenario of approximating earliest arrival flows. It is easy to see that there is always a T-time-approximate earliest arrival flow, though.

Observation 4.9. *There is always a T-time-approximate earliest arrival flow in the discrete and continuous time model, where $T + 1$ is the time horizon of the quickest transshipment, if no flow arrives before time 1. T is also a lower bound.*

Proof. In the network in Figure 4.4, it is possible to have two units of flow arriving at time T or one at time 1 and one at time T^2, implying that an α-approximate earliest arrival flow has to satisfy $f(\lceil \alpha \rceil) = 1$ and $f(\lceil \alpha T \rceil) = 2$ (in the discrete time model). This is impossible unless $\alpha \geq T$. The same holds in the continuous time model by Lemma 4.5. □

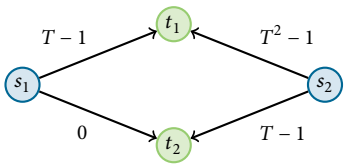


Figure 4.4: Counter example providing a lower bound of T. Edges have unit capacities, the supplies and demands are 1 and -1, respectively.

Single-source Networks. In the special case of networks with only one source, we give improved lower and upper bounds on the best possible time-approximation guarantee. The upper bound follows from the work of Baumann and Köhler [BK07]: They describe a technique that splits the time horizon into exponentially growing intervals in which they compute independent flows over time. Demand and supply vectors create a dependence between the intervals (which is why this does not work for multiple sources *and* sinks), but for the case of only one source, this dependence does not worsen the approximation guarantee compared to their original result.

Theorem 4.10. *For networks with one source s, there is always a 4-time-approximate earliest arrival transshipment in the discrete time model. The same is true in the continuous time model if the shortest travel time from s to any sink is positive.*

Proof. To prove the bound, we apply the interval stacking technique used for computing value-approximate s-t-earliest arrival flows with flow dependent transit times in [BK07].

First, assume that $T = 2^k - 1$ for some $k \in \mathbb{N}_{\geq 1}$, i. e., in particular $T > 0$. We partition the time frame $\{1, \dots, T\}$ into exponentially growing intervals $I_i := [2^i - 1, 2^{i+1} - 1[$ for $i \in \{0, \dots, k-1\}$. Notice that the length $\text{length}(I_i)$ of interval i is exactly 2^i. In the following, we compute a flow f_i to be sent during interval I_i and prove that the resulting compound flow is in fact the desired approximation. For this, let b_t^i be the remaining demand of sink $t \in S^-$ at the beginning of the i-th interval, i. e., before f_i is to be sent. We assume without loss of generality that the shortest path from s to any sink t has length at least 1 with respect to the transit times; this can be achieved by scaling since we assume that there is no length 0 path from s to any sink.

We now construct a flow f_i for $i \geq 0$ that is to be sent within interval I_i. Let f_i^* be a maximum flow with time horizon $\text{length}(I_i)$ with respect to the original demands b_t. Notice, that f_i^* can be sent completely within interval I_i, if the flow is shifted such that it starts later. We compute a path decomposition of f_i^* and denote by $\mathcal{P}_i^*(t)$ the paths that f_i^* uses to send flow into sink t, for all $t \in S^-$. We want to send flows $f_1, f_2, \dots, f_k$ one after another. Therefore, for f_i, we possibly cannot send all flow that is possible according to the max flow f_i^* if some flow was sent earlier. Thus, we define the flow f_i as the flow that arises from f_i^* by decreasing the flow along paths in $\mathcal{P}_i^*(t)$ in total by $b_t - b_t^i \geq 0$ for all $t \in S^-$.

Observe now, that the compound flow $f := f_1, f_2, \dots, f_k$ is a multiple deadline flow for the deadlines $1, 3, 7, \dots, 2^k - 1$, i.e. for the ends of all intervals. This is true because f_i^* is a maximum flow over time for the time horizon $\text{length}(I_i)$ and by the above construction, the compound flow sends at least as much flow as f_i^*. Hence, if we let I be the largest interval that ends before time $\theta \geq 1$ and define α_θ to be $\frac{\theta}{\text{length}(I)}$, we get that

$$|f|_\theta \geq p(\text{length}(I)) = p\left(\frac{\theta}{\alpha_\theta}\right).$$

For $0 \leq \theta < 1$, we get an approximation guarantee of 1, since no flow can reach the sinks before time 1 by our above assumption. This motivates setting $a_\theta := 1$ for $\theta \in [0, 1[$. We thus obtain an α-approximation with $\alpha := \max\{\alpha_\theta \mid \theta \in [0, T[\}$. It remains to show that $\alpha_\theta \leq 4$ for all $\theta \leq T$. Notice that for every $\theta \in I_i$, it holds $\theta < 2^{k+1} - 1$ and thus

$$\alpha_\theta < \frac{2^{i+1} - 1}{2^{i-1}} = 4 - \frac{1}{2^{i-1}} \leq 4.$$

In the second step we extend the construction to general T. Let k be the smallest integer such that $T \leq 2^{k+1} - 1$. We build the first k intervals $I_0, \dots, I_{k-1}$ as in the above construction, thus partitioning the time frame $[0, 2^k - 1[$. Additionally, we now add a $(k+1)$-th interval $I_k := [2^k - 1, T[$. We still have that $|f|_\theta \geq p(\frac{\theta}{\alpha_\theta})$ for all $\theta \leq T$ as above. Also, for $\theta \in I_0, \dots, I_{k-1}$, it still holds $\alpha_\theta \leq 4$ by the same calculation as before. Finally, if $\theta \in I_k$, then still $\theta < 2^{k+1} - 1$ holds and thus we have $\alpha_\theta < 4$.

The same technique can be applied in straightforward fashion to obtain the result for the discrete time model; however, we do not need to assume a positive source-sink path length in this case. □

If the path with shortest travel time to any sink has a length of zero we cannot use the interval stacking technique used in the proof of Theorem 4.10. However, it is possible to find a time approximation that comprises an additional additive delay.

Corollary 4.11. *There is an ε-delayed 4-time-approximate earliest arrival transshipment f_ε in the continuous model for networks with a single source and any $\varepsilon > 0$. More precise, the flow satisfies*

$$|f|_{\theta+\varepsilon} \geq p^*\left(\frac{\theta}{4}\right).$$

Proof. We first show how to compute a flow f_δ satisfying $|f_\delta|_{\theta+4\cdot\delta} \geq p\left(\frac{\theta}{4}\right)$ for $\delta > 0$ and then choose δ accordingly.

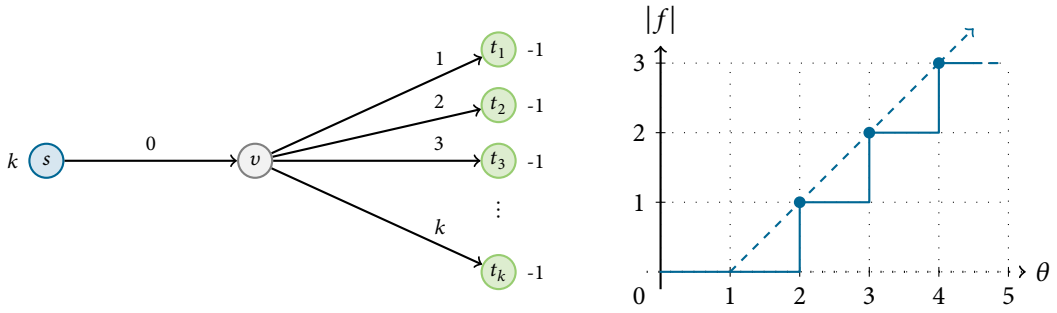
We create a new network $\mathcal{N}'$ based on the given instance $\mathcal{N}$ with source s by adding a new super source s^* and connecting it via the edge (s^*, s), and moving all supplies from s to s^* accordingly. Any flow in the new network $\mathcal{N}'$ corresponds to a flow in $\mathcal{N}$ that is delayed by $\delta > 0$. Let

$$p_\delta(\theta) = \begin{cases} 0 & \theta \leq \delta \\ p^*(\theta - \delta) & \theta > \delta \end{cases}$$

denote the delayed earliest arrival pattern in $\mathcal{N}'$.

We can now use Theorem 4.10 to compute the 4-time-approximate earliest arrival flow f_δ in $\mathcal{N}$. For any $\theta \in [0, T[$ it now holds that $|f_\delta|_{\theta+4\cdot\delta} = p_\delta\left(\frac{\theta+4\cdot\delta}{4}\right) = p^*\left(\frac{\theta}{4} + \delta - \delta\right) = p^*\left(\frac{\theta}{4}\right)$.

By choosing $\delta := \frac{\varepsilon}{4}$ we get the statement for $|f_\varepsilon|_{\theta+\varepsilon}$. □



(a) An example with a single source and k sinks. Arcs have unit capacity and the depicted transit time. Supplies and demands are k and -1, respectively.

(b) The discrete and continuous patterns for the network. Both patterns are equal at integral points in time.

Figure 4.5: Counter example providing lower bounds for both α-time and β-value approximate earliest arrival flows.

Lower Bounds for Time-approximate Earliest Arrival Flows. We will use the network depicted in Figure 4.5 to show a lower bound for α-time approximations, and later in Section 4.3 also to show the corresponding lower bound for β-value approximations of earliest arrival flows.

In the following, we need a notion of *weighted sum of arrival times*. To get this, we weight each flow unit by its arrival time. Let x be a path flow. Then the **weighted sum of arrival times** is defined as

$$\Xi_x \coloneqq \sum_{P \in \mathcal{P}} \sum_{\theta=1}^{T} x_P(\theta) \cdot \big(\theta + \tau(P)\big).$$

For a given edge flow, we compute the corresponding weighted sum of arrival times $\Xi_f \coloneqq \Xi_x$ as the weighted sum of arrival times of a corresponding path decomposition x.

In order to prove the existence of the lower bound, we show a connection between the weighted sum of arrival times and the number of demands sent by the flow in the network depicted in Figure 4.5. Let therefore f be a flow in the discrete time model and x be a path decomposition of f with $\mathcal{P}_f$ being the set of paths with positive flow value. For $P \in \mathcal{P}_f$ we denote by $\sigma(P)$ the point in time where f starts to send flow along P and by $\tau(P)$ the length of P. If f sends B units of flow, then the weighted sum of arrival times Ξ_e of f is at least $\lfloor B \rfloor^2 + \lfloor B \rfloor$.

Lemma 4.12. *Let f be a flow in the discrete time model that lives on a fan graph with k sinks connected via travel times $1, 2, \dots, k$. If f sends B units of flow, then the weighted sum of arrival times Ξ_f f with a path decomposition x satisfies*

$$\Xi_f \geq 2 \cdot \sum_{i=1}^{\lfloor B \rfloor} i = \lfloor B \rfloor^2 + \lfloor B \rfloor.$$

Proof. Let f be a flow on the network in Figure 4.5. We will first decompose f. Let P_i be the path (s, v, t_i) to sink t_i and $P_i(\theta) \coloneqq |f_{i,\theta}|$, where $|f_{i,\theta}|$ equals the amount of flow that f starts to send to sink t_i at time θ. We also define $\tilde{T}$ to be the last time at which f starts sending flow to a sink.

By this definition, we can write $\Xi_f = \sum_{i=1}^{k} \sum_{\theta=1}^{\tilde{T}} P_i(\theta) \cdot (\theta + i)$. It follows that

$$\begin{aligned}
\Xi_f &= \sum_{i=1}^{k} \sum_{\theta=1}^{\tilde{T}} P_i(\theta) \cdot (\theta + i) \\
&= \sum_{i=1}^{k} \sum_{\theta=1}^{\tilde{T}} P_i(\theta) \cdot \theta + \sum_{i=1}^{k} \sum_{\theta=1}^{\tilde{T}} P_i(\theta) \cdot i \\
&\geq \sum_{\theta=1}^{\lfloor B \rfloor} \theta + \sum_{i=1}^{\lfloor B \rfloor} i \\
&= \lfloor B \rfloor^2 + \lfloor B \rfloor.
\end{aligned}$$

The inequality is due to the fact, that we have to use at least $\lfloor B \rfloor$ paths and have to send flow out of the source for at least $\lfloor B \rfloor$ time units, which we use as a lower bound for the two sums. □

For the lower bound, we consider the network in Figure 4.5. We already know by Lemma 4.12 that the weighted sum of arrival times Ξ_f satisfies

$$\Xi_f \geq \lfloor B \rfloor^2 + \lfloor B \rfloor$$

if f is a flow in this network with flow value B. Additionally, every α-time-approximate earliest arrival flow f satisfies

$$\Xi_f \leq \sum_{i=1}^{k} \lceil \alpha \cdot (i+1) \rceil \leq 2k + \sum_{i=1}^{k} \alpha i$$

because $\alpha \geq 1$, thus implying the following theorem.

Theorem 4.13. *For every $\alpha < 2$, there exists a dynamic network with a single source that does not allow for an α-time-approximate earliest arrival flow. This holds in the discrete and continuous time model.*

Proof. Let f be an α-time-approximate flow on the network in Figure 4.5. The arrival pattern for the instance described in Figure 4.5 is given by $p^*(i) = i - 1$, for $i = 1, \dots, k+1$. This means that at least i flow units must arrive at the sinks up to time $\lceil \alpha \cdot (i+1) \rceil$. This implies that we can bound Ξ_f from above by

$$\Xi_f \leq \sum_{i=1}^{k} \lceil \alpha(i+1) \rceil \leq 2k + \sum_{i=1}^{k} \alpha i.$$

Together with Lemma 4.12, we get

$$\begin{aligned}
& \sum_{i=1}^{k} \alpha i \geq 2 \sum_{i=1}^{k} i - 2k \\
\Leftrightarrow \qquad & \alpha \geq 2 - \frac{2k}{k(k+1)/2} = 2 - \frac{4}{k+1}.
\end{aligned}$$

Thus, for every $\alpha < 2$, there exists a k such that the above example does not allow for an α-time-approximate earliest arrival flow. □

Zero Transit Times. We will shortly review the special case of zero transit times. One might expect to gain better approximation results for this restricted class of instances. However, even in the case of zero transit times, time-approximation is hopeless for multiple sources and sinks. This is unlike to β-value-approximations, which allow for a simple form of approximation in the case of zero transit times that can also be implemented to have polynomial running time.

Let T^* be the time horizon of a quickest flow in $\mathcal{N}$. In the discrete time model, there exists always an $\alpha = T^*$-time-approximate earliest arrival transshipment, because the quickest flow has this approximation guarantee as there is no flow arriving before time one (the same also holds in the continuous time model if no flow can arrive before time one). This cannot be improved much, as we see in the next lemma, and the general continuous case does not allow for any approximation.

Lemma 4.14 (Lower bound for zero transit times). *Let T^* be the time horizon of a quickest transshipment for a given earliest arrival transshipment instance.*

For every $\alpha \leq \frac{T^*}{2}$, there exists a network such that it does not allow for an α-time-approximate earliest arrival flow in the discrete time model.

For every finite α, there exists a network such that it does not allow for an α-time-approximate earliest arrival flow in the continuous time model.

Proof. We consider the family of instances $I_{C,U}$ for $C, U \in \mathbb{R}$ depicted in Figure 4.6. For the first lower bound, we only use instances of the form $I_{C,C}$ for $C \in \mathbb{R}$.

All supplies and demands can be satisfied in time C. In order to satisfy all demands, which any α-time-approximation has to do eventually, we must not use the edge (s_1, t_2) as all supply in s_1 is needed to satisfy the demand of t_1.

Therefore, any α-time-approximation has to send flow using only the edges (s_1, t_1) and (s_2, t_2), leading to flow arriving at the sinks with a rate of 2 until all flow units have been sent at time C. However, it is possible to send $C + 1$ flow units until time 1 by sending $C - 1$ flow units through (s_1, t_2) in addition to using (s_1, t_1) and (s_2, t_2). An α-time-approximation cannot send $C + 1$ flow units before time $\frac{C+1}{2}$, yielding a lower bound of $\frac{T+1}{2} > \frac{T^*}{2}$ for α because the time horizon of a quickest flow for this instance is $T^\star := C$. Therefore there are no $\alpha \leq \frac{T^*}{2}$-time approximations for this family of instances.

For the bound in the continuous model, consider the general family of instances $I_{C,U}$ and notice that an α-time-approximation cannot send C flow units before time $\theta := \frac{C}{2}$. However, we can now send C units using (s_1, t_2) in $\theta' := \frac{C}{U}$ time. For $C \to \infty$, θ' converges to 0 and thus $\alpha \geq \frac{\theta}{\theta'} \to \infty$. □

Lemma 4.15 (Upper bound for zero transit times). *Let T^* be the time horizon of a quickest transshipment for a given dynamic network. Then there exists a T^*-time-approximate earliest arrival flow*

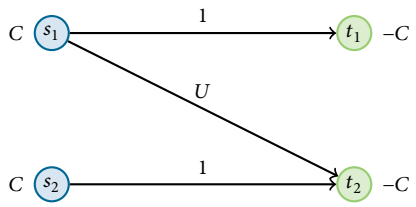


Figure 4.6: Instance $I_{C,U}$ with zero transit times $\tau \equiv 0$. Capacities are as specified on the edges, supplies and demands are as given next to the nodes.

in any network in the discrete time model.

Proof. Let f be a quickest flow for a given instance. By definition, f satisfies all supplies and demands in time T^*. In the discrete time model, we do not have flow arriving before time 1. Since f sends all possible flow at time T^*, f is an $\alpha = T^*$-time-approximation. □

Notice that the examples can be modified such that all sources can reach all sinks by adding the edge (s_2, t_1) with capacity C^{-1}. The existence of this edge makes the analysis slightly more involved but does not change the result.

4.3 A Constant Approximation Framework

In this section we will establish a general approximation framework based on a simple idea that allows for constant value-approximate earliest arrival flows, if certain conditions are met. In Sections 4.3.1 and 4.3.3 we will show that concrete implementations of the framework allow for constant approximations in the classical settings of earliest arrival transshipments, but also in the more elaborate setting of multi commodity flows.

So far, to the best of the author's knowledge, value-approximation of flows over time has not been studied, although it is the more intuitive form of approximation: At each point in time, we wish to approximate the maximum amount of flow at this time. We will see, that this notion of approximation is surprisingly strong, as it allows for 2-approximations in the general case of multiple sinks *and* sources, in contrast to time-approximation.

Increasing Time-expanded Networks. We establish a general algorithm based on time expansion for discrete flows over time. In contrast to known algorithms, we use time-expanded networks of increasing time horizon and start with a network $\mathcal{N}^1$ that only contains the original graph. We will iteratively compute network flows in the increasing time-expanded networks $\mathcal{N}^\theta$ for $\theta \in \{1, 2, \dots, T\}$ for some T large enough. This leads us to the general framework for value-approximate earliest arrival flows in Algorithm 4.1.

Obviously, we cannot prove any runtime bound for the algorithm as its runtime heavily relies on the computation of feasible flows f^i in step 4. Observe also, that the flow f^i has to remain the same in $\mathcal{N}^{i+1}$ which is a fairly strong requirement. Due to the time expansion, we expect any concrete implementation of the general algorithm to have super-polynomial running time. We will see concrete implementations later in Sections 4.3.1 and 4.3.3, and see that in the special case of zero transit times, the implementation can be improved to be polynomial.

The construction of a flow satisfying a specific value-approximation factor in step 4 is the crucial aspect for proving correctness of the algorithm. If such a flow can be found, any flow that is returned by the algorithm is a c-value-approximate earliest arrival flow.

Observation 4.16. *A concrete implementation of Algorithm 4.1 computes a c-value-earliest arrival flow for the given instance, if such a flow exists.*

Proof. Because we have a concrete implementation given, we can assume there exists the algorithm required in step 4 in the computation. The additional requirements make sure that the computed

Algorithm 4.1: General c-value-approximate Earliest Arrival Flow Algorithm

Input: An instance $\langle I \rangle$ of a network flow problem in any flow over time model, possibly with additional constraints, that allows for time expansion, $c \in \mathbb{R}$.

Output: A c-value-approximate flow over time or the statement that such a flow does not exist.

1. Define an increasing time expanded version $\mathcal{N}^i$ of the network specified by $\langle I \rangle$. This may be either for a sufficient time bound T, or dynamically increasing.
2. Set $i \coloneqq 1$ and compute a maximum flow f^1 in $\mathcal{N}^1$ according to the constraints of the model.
3. Let i be the current step and f^i be a feasible flow in $\mathcal{N}^i$.
4. Compute a feasible maximum flow f^{i+1} for the given model in $\mathcal{N}^{i+1}$ subject to the following additional constraints:

$$|f^{i+1}|_\theta \geq \frac{1}{c}|f^{\theta,*}|,$$

   where $f^{\theta,*}$ is a maximum flow for time horizon θ. If no such flow can be found, return for infeasible instance.
5. If all demands are satisfied by f^{i+1} or it is not possible to send additional flow because all reachable sinks have satisfied demands, return f^{i+1}. Otherwise, set $i \coloneqq i + 1$ and continue with 2.

flow is indeed a c-value-approximation. □

Existence Criterion. The general algorithm allows to check for existence for a given approximation factor c. However, we are more interested in an algorithm that *always* computes a feasible flow, preferably with a best possible upper bound. This is easily achieved by changing step 4 to act greedily as described in the following. Instead of requiring the flow to achieve at least a $\frac{1}{c}$-fraction of the maximum amount of flow, we can just enforce that the flow value for earlier points in time is *not allowed* to be changed later. Thus, proving an upper bound of c only requires that the flow computed in each iteration is upper bounded by c. This leads to the following general greedy-value-approximation earliest arrival flow algorithm.

The modified greedy version of the algorithm does not send exactly a c-fraction of the maximum flow any more, but possibly more. Therefore we cannot give a general bound on the approximation quality, this has to be proven for each concrete implementation. However, we can state the following lemma that gives a hint of what type of bound should be proven for any concrete implementation.

Lemma 4.17. *If the computed flows in each iteration satisfy* $|f^{i,*}| \geq \frac{1}{c} \cdot |f^{i,*}|$, *where* $f^{i,*}$ *is a maximum flow for time horizon* i *in the given model, then Algorithm 4.2 computes a* c*-value-approximate earliest arrival flow for the given instance.*

Algorithm 4.2: General Greedy Value-approximate Earliest Arrival Flow Algorithm

Input: An instance $\langle I \rangle$ of a network flow problem in any flow over time model, possibly with additional constraints, that allows for time expansion.

Output: A value-approximate flow over time.

1. Define an increasing time expanded version $\mathcal{N}^i$ of the network specified by $\langle I \rangle$. This may be either for a sufficient time bound T, or dynamically increasing.
2. Set $i \coloneqq 1$ and compute a maximum flow f^1 in $\mathcal{N}^1$ according to the constraints of the model.
3. Let i be the current step and f^i be a feasible flow in $\mathcal{N}^i$.
4. Compute a feasible maximum flow f^{i+1} for the given model in $\mathcal{N}^{i+1}$ subject to the following constraints:

$$|f^{i+1}|_\theta = |f^\theta|,$$

   where f^θ are the computed flows in time steps $\theta \leq i$.
5. If all demands are satisfied by f^{i+1} or it is not possible to send additional flow because all reachable sinks have satisfied demands, return f^{i+1}. Otherwise, set $i \coloneqq i + 1$ and continue with 2.

Proof. In each time step i, the computed flow sends at least a fraction of $\frac{1}{c}|f^{i,*}|$ by assumption of the lemma. Step 4 ensures, that any consecutively computed flows send at least as much flow in earlier time steps, and thus the computed flow is indeed a c-value-approximate earliest arrival flow. □

In the remainder of the section we will see two examples how the greedy algorithm can be implemented such that the implementation is practically usable and the result is good enough. We will complement the upper bounds with according lower bounds at least for the case of classical flows over time.

4.3.1 Classical Flows

For the case of classical flows, i. e., flows with only one commodity and constant transit times, we show that the General Greedy Value-approximate Earliest Arrival Flow Algorithm 4.2 can be implemented to produce 2-value-approximate earliest arrival transshipments. For flows in the classical model, the requirements in step 4 of the general algorithm can be easily met: The discussion in Section 1.2 shows, that augmenting path algorithms are a possible way to compute maximum flows over time in a residual network. We can force such an algorithm to ensure capacities in a time-expanded network by simply removing residual arcs (t_i^*, t_i^θ) connecting super terminals to temporal copies of sinks for past times. If the flow value on the residual copy cannot be changed, the flow at earlier times cannot be reduced. Notice, that it is not necessary to remove also the according forward arc, as the algorithm sends maximum flows and flow on the arc is therefore not changed. An example

of the situation is given in Figure 4.7c. This leads to the following concrete implementation of the general greedy constant approximation framework.

Algorithm 4.3: Greedy 2-value-approximate Earliest Arrival Flow Algorithm

Input: A dynamic network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$, and b supplies and demands.

Output: 2-value-approximate earliest arrival flow.

1. Create an increasing time-expanded network $\mathcal{N}^T$ for a sufficient time bound.
2. Set $i \coloneqq 1$ and compute a maximum flow f^1 in $\mathcal{N}^1$.
3. Let i be the current iteration, f^i be the computed flow for time i that is feasible in $\mathcal{N}^i$, and $\mathcal{R}^{i+1}$ be the (static) residual network induced by f^i in the time-expanded network $\mathcal{N}^{i+1}$. Define a new network $\mathcal{R}'^{i+1}$ by deleting all edges of the form (t_i^*, t_i^θ) for $\theta \in \{1, 2, \dots, i\}$ and all $t \in S^-$.
4. Augment f^i by s^*-t^*-paths in $\mathcal{R}'^{i+1}$ as long as such paths exist. Denote the resulting flow by f^{i+1}.
5. If no sink is reachable any more or all demands $\sum_{s \in S^+} b_s$ are satisfied, return f^{i+1}. Otherwise, set $i \coloneqq i + 1$ and continue with 3.

Notice, that step 4 is the computation of a maximum flow in the modified residual time-expanded network $\mathcal{R}'^{i+1}$, and as such satisfies the requirements of step 4 in the general greedy framework from Algorithm 4.2. To prove a constant bound for the value-approximation guarantee, we have to prove a statement that allows us to use Lemma 4.17. In fact, it is true that the maximum flow and the computed flows in each iteration differ at most by a factor of 2. The idea to see this is fairly simple: If it is possible to send more flow than we actually have done, this must be the case by using some of the removed arcs. This means, that for every flow unit that is possible to be sent, another flow unit at an earlier point in time is blocking. Thus we have a factor of two. This is stated in the following lemma.

Lemma 4.18. *Let $f^{\theta,*}$ be a maximum flow for time horizon θ and f^θ be the flow computed by Algorithm 4.3. Then it holds that*

$$|f^{\theta,*}| \leq 2 \cdot |f^\theta|.$$

Proof. We consider the difference flow that emerges from subtracting the flow f^θ from the maximum flow $f^{\theta,*}$ in a step of the algorithm. Let $g \coloneqq f^{\theta,*} - f^\theta$ be the difference flow that originates by sending $(f^{\theta,*} - f^\theta)$ on a forward arc e, if the value is positive, and sending $-(f^{\theta,*} - f^\theta)$ on reverse arcs $\overleftarrow{e}$ if the value is negative. Notice that for the flow values we now have $\left|f^{\theta,*}\right| = \left|f^\theta\right| + |g|$.

The constructed flow g is a valid flow in the residual network $\mathcal{R}^\theta$, but not necessarily in $\mathcal{R}'^\theta$. Consider any path P in the path decomposition of g. This path sends an additional unit of flow, that is not sent by f^θ. Because f^θ is a maximum flow and the path augmenting algorithm has not found another path, P must be an s^*-t^*-path using one of the deleted edges.

Notice now, that the total value of these paths is bounded by f^θ, because only residual arcs (corresponding to flow units that have arrived at an earlier time) were removed. Thus we have $|g| \leq |f^\theta|$ and consequently $|f^{\theta,*}| \leq |f^\theta| + |f^\theta| = 2 \cdot |f^\theta|$. □

Theorem 4.19. *Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network, and b supplies and demands. Then Algorithm 4.3 computes a 2-value-approximative earliest arrival flow in $\mathcal{N}$ in the discrete time model.*

Proof. Observe that Algorithm 4.3 is a concrete implementation of the General Greedy Value-approximate Earliest Arrival Flow Algorithm 4.2. Due to the bound $|f^{\theta,*}| \leq 2 \cdot |f^\theta|$ from Lemma 4.18 we can apply Lemma 4.17 proving that the resulting flow is in fact a 2-value-approximate earliest arrival flow.

□

Algorithm 4.3 has a pseudo-polynomial running time and only works in the discrete time model. However, together with the FPTAS described in Section 4.4 we obtain an efficient approximation.

Example 4.20 (Time-expanded Fan Graph)**.** *We consider the fan graph for $k = 3$ as depicted in Figure 4.1a. The time-expanded network in the beginning, and after the first and second iteration of Algorithm 4.3 are depicted in Figure 4.7.*

There is one path with zero travel time to copy t_1^1 of the first sink that is used in step 2 for the initial maximum flow computation. When the available time horizon is increased, it is possible to send one unit of flow in the residual network. However, the augmenting path (as depicted in Figure 4.7c) would use the reverse arc created by the first maximum flow computation. Thus, the path is not augmented and for time $\theta = 2$ we miss the maximum flow value by the one flow unit already sent at time 1.

Algorithm 4.3 only works in the discrete model. However, there still is a 2-value approximate earliest arrival flow in the continuous model that loses additionally a bit with respect to the time approximation.

Corollary 4.21. *There exists a $(1 + \varepsilon, 2)$-time-value-approximate earliest arrival transshipment in the continuous model.*

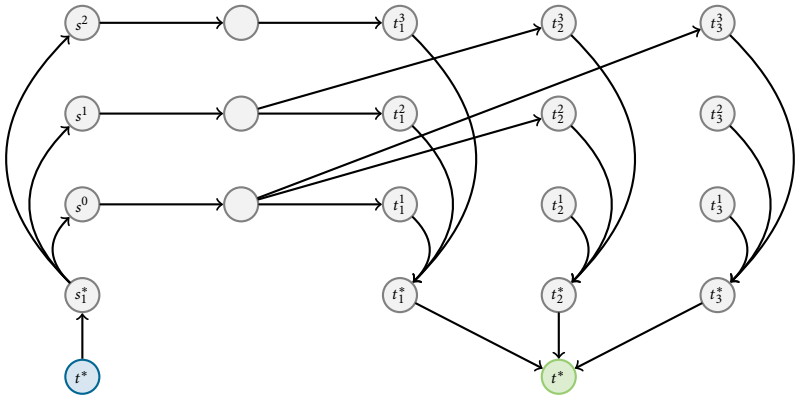
Proof. Consider the flow computed by Algorithm 4.3, that is a 2-value-approximate earliest arrival flow by Theorem 4.19. Then Lemma 4.6 directly gives the result. □

4.3.2 Lower Bounds for Value-approximate Earliest Arrival Flows

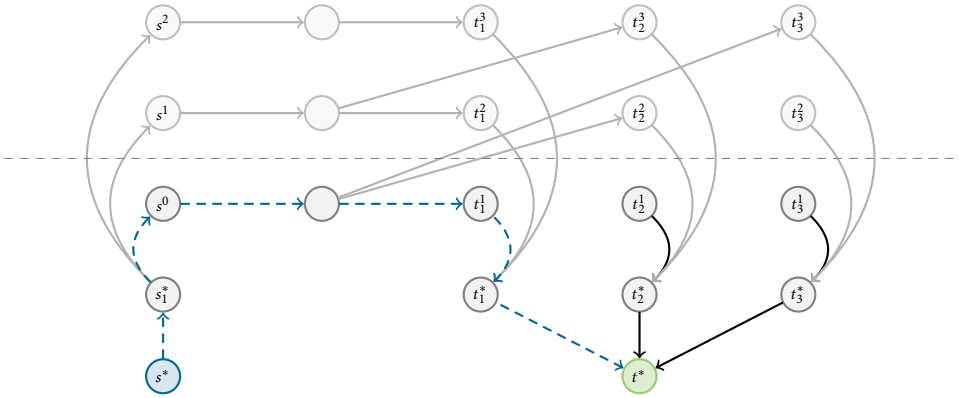
There exist simple examples of networks with only one source that provide also a lower bound of two. We will use the family of fan graphs depicted in Figure 4.5 to show that for any $\beta < 2$ there exists an instantiation for some k, such that the network does not allow for a β-value-approximate earliest arrival transshipment.

Theorem 4.22. *For every $\beta < 2$, there exists a dynamic network with only one source that does not allow for a β-value-approximate earliest arrival flow in the discrete and continuous time model.*

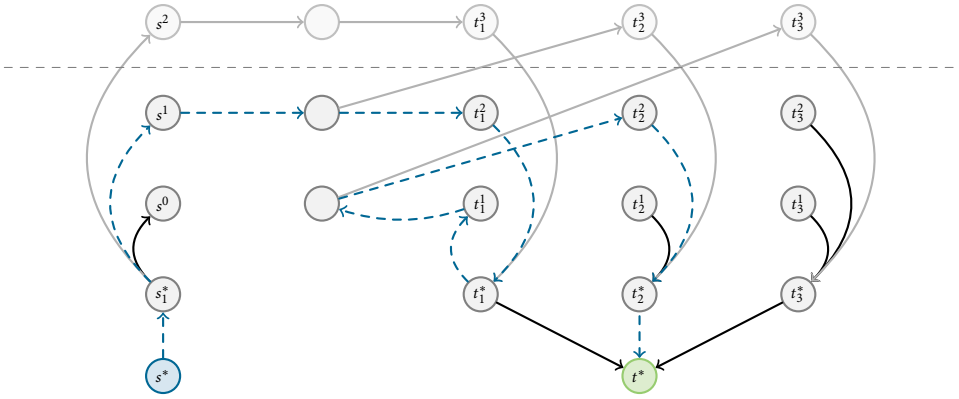
Proof. Consider the fan graph as depicted in Figure 4.5. The earliest arrival pattern is given by $p^*(\theta) = \theta - 1$ for $\theta \in \{1, 2, \dots, k + 1\}$ for both time models and is convex. Theorem 4.13 states that there is no α-time-approximation for any $\alpha < 2$. By Lemma 4.8 we know that a value-approximate



(a) The time-expanded fan graph for $k = 3$ and time horizon $T = 3$.



(b) The network $\mathcal{N}^1$ (all nodes and edges completely below the dashed grey line) as part of the time-expanded network. The dashed path is the s^*-t^*-path that is augmented in the first iteration.



(c) The network R^2 (all nodes and edges completely below the dashed grey line) as part of the time-expanded network. The dashed path (which is the only existing s^*-t^*-path) cannot be augmented in R'^2, as the dashed backward (t_1^*, t_1^1)-edge does not exist there.

Figure 4.7: The time-expanded network for the fan graph with $k = 3$ with time horizon $T = 3$ and the first two iterations of Algorithm 4.3.

earliest arrival transshipment would imply the existence of a time-approximate transshipment with the same factor. Thus, no β-value-approximation is possible for any $\beta < 2$. □

Zero Transit Times. We will simplify the Greedy 2-value-approximate Earliest Arrival Flow Algorithm 4.3 for the special case of zero transit times. The algorithm computes a maximum flow that is added to the flow f^θ in step 4. Due to the fact that all arcs only lie within a time layer, e. g., they have the form $e = (v^\theta, w^\theta)$ for some time θ, the max flow computation is mostly independent from the previously computed flow in earlier time layers. As long as the supplies and demands are not depleted, the *same* flow can be sent in each time. Thus we can repeatedly use the same static flow that can be computed in the original network until a source runs out of supply or a sink's demand is satisfied. This can not happen too often and we can calculate the time, when this will happen. In total, this yields a polynomial algorithm.

Algorithm 4.4: Zero Travel Time Value-approximate Earliest Arrival Algorithm

Input: An instance $\mathcal{N} = (V, E, u, \tau, S^+, S^-)$ of the earliest arrival flow problem with zero transit times $\tau_e \equiv 0$ and node balances b.

Output: A $\beta = 2$-value-approximation for the given instance.

1. Define new balances $b'_v := b_v$ and set $\theta := 1$.
2. Compute a static maximum transshipment f^θ respecting b'.
3. Determine the maximal amount of times r^θ the static flow f^θ can be sent until a source or sink runs empty:
   $$r^\theta := \min\left\{\left\lfloor \frac{b'_s}{|f^\theta|} \right\rfloor \,\middle|\, s \in S^+, b'_s > 0\right\} \cup \left\{\left\lfloor \frac{-b'_t}{|f^\theta|} \right\rfloor \,\middle|\, t \in S^-, b'_t < 0\right\}.$$
4. Update the balances according to the flow f^θ sent:
   $$b'_s := b'_s - r^\theta \cdot \mathrm{ex}(f^\theta) \text{ for } s \in S^+ \text{ with } b'_s > 0,$$
   $$b'_t := b'_t + r^\theta \cdot \mathrm{ex}(f^\theta) \text{ for } t \in S^- \text{ with } b'_t < 0.$$
5. If $b' \neq 0$, set $\theta := \theta + 1$ and continue with 2.
6. Output the flow over time that sends f^θ starting at $\sum_{j=1}^{\theta-1} r^\theta$ for r^θ time units.

For zero transit times, a maximum flow with time horizon T with supply / demands is equivalent to a static maximum flow in the network in which all edges are multiplied by T and the supplies / demands are represented by edges to super terminals. To show that the flow computed by the above algorithm is a 2-value-approximation, consider the residual network of the computed flow in the "multiplied" network, in which again the reverse edges of the super terminal edges are deleted.

Theorem 4.23. *Algorithm 4.4 computes a 2-value-approximate earliest arrival flow in dynamic networks with zero travel times.*

The flow can be computed with at most B static maximum flow computations in the continuous

time model and at most $B \log b_{max}$ *static maximum flow computations in the discrete time model, with* $B := |S^+ \cup S^-|$ *being the number of terminals and* $b_{max} := \max\{|b_v| \mid v \in S^+ \cup S^-\}$ *being the largest supply or demand.*

Proof. We have to show that the algorithm computes a feasible flow. At the beginning of step 3, there is at least one terminal which still has supply or demand. Because f^θ is a maximum transshipment respecting b', the number of iterations r^θ is at least 1. By definition of r^θ, the new values for supplies and demands computed in step 4 are feasible.

We show that the computed flow is indeed a 2-value approximate earliest arrival flow. Observe, that a maximum flow over time with zero transit times with time horizon T can be computed in the extended network $\mathcal{N}'$ that contains T copies of each arc and whose supplies and demands are shifted to newly introduced super terminals. The network has a one on one correspondence between an arc copy e^θ and the copy of the arc (v^θ, w^θ) on time layer θ in the time-expanded network. To prove the bound, consider the computed flow in the residual network of $\mathcal{N}'$ with respect to f in which the reverse edges of the super terminal edges are deleted. This leads to the same argument as in the proof of Theorem 4.19.

The algorithm performs one static maximum flow calculation per step. The choice of r^θ guarantees that at least one source or sink runs empty in every iteration in the continuous model, ensuring that $b' = 0$ after B iterations. Note that in the discrete time model, we have to choose an integer r^θ, so we get $\lfloor r^\theta \rfloor$ compared to the continuous case. This means that we have sent more than half of the supply or demand of a terminal since otherwise, we could have send f^θ at least one more time. This leads to at most $\log(b_{max}) + 1$ iterations per terminal, since our capacities and balances are integral. □

Lower Bounds for Zero Travel Times. In the following, we show that $\beta = 2$ is best possible even in the case of zero transit times. For this purpose, we again use time-expanded networks. Recall from Theorem 4.19 that in a time-expanded network, we have copies of the network in each time step and super-terminals that send flow into the layers or receive flow from the layers, respectively. Notice that due to zero-transit times, there are no edges between time layers.

Feasibility. The existence of a β-approximate earliest arrival flow can be related to a solution in a time-expanded network if the arrival pattern is known. This can be done for example by a linear program which then can be used to verify existence of approximate flows. We can then use duality theory to identify instances that do not allow for feasible value-approximate earliest arrival flows.

Problem:	Earliest Arrival Approximation Feasibility
Instance:	A dynamic network $\mathcal{N} = (G = (V, E), u, \tau \equiv 0, S^+, S^-)$ with zero transit times, supplies and demands for the sources and sinks $b : V \to \mathbb{R}$, approximation factor β.
Task:	Decide, whether a β-value-approximate earliest arrival flow exists in $\mathcal{N}$.

To further investigate this problem we need some notations. Let $\mathcal{N} = (G = (V, E), u, \tau \equiv 0, S^+, S^-)$ be a dynamic network with zero travel times and $\mathcal{N}^T = (G = (V^T, E^T), u^T, S^{+T}, S^{-T})$ the corresponding time-expanded network. Let $\mathcal{P}^\theta$ be the set of s^*-t^*-paths in $\mathcal{N}^T$ that use node copies v^θ

in time layer θ for each node $v \in V$ in the underlying static network. In the case of zero travel times $\mathscr{P}^\theta \cap \mathscr{P}^{\theta'} = \varnothing$ holds for two points in time $\theta \neq \theta'$ because paths do not leave their respective layer. We denote the set of all paths by $\mathscr{P} \coloneqq \bigcup_{\theta=1}^{T} \mathscr{P}^\theta$. For an earliest arrival pattern p^* we define $p' : \{1, 2, \dots, T\} \to \mathbb{R}$ to be the slope of the arrival pattern between two time steps, i. e.,

$$p'(\theta) \coloneqq \begin{cases} p^*(1) & \text{if } \theta = 1, \\ p^*(\theta) - p^*(\theta - 1) & \text{else.} \end{cases}$$

Using these preliminary definition we now define the following linear program, which corresponds to the Earliest Arrival Approximation Feasibility Problem.

$$\begin{aligned} \max \quad & 0, && && \text{(EAA)} \\ \text{s.t.} \quad & \sum_{P \in \mathcal{P} : e \in \mathcal{P}} x_P \leq u_e && \text{for all } e \in E_T, \\ & \sum_{P \in \mathcal{P}^\theta} x_P \geq \frac{p'(\theta)}{\beta} && \text{for all } \theta \in \{1, \dots, T\}, \\ & x_P \geq 0 && \text{for all } P \in \mathcal{P}. \end{aligned}$$

The dual of the linear program is the Earliest Arrival Cut Problem Problem which is specified in the following LP formulation.

$$\begin{aligned} \min \quad & \sum_{e \in \mathcal{N}_T} y_e u_e - \sum_{\theta=1}^{T} \frac{p'(\theta)}{\beta} z_\theta, && && \text{(EAC)} \\ \text{s.t.} \quad & \sum_{e \in P} y_e - z_\theta \geq 0 && \text{for all } \theta \in \{1, \dots, T\}, P \in \mathcal{P}^\theta, \\ & y_e \geq 0 && \text{for all } e \in E^T, \\ & z_\theta \geq 0 && \text{for all } \theta \in \{1, \dots, T\}. \end{aligned}$$

Notice that (EAA) basically consists of the path formulation for network flows in a time-expanded network. The LP additionally ensures, that at each point in time the arriving amount is exactly the necessary fraction of $\frac{1}{\beta}$ to get a β-approximate flow as solution. In order to show that the LP defines exactly the feasible instances, we use the following observation that shows how flow can be shifted to later points in time and still remains feasible.

Observation 4.24 (Observation 5 in [SS14]). *Let $\mathcal{N} = (G = (V, E), u, \tau \equiv 0, S^+, S^-)$ be a dynamic network with zero transit times and f be a feasible transshipment in $\mathcal{N}$ that sends a (static) flow f^θ at each point in time $\theta \in \{1, 2, \dots, T\}$. Let $\theta_1 < \theta_2 \in \{1, 2, \dots, T\}$ with $|f^{\theta_1}| \geq |f^{\theta_2}|$. Then, for any $x \in [|f^{\theta_2}|, |f^{\theta_1}|]$ there exist two static network flows $\hat{f}^{\theta_1}$ and $\hat{f}^{\theta_2}$ with*

$$\begin{aligned} |\hat{f}^{\theta_1}| &= x, \text{ and} \\ |\hat{f}^{\theta_2}| &= |f^{\theta_1}| - x + |f^{\theta_2}|, \end{aligned}$$

such that replacing f^{θ_1} by $\hat{f}^{\theta_1}$ and f^{θ_2} by $\hat{f}^{\theta_2}$ results in a feasible transshipment with the same value as f.

Proof. For the given $x \in [|f^{\theta_2}|, |f^{\theta_1}|]$ and any $a \in [0,1]$ we define a feasible flow $\hat{f}$ by setting the flow values for each point in time ξ as

$$\hat{f} \coloneqq \begin{cases} \hat{f} & \xi \neq \theta_1, \theta_2, \\ af^{\theta_1} + (1-a)f^{\theta_2} & \xi = \theta_1, \\ (1-a)f^{\theta_1} + af^{\theta_2} & \xi = \theta_2. \end{cases}$$

$\hat{f}$ is feasible because it is a convex combination of two feasible flows. The result follows by setting $a \coloneqq \frac{x - |f^{\theta_2}|}{|f^{\theta_1}| - |f^{\theta_2}|} \in [0,1]$. □

We can now prove that (EAA) in fact decides whether an instance is feasible.

Lemma 4.25. *A β-value-approximation in a network $\mathcal{N} = (G = (V,E), u, \tau \equiv 0, S^+, S^-)$ with zero transit times and supplies and demands b exists if and only if* (EAA) *has a feasible solution.*

Proof. Any feasible solution of (EAA) obeys capacity constraints, satisfies flow conservation and sends at least $\frac{p'(\theta)}{\beta}$ in every step $\theta \in \{1, 2, \dots, T\}$ by construction of the LP as path based network flow (cf. Definition 2.2). Feasible solutions can therefore be transformed into β-value-approximate earliest arrival flows.

The reverse direction is not immediately clear, because the constraint in the LP requires that in each time step at least $\frac{1}{\beta}$ times the increase in the earliest arrival pattern is sent. However, a β-value approximate earliest arrival flow is allowed to send more flow earlier and less flow later on. In the following, we will show that every β-value-approximation can be modified to send exactly $\frac{1}{\beta}p'(\theta)$ flow at each point in time θ by shifting flow from earlier time steps to later time steps, which then induces a feasible solution of (EAA).

We know that $|f^1| \geq \frac{p'(1)}{\beta}$ because f is β-value approximate. For an arbitrary time step θ, we ensure that the β-condition holds by shifting flow from time steps $1, 2, \dots, \theta - 1$ to time step θ, if necessary. We do so by (repeatedly) choosing an f^ξ for an adequate $\xi \in \{1, 2, \dots, \theta - 1\}$ with $|f^\xi| > \frac{p'(\xi)}{\beta}$ and applying Observation 4.24 to f^ξ and f^θ with $\theta \coloneqq |f^\xi| - \frac{p'(\xi)}{\beta}$. We repeat this until $|f^\theta| \geq \frac{p'(\theta)}{\beta}$. The time steps before θ must carry enough flow to establish this procedure, because otherwise

$$\sum_{\xi=1}^{\theta} |f^\xi| < \sum_{\xi=1}^{\theta} \frac{p'(\theta)}{\beta} = \frac{p^*(\theta)}{\beta}$$

holds, which is a contradiction to f being β-value-approximate. □

Using duality we can also establish a feasibility criterion for the dual problem.

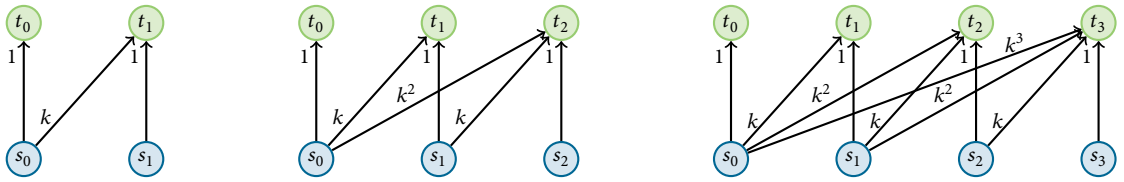
Lemma 4.26. *A β-value-approximation in a network $\mathcal{N} = (G = (V,E), u, \tau \equiv 0, S^+, S^-)$ with zero transit times and supplies and demands b exists if and only if* (EAC) *is bounded.*

Proof. By Lemma 4.25 we know that a β-approximate earliest arrival flow exists, if and only if (EAA) has a feasible solution. Observe that (EAC) is the corresponding dual program, and that setting all variables to 0 is a feasible solution of the dual, hence the primal LP is feasible if the dual is bounded. □

Before stating the main result that 2 is also a lower bound for the best possible approximation factor for the case with zero-transit times, we will consider the following motivating example.

Example 4.27. *We will define a class of dynamic networks using ℓ as a parameter for both the number of sources and sinks in the network, and k as a parameter for their respective capacities. We then define $\mathcal{N}_{\ell,k} = (V_{\ell,k}, E_{\ell,k}, u, \tau_0, S^+, S^-)$ by using nodes $V_{\ell,k} := \{s_i, t_i \mid i \in \{0, 1, \dots, \ell\}\}$, edges $E_{\ell,k} := \bigcup_{i=0}^{\ell} \{(s_i, t_j) \mid j \in \{i, i+1, \dots, \ell\}\}$, sources $S^+ := \bigcup_{i=0}^{\ell} s_i$, sinks $S^- := \bigcup_{i=0}^{\ell} t_i$, capacities $u_{(s_i,t_j)} := k^{j-i}$ for all $(s_i, t_j) \in E$ and balances $b_{s_i} := k^\ell$, $b_{t_i} := -k^\ell$ for all $i \in \{0, 1, \dots, \ell\}$. Figure 4.8 depicts the instances $\mathcal{N}_{1,k}$, $\mathcal{N}_{2,k}$ and $\mathcal{N}_{3,k}$.*

The idea behind the construction of $\mathcal{N}_{\ell,k}$ is that maximizing the flow for different time steps requires flow to be sent from different sources to different sinks, making it impossible to extend a maximum flow for one time step to be (nearly) maximal for all time steps. Consider network $\mathcal{N}_{3,k}$ with k sources and k sinks, as depicted in Figure 4.8c. To maximize the flow after one time step, s_0 must send (nearly) k^3 units of flow to sink t_3. Maximizing the flow at time k requires s_0 to send its k^3 flow units into t_2 instead of t_3, while s_1 sends its flow to t_3. For time step k^2, the destination sinks are again completely different: Now s_2 can also send k^3 flow units (to t_3) but only if s_1 instead sends its supply to t_2 and s_0 sends flow to sink t_1. Finally, to maximize the flow at time k^3, every source s_i has to send its supply to the corresponding sinks t_i. Alltogether, for each of the points in time $1, k, k^2$ and k^3, the source-sink pairs that have to be used are completely different.



(a) The instance $\mathcal{N}_{1,k}$.

(b) The instance $\mathcal{N}_{2,k}$.

(c) The instance $\mathcal{N}_{3,k}$.

Figure 4.8: Example of the instances $\mathcal{N}_{\ell,k}$ for $\ell = 1, 2, 3$. The supplies and demands are defined to be k^ℓ.

Theorem 4.28. *For every $\beta < 2$, there exists a network with zero transit times which does not allow for a β-value-approximate earliest arrival flow.*

Proof. We show that for the family of networks $\mathcal{N}_{\ell,k}$ with supplies and demands b as described in Example 4.27, for every $\beta < 2$, there is an ℓ and a k, such that the corresponding instance of (EAA) is unbounded.

Consider an instance of dynamic network $\mathcal{N}_{\ell,k}$. These instances require flow to be sent from different sources to different sinks to maximize flow at different points in time. A β-value-approximation algorithm can mediate between the different time steps by only partly sending the maximum flow, but the more sources and sinks we have, the more conflicting source-sink pairs come into play that make the mediation harder. We will see that for every $\beta < 2$, there are values for ℓ and k such that this mediation has to fail.

We prove this by the technique outlined above the theorem: We define values z_θ such that we can find a cut with costs less than $\sum_{\theta=1}^{k^\ell} z_\theta \cdot p'(\theta)$ which ensures that for all θ, every path in $\mathcal{P}^\theta$ is cut

at least z_θ times. As time horizon we choose $T = k^\ell$. Thus, our time-expanded graph contains k^ℓ copies of $\mathcal{N}_{\ell,k}$, and (EAC) has k^ℓ variables z_i. We define

$$z_\theta \coloneqq \ell + 1 - \lceil \log_k \theta \rceil$$

for $\theta \in \{1, 2, \dots, k^\ell\}$, i. e., $z_1 = \ell + 1$, $z_2 = \cdots = z_k = \ell$, $z_{k+1} = \cdots z_{k^2-1} = \ell - 1$, …, $z_{k^\ell - 1} = 2$, $z_{k^\ell} = 1$.

Now notice that in every network $\mathcal{N}_{\ell,k}$, we can always send k^ℓ units of flow in the first time step using the edge (s_0, t_ℓ). Furthermore, for $r \in \{1, 2, \dots, \ell + 1\}$, at time k^{r-1} we can use all edges (s_i, t_j) with $j - i = \ell + 1 - r$. Each of these r arcs has a capacity of $k^{\ell+1-r}$ and can be used for k^{r-1} time steps to send a total amount of $r \cdot k^\ell$ units of flow. We can actually send a bit more flow, but this is sufficient to see that $p(k^{r-1}) \geq r \cdot k^\ell$ for $r \in \{1, \dots, \ell + 1\}$. Thus, it holds that

$$\sum_{\theta=1}^{k^\ell} z_\theta \cdot p'(\theta) = \sum_{r=1}^{\ell+1} \sum_{\theta=1}^{k^{r-1}} p'(\theta) = \sum_{r=1}^{\ell+1} p(k^{r-1}) \geq \sum_{r=1}^{\ell+1} r \cdot k^\ell = \frac{(\ell+1)(\ell+2)}{2} k^\ell.$$

What remains is to define the y-variables for a solution of (EAC). Note that we have a y-variable for each edge of the time-expanded LP; so we have variables for each copy of an edge in a time layer and also variables for the edges connecting the time layers to the super-terminals $s_i \in S^{+T}$ and $t_i \in S^{-T}$. Figure 4.9 shows a schematic depiction of the time expansion. We begin by defining values for the latter edges, i. e., the edges connecting the super-terminals to the time layers. These edges cut paths through different copies simultaneously. Without loss of generality, we assume that ℓ is odd. We set

$$y_{(s^*, s_r)} \coloneqq \begin{cases} \frac{\ell+1}{2} - r & r \in \left\{0, \dots, \frac{\ell-1}{2}\right\}, \\ 0 & \text{else}, \end{cases}$$

for arcs connecting the sources, and for the arcs connecting the sinks we set

$$y_{(t_r, t^*)} \coloneqq \begin{cases} r - \frac{\ell-1}{2} & r \in \left\{\frac{\ell-1}{2} + 1, \dots, \ell\right\}, \\ 0 & \text{else}. \end{cases}$$

Because each of these arcs has a capacity equal to the demand of k^ℓ, this incurs costs of

$$\sum_{r=0}^{\frac{\ell-1}{2}} \left(\frac{\ell+1}{2} - r\right) k^\ell + \sum_{r=\frac{\ell-1}{2}+1}^{\ell} \left(r - \frac{\ell-1}{2}\right) k^\ell = 2 \sum_{y=1}^{\frac{\ell+1}{2}} y \cdot k^\ell = \frac{(\ell+1)(\ell+3)}{2} k^\ell.$$

By this choice we have cut each path at least once; paths arriving in the first half of the available time are cut on the source side, the other paths are cut on the sink side. This is sufficient for the time layers $k^{\ell-1} + 1$ to k^ℓ. The paths through time layers 1 to $k^{\ell-1}$ have to be cut at least twice, which is already the case for all paths except those going through $\left(s_{\frac{\ell-1}{2}}, t_{\frac{\ell-1}{2}}\right)$ or $\left(s_{\frac{\ell-1}{2}+1}, t_{\frac{\ell-1}{2}+1}\right)$ because the sum of their y values is only 1. We can fix this by setting $y_e = 1$ for arc copies $e = \left(s^\theta_{\frac{\ell-1}{2}}, t^\theta_{\frac{\ell-1}{2}}\right)$ and $e = \left(s^\theta_{\frac{\ell-1}{2}+1}, t^\theta_{\frac{\ell-1}{2}+1}\right)$ for time $\theta \in \{1, \dots, k^{\ell-1}\}$, as all paths that have to be cut again use one of these edges. Each of these edges has a capacity of 1 and there are $2k^{\ell-1}$ of these edges, yielding costs of $2k^{\ell-1}$.

In general, we have to cut the paths in the copies 1 up to $k^{\ell-m}$ at least $m+1$ times. Note that a path through the edge $(s^i_{r_1}, t^i_{r_2})$ for $i \in \{1, \dots, k^{\ell-m}\}$, $r_1 \le r_2$, is already cut at least $r_2 - r_1 + 1$ times by the way we cut the edges connecting the sources and sinks to the super-terminals. Thus, only paths going through an edge $e = \left(s^i_{r_1}, t^i_{r_2}\right)$ with $r_2 - r_1 \le m-1$ are not cut sufficiently often. For these edges $u_e = k^{r_2-r_1} \le k^{m-1}$ holds. Thus, we set $y_e = m+1$ for all those edges, inducing additional costs less than $\ell^3 \cdot k^{m-1} \cdot k^{\ell-m} = \ell^3 \cdot k^{\ell-1}$ because $m \le \ell$ and we have at most ℓ^2 edges within each copy. Now it holds that

$$
\begin{aligned}
& \frac{1}{\beta} \sum_{i=1}^{k^\ell} z_i \cdot p'(i) < \frac{(\ell+1)(\ell+3)}{2} k^\ell + \mathcal{O}(\ell^3 k^{\ell-1}) \\
\Leftrightarrow \quad & \frac{1}{\beta} \frac{(\ell+1)(\ell+2)}{2} k^\ell < \frac{(\ell+1)(\ell+3)}{2} k^\ell + \mathcal{O}(\ell^3 k^{\ell-1}) \\
\Leftrightarrow \quad & \frac{1}{\beta} \frac{(\ell+1)(\ell+2)}{2} < \frac{(\ell+1)(\ell+3)}{2} + \mathcal{O}\left(\frac{\ell^3}{k}\right) \\
\Leftrightarrow \quad & \frac{1}{\beta} < \frac{1}{2} \cdot \frac{\ell+3}{\ell+2} + \mathcal{O}\left(\frac{\ell}{k}\right) = \frac{1}{2} + \frac{\frac{1}{2}}{\ell+2} + \mathcal{O}\left(\frac{\ell}{k}\right)
\end{aligned}
$$

Thus, for every $\frac{1}{\beta} > \frac{1}{2}$, there exist values for k and ℓ such that the network has no β-approximate earliest arrival flow. For $\frac{1}{\beta} = \frac{1}{2} + \varepsilon$, first set ℓ such that $\frac{1/2}{\ell+2} < \frac{\varepsilon}{2}$, and then set k such that $c' \cdot \frac{\ell}{k} < \frac{\varepsilon}{2}$ for the constant c' hidden in $\mathcal{O}\left(\frac{\ell}{k}\right)$. □

Implementation Hints

The algorithm mainly consists of computing iterative maximum flows over time in the increasing time-expanded networks. However, when computing the maximum flow the used algorithm has to be modified, such that the flow in earlier points in time is not decreased. This can be easily achieved by using an augmenting path algorithm, such as the algorithms from Ford and Fulkerson, or Edmonds and Karp. Notice that these algorithms behave as the Successive Shortest Path Algorithm on increasing networks if arcs connecting sinks with super sinks exist in the current time step. This implies a practical improvement over the standard implementation, but does not improve theoretical worst-case running time. However, it can be implemented within a single network flow computation using the parametric push-relabel algorithm by Gallo, Grigoriadis, and Tarjan [GGT89].

Increasing Time-expanded Networks. Residual increasing time-expanded networks can be implemented in a forward star data structure with additional properties. The forward star representation of a network is a commonly used representation that is memory efficient. In contrast, changes in the network structure need linear time. For an introduction see for example Section 2.3 in the text book by Ahuja, Magnanti and Orlin [AMO93]. The network structure remains constant during the run of the value-approximate earliest arrival flow algorithm such that the disadvantage is negligible.

An increasing network can be simulated by ordering arcs and disabling later time layers. We assume that arcs with higher index point to node layers for later points in time. By storing an index

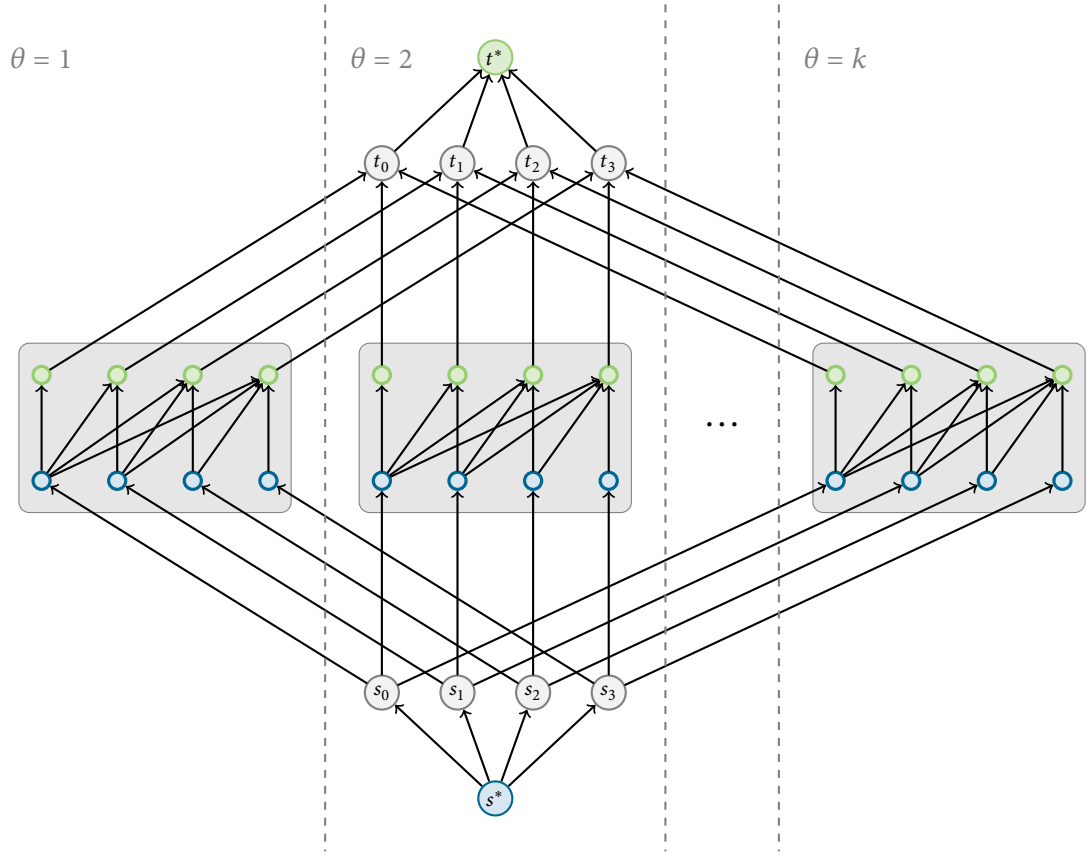


Figure 4.9: The network $\mathcal{N}^{\sim}_{3,k}$ from Figure 4.8c within the time expansion as defined in Definition 2.4 and used in the proof of Theorem 4.28.

for each node denoting the last currently visible outgoing arc, higher levels are ignored by algorithms using the network. Each time layer has to be activated once only. Because this is done successively from layer 1 to T the updates of the network structure can be implemented in amortized linear time. Such an approach can significantly speed up a simple Successive Shortest Path approach in the time expanded network, because the nodes for higher time layers are ignored by the shortest path algorithm. The technique simulates Tjandra's [Tja03] algorithm computing an earliest arrival transshipment but is easier to implement because it does not use functions to represent flow values on arcs for different points in time.

4.3.3 The Framework for Multi-commodity Flows

We now investigate how the results for the classical case for a single commodity from Section 4.3.1 can be transferred to the case of multiple commodities. First, observe that the typical examples for the classical case also work as counter examples for the existence in the multi-commodity case. We also briefly discuss that the same technique used to prove the constant approximation factor in Lemma 4.18 cannot easily extended to the case of multi-commodity flows. However, we see that k and $2k$-value-approximate earliest arrival flows exist.

Non-existence. Multi-commodity flows are an extension of the classical problem and thus the existence of earliest arrival flows is not guaranteed. The fan graph from Example 4.1 is a counter example for an instance with k commodities, each of which has a supply and demand of 1. A counter example for zero travel times is depicted in Figure 4.10. Notice, that these instances are as simple as possible and each commodity sends flow from a single source to a single sink.

Multi-commodity Greedy Algorithm. First observe, that a naive transformation of the Greedy 2-value-approximate Earliest Arrival Flow Algorithm 4.3 to the multi-commodity case does not work. Assume that in step 5 of the algorithm we compute a maximum multi-commodity flow in the residual network which satisfies that flow already sent in earlier time steps does not change, which is the algorithm's behaviour. It therefore fails to send any constant approximation for the instance depicted in Figure 4.11. During the first iteration, it would send a flow unit of commodity 1 from the left to the right side, thus blocking any other flow arriving at time 2 making any result an M-value-approximate flow. However, if the algorithm only sends $\frac{1}{2}$ unit in the first iteration, it achieves a 2-value-approximate flow which is best possible in the instance. Notice, that the M sources and sinks can be combined such that the algorithm also fails in cases where each commodity has a single source-sink pair.

Constant Multi-commodity Approximation. The above discussion shows, that it does not work to simply exchange the maximum flow computation with a multi-commodity maximum flow computation in the time expanded network. However, we can use the approach used in Section 2.4 and compute several single-commodity flows in networks with scaled capacities. By splitting the instance and computing k approximate single-commodity earliest arrival transshipments it is possible to get an approximate multi-commodity earliest arrival transshipment.

Algorithm 4.5: Multi-commodity Value Earliest Arrival Approximation Algorithm

---

Input: $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ with k commodities K and supplies and demands $b_{i,v}$.

Output: A $2k$-value-approximate multi-commodity earliest arrival transshipment.

1. Define scaled capacities $u'_e := \frac{u_e}{k}$ for each arc $e \in E$ and sets of sources and sinks are defined $S_i^+ := \{s \in S^+ \mid b_{i,s} > 0\}$ and $S_i^- := \{t \in S^- \mid b_{i,t} < 0\}$, respectively.
2. Create k scaled dynamic networks $\mathcal{N}_i = (G = (V, E), u', \tau, S_i^+, S_i^-)$.
3. Compute value-approximate earliest arrival transshipments f_i in $\mathcal{N}_i$ for each commodity $i \in K$ using Algorithm 4.3 with the respective supplies and demands b_i.
4. Return the multi-commodity flow $f := (f_1, f_2, \dots, f_k)$.

Theorem 4.29. *Algorithm 4.5 computes a $2k$-value-approximate earliest arrival transshipment on instances for the multi-commodity earliest arrival transshipment with k commodities.*

Proof. The flow computed by the algorithm is feasible. Any k flows of the splitted instances can be combined to another feasible flow because $\sum_{i \in K} f_{i,e} \leq \sum_{i \in K} \frac{u_e}{k} = u_e$ holds at each point in time.

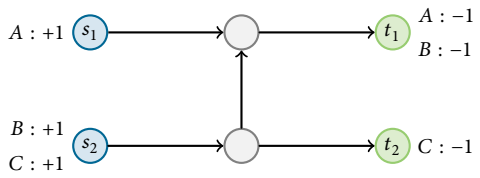


Figure 4.10: A simple instance for the Multi-commodity Earliest Arrival Flow Problem. Each commodity has only one source and one sink and has to ship one unit of flow.

Scaling does not affect flow conservation. Scaling of instances loses a factor of k in the max flow for each point in time thus also losing a factor of k for the earliest arrival transshipment. Using Algorithm 4.3 gives a 2-approximation of the best possible for the scaled instance. The result is a $2k$-value-aproximate earliest arrival transshipment. □

The analysis of Algorithm 4.5 is tight the sense, that there are instances on which the result is only a $2k$ approximation. To see this, consider an instance that consists of k copies of the fan graph, each containing the source and sinks for one commodity. The best approximation factor on this instance is 2 because the approximate flow from Theorem 4.22 can be sent independently in each of the copies. However, the algorithm only returns a $2k$-approximation. However, on instances which only contain a single sink and a single source for each commodity, the analysis can be improved.

Corollary 4.30. *Let $\mathcal{N} = (G = (V, E), u, \tau \geq 0, S^+, S^-)$ be a dynamic network with a set K of k commodities and supplies and demands $b_{i,v}$. If $|\{v \in V \mid b_{i,v} < 0\}| = 1$ for each commodity $i \in K$, then Algorithm 4.5 computes a k-value-approximate earliest arrival flow.*

Proof. Each of the k generated scaled instances $\mathcal{N}_i = (G = (V, E), u', \tau, S_i^+, S_i^-)$ allows for an earliest arrival transshipment due to Theorem 2.18. The combined flow hence loses at most a factor of k due to the scaling. □

Lower Bounds

From the instance in Figure 4.11 we develop a generalized class of instances for which we can show lower bounds for value-approximate earliest arrival transshipments for any $k \in \mathbb{N}$. The class we

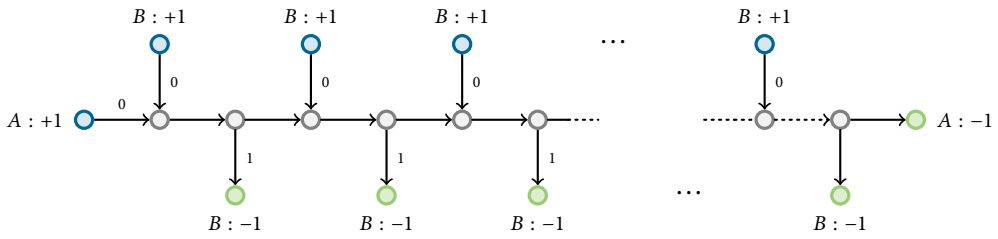


Figure 4.11: A simple multi-commodity instance with 2 commodities. The first has a supply of one at the left node that has to be shipped to the right node. The second commodity has M sources with 1 supply each that have to be shipped to M sinks. The arcs are labelled with their respective transit times.

define uses a set of k commodities such that they all share arcs with low capacity. We first consider a small example and then define the general class of instances.

Example 4.31. *Consider the instance with 2 commodities depicted in Figure 4.12. The instance consists of the graph with node set* $V = \{s_1^1, s_2^1, s_2^1, t_1^1, t_2^1, t_2^2, v_2^1, v_2^2, w_2^1, w_2^1\}$ *and edges* $E = \{(s_1^1, v_2^1), (s_2^1, v_2^1), e_2^1 = (v_2^1, w_2^1), (w_2^1, t_2^1), (w_2^1, v_2^2), (s_2^2, v_2^2), e_2^2 = (v_2^2, w_2^2), (w_2^2, t_2^2), (w_2^2, t_1^1)\}$. *All arcs have capacity of 1, the supplies and demands are* 1 *and* -1 *for the source and sinks, respectively. The arcs leading to the sinks have transit time of 1, all other arcs have zero transit times.*

The maximum flow for time horizon $T = 1$ *is 1, the flow unit is sent along path* $P = (s_1^1, v_2^1, w_2^1, v_2^2, w_2^2, t_1^1)$ *with total transit time 1. For a time horizon of* $T = 2$, *all units can be sent. At the beginning flow from commodity B is sent along paths* $(s_2^1, v_2^1, w_2^1, t_2^1)$ *and* $(s_2^2, v_2^2, w_2^2, t_2^2)$. *At time 1 again the zero travel time path P is used to send the remaining flow of commodity A from* s_1^1 *to* t_1^2.

The best possible approximation for this instance is a $\frac{5}{3}$*-value-approximate earliest arrival flow. Such a flow can be realized by sending* $\frac{3}{5} = 0.6$ *units of flow along P in the first time step and sending* $\frac{2}{5} = 0.4$ *at time 1. The remaining capacity of the arcs* e_2^1 *and* e_2^1 *will be used by sending another* $\frac{2}{5}$ *flow unit from* s_2^1 *to* t_2^1 *and* s_2^2 *to* t_2^2 *each. Optimality can be verified by checking the dual value of the multi-commodity variant of the linear program (*β^**-EAT) that we discuss in the next section.*

Class of Instances. We now define instances G_k^k that do not allow for a $k-1$ approximate earliest arrival flow. All instances G_k^1 are the same for any $k \in \mathbb{N}$ and only consist of a single arc $e_1^1 = (s_1^1, t_1^2)$ connecting the source and sink of the first commodity. We iteratively create graph G_k^{i+1} from graph G_k^i due to the following iterative process.

Consider each arc $e_i^j = (v_{i-1}^j, w_{i-1}^j)$ for $j = 1, \dots, i$. We remove the arc from the graph and introduce new intermediate vertices $v_{i+1}^1, w_{i+1}^1, \dots, v_{i+1}^k, w_{i+1}^k$. We add new edges $e_{i+1}^j = (v_{i+1}^j, w_{i+1}^j)$ for $j = 1, \dots, k$ and also connect w_{i+1}^{j-1} with v_{i-1}^j for $j = 1, \dots, k+1$. k new sources s_{i+1}^j are connected with v_{i+1}^j, and nodes w_{i+1}^j is connected with sink t_{i+1}^j for $j = 1, \dots, k$. Such a part is depicted in Figure 4.13.

The recursive procedure defines graphs G_k^k with k^{j-1} sources/sinks for commodity $j \in \{1, 2, \dots, k\}$ and the maximum flow that can be send within a time horizon of θ is $\sum_{i=1}^{\theta} k^{i-1}$. With this definition we now see that the graph of Example 4.31 is actually G_2^2. Because the flow values that can be sent by the commodities grow exponentially with the time, we can show a lower bound of $k-1$ for graph G_k^k. The idea is the following: Paths of commodity i have a travel time of i. If a given fraction of them should be sent until this time, the flow has to start at time 0 to arrive in time. Therefore, the capacity of the arcs e_i^j becomes the limiting factor for the flow value. Any flow sent from s_i^j to t_i^j for

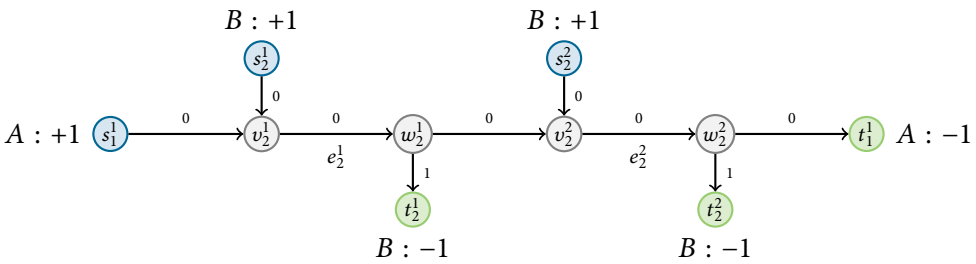


Figure 4.12: The graph G_2 for two commodities. $2 - 1 = 1$ iterations have been done introducing $K = 2$ source and sink pairs.

commodity i at time 1 blocks k times as much flow from commodity $i+1$, k^2 times as much flow from commodity $i+2$ and so on. We formalize this in the following theorem.

Theorem 4.32. *For any $k \in \mathbb{N}$, there exists a network with k commodities that does not allow for a $k-1$-value-approximate earliest arrival multi-commodity flow.*

Proof. Consider a graph G_k^k for the given $k \in \mathbb{N}$. The pattern for G_k^k with k commodities is $p^*(\theta) = \frac{k^\theta - 1}{k-1}$ for $\theta \in \{1, 2, \dots, k\}$. To get a $k-1$-value-approximate flow, $\frac{1}{k-1}$ flow units have to be sent on the s_1-t_1-path for commodity 1 at the first time step $\theta = 1$. Consider now any $\theta \in \{2, 3, \dots, k\}$ and assume that for time $\theta - 1$ a $k-1$-value-approximate flow has been computed. That is, $\frac{1}{k-1}\frac{k^{\theta-1}-1}{k-1}$ flow units are already sent. To get the desired approximation factor, we have to send

$$\frac{1}{k-1}k^{\theta-1} - \frac{1}{k-1}k^\theta - 2 = \frac{(k-1)k^\theta - 2}{k-1} = \frac{k^{\theta-1}}{k-1}$$

flow units in addition. There are $k^{\theta-1}$ paths between source s_θ and sink t_θ available that start sending flow at time 1 and reach the sinks within a time horizon of θ. These paths can be used to send $\frac{k^{\theta-1}}{k^{\theta-1}} = \frac{1}{k-1}$ additional flow on each of the available paths using up the remaining capacity of arcs e_k^j. This only works $k-1$ times until the edges have no capacity left and no paths can be used to send the necessary flow for time horizon $\theta = k$.

Observe that in fact we can send a little more flow up to time θ for $\theta \geq 2$. We can use flows on paths in later time steps to send more of the remaining supplies from the sources. However, it is not possible to send all of the demands one time step later. If we assume it would be possible to send $\frac{k^{\theta-2}-1}{k-1}$ units of flow up to time θ, we still need a fraction of $\frac{1}{k}$ in each step (but the first). This also needs a flow for $\theta = k$ that requires little more capacity than is available. □

Practical Behaviour. For a practical study of the optimal approximation value β^* on instances G_k^k we have to create the graph. Graphs like G_k^k basically consisting of a single path can be easily specified by a sequence of numbers in $\mathbb{Z}$. Each element in the sequence then stands for a node on the path. If entry a_n in the sequence is positive, a source is connected to node v_n, negative entries represent a sink reachable by v_n. We also use $|a_n|$ as transit times for negative sequence entries. The graph G_2^2 depicted in Figure 4.12 can be represented as $(0, 1, -1, 1, -1, 0)$ and the graph G_3^3 from Figure 4.14c is represented by the sequence $(0, 1, 2, -2, 2, -2, 2, -2, -1, 1, 2, -2, 2, -2, 2, -2, -1, 1, 2, -2, 2, -2, 2, -2, -1, 0)$. Using such a sequence that can be easily generated recursively a graph data structure can be built without necessity of deleting edges.

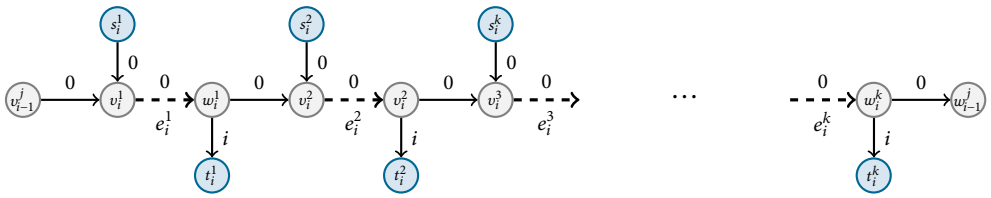


Figure 4.13: The gadget used to generate the graph G_j. It contains K source-sink pairs, each of which has one connecting path with travel time j.

k	2	3	4	5
β'	0.6	0.452 174	0.328 548	0.249 62
β^*	1.666 67	2.211 54	3.0437	4.0061
gap	$\approx 6.6667 \times 10^{-1}$	$\approx 2.1154 \times 10^{-1}$	$\approx 4.3697 \times 10^{-2}$	$\approx 6.0902 \times 10^{-3}$

Table 4.1: Optimal bounds for the graphs G_k^k for $k \in \{2, 3, 4, 5\}$ computed as optimal solution of a path based LP-formulation similar (β^*-EAT).

Denote the best possible approximation factor by β^*. It can be computed using the (pseudo-polynomially large) linear program (β^*-EAT) that we will analyse in detail in the next section. See Table 4.1 for the optimal values $\beta' = \frac{1}{\beta^*}$ for the instances G_2^2, G_3^3, G_4^4 and G_5^5. The optimal value approximate flow for G_2^2 is $\beta^* = \frac{5}{3}$ as we have already mentioned in Example 4.31. The gap converges very fast against 0. There is no $k-1$-value approximate earliest arrival flow, but for any $\varepsilon > 0$ there is a $k \in \mathbb{N}$ such there exists a graph G_k^k that allows for a $k - 1 + \varepsilon$-value-approximate earliest arrival flow. As a last observation we see that the same approximation factor can be reached for the case of a single source and sink for each commodity.

Corollary 4.33. *The lower bound for instances with single source and sink pair for each commodity is* $k - 1$.

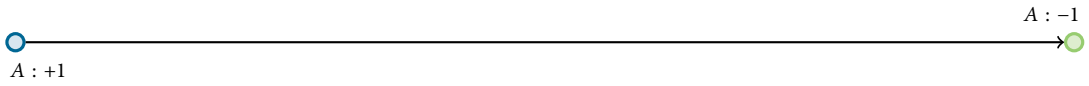
Proof. Consider the modified graph $\tilde{G}_k^k$ which is the result of combining all sources and sinks for the same commodity, i. e., we combine the sources $s_i^1, \dots, s_i^{i^k}$ for commodity i to a single source s_i and the same for the sinks. For each point in time $\theta \in \{1, 2, \dots, k\}$ the flow values $\left|\tilde{f}\right|^\theta = |f|^\theta$ for two flows in $\tilde{f}$ in $\tilde{G}_k^k$ and f in G_k^k. Observe now that the same arguments as in Theorem 4.32 also hold. □

4.4 Instance Optimal Approximation

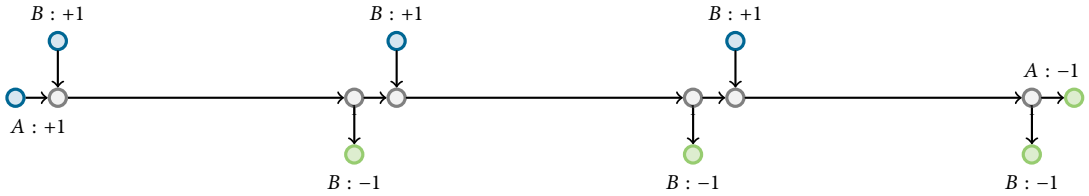
In Section 4.3 we described algorithms for computing value-approximate earliest arrival flows in the single-commodity case and gave instances which allow no better approximation. Thus, our algorithms are best possible in the sense that no better approximation guarantee is possible in general. However, for a given instance, a better value-approximation might be achievable. In the remainder of this chapter we will show existence of such better approximations for instances having a single commodity.

We now focus on approximating the optimal value-approximation factor in the continuous setting under the relaxation that flow may arrive later by a factor $(1+\varepsilon)$. This is equivalent to computing an earliest arrival flow approximation that is $(1+\varepsilon)$-time approximate and $(1+\varepsilon)\beta^*$-value approximate, where β^* is the best possible value approximation factor. We present a pseudo-polynomial exact algorithm in the discrete time model and a fully polynomial time approximation scheme for both time models.

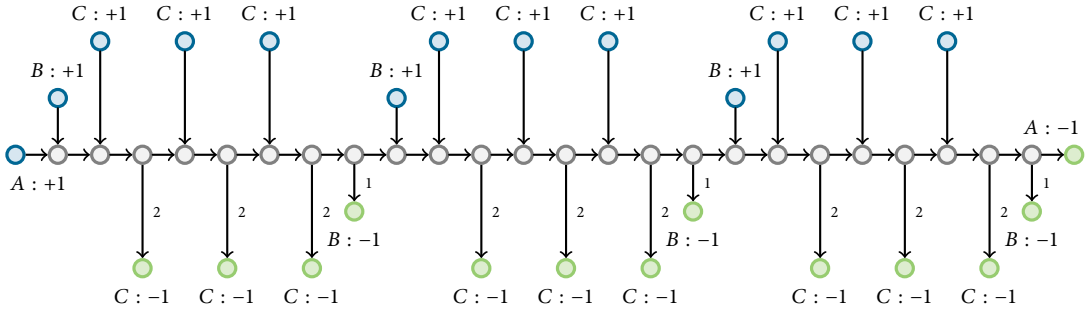
The generic c-Value-approximate Earliest Arrival Flow Algorithm 4.1 together with time-expanded networks for increasing time horizons can be used to decide for an arbitrary β, if a β-value-earliest arrival flow in a given instance exists. Together with a binary search framework the optimal value can be computed (or at least approximated with arbitrary precision) in theory. Such



(a) The initial graph G_3^1 for one commodity.



(b) The graph G_3^2 with two commodities.



(c) The final graph G_3^3 for three commodities.

Figure 4.14: Iterative process to generate graph G_3^3. Between each pair of source and sinks three pairs additional terminals for the next commodity are inserted in each step.

an implementation does achieve an algorithm that can be used in practice. Already one solution of a flow in a time-expanded network may take long time such that it is not feasible to call it within a binary search framework.

In the discrete setting, we can extend the linear programming formulation of the earliest arrival flow problem such that it can be used to approximate the optimum approximation value. We can solve the Earliest Arrival Approximation Feasibility Problem for a given approximation factor by transforming the maximization objective into a constraint (similar to what we have done in the path formulation (EAA)). We therefore add

$$\sum_{t \in S^-} \sum_{e \in \delta_t^-} \sum_{\xi=0}^{\theta-\tau_e} x_{e,\xi} \geq \frac{p^*(\theta)}{\beta} \tag{4.2}$$

as additional (non-linear) constraint. Maximizing β now computes the optimal approximation factor β^*. To formulate the linear program, we use the inverse $\beta' = \frac{1}{\beta}$ of the value-approximation guarantee that we are looking for. Let the dynamic network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ with supplies and demands b be an instance of the earliest arrival flow problem. Let T be a bound for the feasible time horizon, for example as given by Observation 2.8. Assume further, that we are given the earliest arrival pattern p^*. We then can use the following linear program using variables

$x_{e,\theta}$ for the flow on arc copies of $e \in E$ in the time-expanded network to compute the best possible value-approximation.

$$
\begin{aligned}
\max \quad & \beta', && && && (\beta^*\text{-EAT})\\
\text{s.t.} \quad & \sum_{e\in\delta_v^+}\sum_{\xi=0}^{\theta} x_{e,\xi} - \sum_{e\in\delta_v^-}\sum_{\xi=0}^{\theta-\tau_e} x_{e,\xi} && \le \max\{0,b_v\} && \text{for all } v\in V, \theta\in\{1,\dots,T\},\\
& \sum_{e\in\delta_v^+}\sum_{\xi=0}^{\theta} x_{e,\xi} - \sum_{e\in\delta_v^-}\sum_{\xi=0}^{\theta-\tau_e} x_{e,\xi} && \ge \min\{0,b_v\} && \text{for all } v\in V, \theta\in\{1,\dots,T\},\\
& x_{e,\theta} && \le u_e && \text{for all } e\in E, \theta\in\{1,\dots,T\},\\
& \sum_{t\in S^-}\sum_{e\in\delta_t^-}\sum_{\xi=0}^{\theta-\tau_e} x_{e,\xi} && \ge \beta' p^*(\theta) && \text{for all } \theta\in\{1,\dots,T\},\\
& x_{e,\theta} && \ge 0 && \text{for all } e\in E, \theta\in\{1,\dots,T\}.
\end{aligned}
$$

Observe that using the inverse β' makes (β^*-EAT) indeed a linear program. The following lemma states that the LP in fact can be used to compute the optimal approximation factor (in pseudo-polynomial time).

Lemma 4.34. *Let $\mathcal{N} = (G = (V,E), u, \tau, S^+, S^-)$ with with supplies and demands b_v be an instance for the earliest arrival transshipment problem. Then a β^*-value-approximate earliest arrival flow can be computed in pseudo-polynomial time in the discrete model, where β^* is the best possible factor such that a value-approximation exists.*

Proof. We start with building the linear program (β^*-EAT) and computing an optimal solution. By construction of the LP, a solution is a $\frac{1}{\beta'}$-value-approximate earliest arrival flow. Any feasible flow over time f in the discrete time model can be transferred into a solution of the LP by setting $x_{e,\theta} = f(e,\theta)$. Hence we obtain the best possible approximation factor $\beta^* := \frac{1}{\beta'}$ by solving this LP. □

Computing the Optimal Time Approximation. Ideally, we would like to use the same technique to compute the optimal value α^* such that an α^*-time-approximate earlist arrival transshipment exits. Unfortunately, this is not possible. To see this, we try to optimize the time-approximation by replacing constraint (4.2) regarding the value-approximation factor by the corresponding constraint

$$\sum_{t\in S^-}\sum_{e\in\delta_t^-}\sum_{\xi=0}^{\theta-\tau_e} x_{e^\xi} \ge p^*\left(\frac{\theta}{\alpha}\right) \tag{4.3}$$

for the time-approximation factor.

Using (4.3), we can set up a program to compute the optimal approximation factor α^*. However, in contrast to the case of value approximation, the result is not a linear program due to the new inequality (4.3) which is non-linear. Replacing α by $\alpha' = \frac{1}{\alpha}$ does not suffice in this case because

the pattern p^* is not a linear function in general. The constraint system to compute the optimal time-approximate factor is given in the program (α^*-EAT).

$$
\begin{aligned}
\min \quad & \alpha, && && (\alpha^*\text{-EAT})\\
\text{s.t.} \quad & \sum_{e\in\delta_v^+}\sum_{\xi=0}^{\theta} x_{e^\xi} - \sum_{e\in\delta_v^-}\sum_{\xi=0}^{\theta-\tau_e} x_{e^\xi} && \le \max\{0, b_v\} && \text{for all } v\in V, \theta\in\{1,\dots,T\},\\
& \sum_{e\in\delta_v^+}\sum_{\xi=0}^{\theta} x_{e^\xi} - \sum_{e\in\delta_v^-}\sum_{\xi=0}^{\theta-\tau_e} x_{e^\xi} && \ge \min\{0, b_v\} && \text{for all } v\in V, \theta\in\{1,\dots,T\},\\
& x_{e^\theta} && \le u_e && \text{for all } e\in E, \theta\in\{1,\dots,T\},\\
& \sum_{t\in S^-}\sum_{e\in\delta_t^-}\sum_{\xi=0}^{\theta-\tau_e} x_{e^\xi} && \ge p^*\Big(\frac{\theta}{\alpha}\Big) && \text{for all } \theta\in\{1,\dots,T\},\\
& x_{e^\theta} && \ge 0 && \text{for all } e\in E, \theta\in\{1,\dots,T\}.
\end{aligned}
$$

Continuous Model. It is not possible to simply solve the problem in the continuous time model similarly since the flow rate into an edge can change at infinitely many points in time. However, we can discretize time into small intervals, i. e., smaller than 1, in which flow rates are considered to be constant to approximate continuous flows by discrete flows (up to arbitrary precision). For such a discretization it is possible to compute the approximate flow in a version of (β^*-EAT) which is larger by the same factor as the desired precision. We can avoid the blow-up due to discretization by using *geometric time condensation*, a technique introduced by Fleischer and Skutella [FS07] to approximate earliest arrival flows in polynomial time. Time condensation combines several time steps into one layer to shrink the necessary size of the time-expanded network. The flow on an arc between condensed layers is then averaged into a feasible flow on the arc copies in the original network. The discretization cannot be done uniformly for earliest arrival flows because it must be fine graded to approximate the behaviour of an optimal flow in early time steps. However, a rough discretization is enough for later time steps. We show how the technique can be used to compute value-approximative earliest arrival flows.

Definition 4.35 (Condensed Time-expanded Network). Let $L \coloneqq (\theta_1, \theta_1, \dots, \theta_k)$ be a sorted list of points in time with $\theta_1 < \theta_2 < \cdots < \theta_{k-1} < \theta_k = T$.

Let $\mathcal{N} = (G = (V, E), u, \tau \ge 0, S^+, S^-)$ be a network with integral time horizon $T \in \mathbb{Z}_{\ge 0}$, and optionally also supplies and demands for each vertex $v \in V$ and each commodity $i \in K$, denoted by $b_{i,v}$. The **condensed time-expanded network** $\mathcal{N}^L \coloneqq (G = (V^L, E^L), u^L, s^*, t^*)$ is defined as follows. The set of nodes consists of k copies of the original network, a super source s^* and super sink t^* and also the original terminal nodes, that is,

$$V^L \coloneqq \{v^\theta \mid v \in V, \theta \in \{\theta_1, \theta_2, \dots, \theta_k\}\} \cup S^+ \cup S^- \cup \{s^*, t^*\}.$$

A copy of an arc $e = (v, w)$ starting at time θ_q in node v^q may not reach node copy $w^{q+\tau_e}$, because time $\theta_q + \tau_e$ is not contained in the list L. We therefore add *rounded* arcs that connect v^q with node copy $w^{r(q,e)}$ where $r(q, e)$ is the next available time after $\theta_q + \tau_e$ in L.

$$E^T := \left\{ e^q = (v^q, w^{r(q,e)}) \;\middle|\; \begin{array}{l} e = (v,w) \in E, \\ q \in \left\{ q \mid \theta_q \in L, \theta_q + \tau_e \le \theta_k \right\}, \\ r(q,e) = \min\left\{ r \mid \theta_r \in L, \theta_q + \tau_e \le \theta_r \right\} \end{array} \right\}$$
$$\cup \left\{ (s_i^*, s) \;\middle|\; i \in K, s \in S_i^+ \right\} \cup \left\{ (s, s^\theta) \mid i \in K, s \in S_i^+, \theta \in \{1, 2, \dots, T\} \right\}$$
$$\cup \left\{ (t, t_i^*) \;\middle|\; i \in K, t \in S_i^- \right\} \cup \left\{ (t^\theta, t) \mid i \in K, t \in S_i^-, \theta \in \{1, 2, \dots, T\} \right\}$$

If node storage is desired, we additionally add **holdover arcs** between node copies belonging to intermediate nodes of the original network

$$H := \left\{ (v^q, v^{q+1}) \;\middle|\; v \in V \setminus \bigcup_{i \in K} S^+ \uplus S^-, v \in \{1, \dots, k-1\} \right\},$$

and set $E^L := E^L \cup H$ in this case. Arc capacities of copies of arcs are scaled, while other capacities depend on node balances. If no node balances are given, the capacities are defined as

$$u^T_{e'} := \begin{cases} u_e\left(\theta_{q+1} - \theta(q)\right) & \text{if } e' = e^q \\ \infty & \text{else} \end{cases} \quad \text{for all } e' \in E^T.$$

With node balances they are defined as

$$u^T_{e'} := \begin{cases} u_e\left(\theta_{q+1} - \theta(q)\right) & \text{if } e' = e^q, \\ b_{i,s} & e' = (s_i^*, v), \\ -b_{i,t} & e' = (t, t_i^*), \\ \infty & \text{else,} \end{cases} \quad \text{for all } e' \in E^L.$$

◁

We say that L is an **increasing list**, if $\theta_i - \theta_{i-1} \le \theta_{i+1} - \theta_i$ holds for all $i = 2, 3, \dots, k$, e. g., each condensed layer covers at least as much time than the prior ones. We call the corresponding time-expanded network $\mathcal{N}^L$ **increasing condensed time-expanded network**. Observe that standard time-expanded networks as defined in Definition 2.4 are equal to time-expanded networks with the list of time layers $L = (1, 2, \dots, T)$, i. e., in this case $\mathcal{N}^L = \mathcal{N}^T$ holds.

The following two lemmas relate flows over time with flows in condensed time-expanded networks. More precisely, any flow in an increasing time-expanded network $\mathcal{N}^L$ corresponds to a flow over time in the original network. These flows have equal values at the times specified in the list L. For the other direction, there is a slightly weaker version. Here we have that a flow over time in the network in which all transit times are increased by $\theta_k - \theta_{k-1}$ induces a static flow of the same value in the condensed network $\mathcal{N}^L$.

Lemma 4.36 (Lemma 5.2 in [FS07]). *Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network and $L = (\theta_0, \theta_1, \dots, \theta_k)$ be an increasing list of time points. Let f be a static flow in the in the increasing time-expanded network $\mathcal{N}^L$. Then there exists a corresponding flow over time f' in $\mathcal{N}$ such that $|f'|_{\theta_i} = |f|_{\theta_i}$ for all $i = 1, 2, \dots, k$.*

Lemma 4.37 (Lemma 5.3 in [FS07]). *Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network and $L = (\theta_0, \theta_1, \dots, \theta_k)$ be an increasing list of time points. Let $'$ be a flow over time that completes before time θ_k in the network $\mathcal{N}'$ with increased transit times $\theta' := \theta + \Delta$ with $\Delta := T - \theta_k$. Then there is a corresponding static flow over time in the increased time-expanded network $\mathcal{N}^L$ with the same value.*

Geometrically Condensed time-expandeded Networks. For our purposes we use a specific type of condensed time-expanded networks which comprise time layers whose size doubles every few layers. These networks allow to approximate flow changes very detailed in early time steps end gets rougher the nearer time reaches the time horizon. The name is referencing to the geometric series whose terms are increasing by the same factor, similar to the size of the time layer in the condensed network.

Definition 4.38 (Geometrically Condensed Time-expanded Network)**.** Let a dynamic network $\mathcal{N} = (G = (V, E), u, \tau \geq 0, S^+, S^-)$ with non negative travel times and $n := |V|$ nodes be given. Let $p := \left\lfloor \frac{2n}{\varepsilon^2} \right\rfloor$. We then define

$$L := (\ \begin{array}{llll} 1, & 2, & \dots, & p, \\ p+1, & p+2, & \dots, & 2p, \\ 2(p+1), & 2(p+2), & \dots, & 4p, \\ 4(p+1), & 4(p+2), & \dots, & 8p, \\ \vdots & & & \\ 2^{\ell-1}(p+1), & 2^{\ell-1}(p+2), & \dots, & 2^{\ell}p) \end{array}$$

as the geometrically increasing list of time points and the corresponding **geometrically condensed time-expanded network** $\mathcal{N}^L$. ◁

Smoothing. We will briefly describe the *smoothing* technique introduced by Fleischer and Skutella [FS07]. A feasible flow in the time condensed network may not be feasible in the original network due to the rounded transit times. If the time discretization is well chosen, however we can compute a feasible flow with a slightly larger time horizon. For each point in time, the flow value is smoothed by setting it to the average flow of a given flow over time. The averaging introduces two types of errors. It elongates the necessary time horizon and also exceeds the arc capacities. In contrast to the original flow for the smoothed flow we can show that both errors are not too large and can be bounded. The following lemma shows a slightly weaker variation of Proposition 4.7 a) in [FS07] that is tailored such that it can be used in the context of earliest arrival flows with multiple sinks[1].

Lemma 4.39. *Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network and f be a flow without waiting that finishes by time T and $\Delta = \frac{\varepsilon^2 T}{n}$ and $D := |f|$ the value of f.*

Then there is a flow of value $\frac{1}{(1+\varepsilon)}D$ in the network $\mathcal{N}$ in which all transit times are increased by Δ with time horizon $(1+\varepsilon)^2 T$.

Proof. Let $\mathscr{P}$ be the set of all source-sink paths. We can transform the flow f into a flow $\tilde{f}$ that has a larger time horizon $(1+\varepsilon)T$ by scaling it down and sending it longer on the paths. This yields a path flow $\tilde{f}_P$ for $P \in \mathscr{P}$. For a given point in time θ we can calculate the flow value into an arbitrary arc $e = (v, w) \in E$ by summing up flow on paths through e that reach e early enough, i. e.,

$$\tilde{f}(e, \theta) = \sum_{\{P \in \mathscr{P} \mid e \in P\}} \tilde{f}\big(P, \theta - \tau(P_{[\rightarrow, v]})\big).$$

[1] The original result by Fleischer and Skutella shows that smoothing can be applied *without* losing a small factor of the flow value. However, this is not transferable to the case of earliest arrival flows in networks with multiple sinks.

We smooth the flow by averaging flow within a time window with size εT. Thus we get a flow with the same value which needs εT time units longer. For for $P \in \mathscr{P}$ we define the smoothed path flow $\hat{f}_P$ as

$$\hat{f}(P,\theta) \coloneqq \frac{1}{\varepsilon T}\int_{\theta-\varepsilon T}^{\theta} \tilde{f}(P,\xi)\,d\xi$$

for each time in $\theta \in [0,(1+\varepsilon)T+\varepsilon T[$. It is easy to check that $\hat{f}$ obeys capacities of the arcs and the total amount of flow sent on an arbitrary path $P \in \mathscr{P}$ is equal in $\tilde{f}$ and $\hat{f}$, i. e., $\hat{f}$ sends D flow units.

Now, define new rounded transit times $\tau'_e \coloneqq \tau_e + \Delta$ for each arc $e \in E$. Because the arc lengths differ by $\tau'_e - \tau_e = \Delta$ for complete paths $0 \le \tau'(P) - \tau(P) \le n\Delta = \varepsilon^2 T$ holds. More precise,

$$0 \le \tau'(P_{[\to,v]}) - \tau\big(P_{[\to,v]}\big) \le \varepsilon^2 T$$

holds for each path $P \in \mathscr{P}$ and each arc $e = (v,w) \in E$ on P. We can interpret $\hat{f}$ as flow over time in the network $\mathcal{N} = (G=(V,E),u,\tau',S^+,S^-)$. Observe, that $\hat{f}$ is not necessary a feasible flow, because due to the increased transit times flow travelling along an arc at different time steps in the original flow $\tilde{f}$ may now flow on the arc at the same time. We bound the error, i. e., the amount of flow that potentially violates the arc capacity of $e=(v,w) \in E$ at any point in time $\theta \in [0,(1+\varepsilon)T+\varepsilon T+\varepsilon^2 T[$ by

$$\begin{aligned}
\hat{f}(e,\theta) &= \sum_{\{P\in\mathscr{P}\mid e\in P\}} \hat{f}\Big(P,\theta-\tau'\big(P_{[\to,v]}\big)\Big)\\
&= \frac{1}{\varepsilon T}\sum_{\{P\in\mathscr{P}\mid e\in P\}} \int_{\theta-\tau'\left(P_{[\to,v]}\right)-\varepsilon T}^{\theta-\tau'\left(P_{[\to,v]}\right)} \tilde{f}(P,\xi)\,d\xi\\
&\le \frac{1}{\varepsilon T}\sum_{\{P\in\mathscr{P}\mid e\in P\}} \int_{\theta-\tau\left(P_{[\to,v]}\right)-\varepsilon^2 T-\varepsilon T}^{\theta-\tau\left(P_{[\to,v]}\right)} \tilde{f}(P,\xi)\,d\xi\\
&= \frac{1}{\varepsilon T}\int_{\theta-\varepsilon^2 T-\varepsilon T}^{\theta} \sum_{\{P\in\mathscr{P}\mid e\in P\}} \tilde{f}\Big(P,\xi-\tau\big(P_{[\to,e]}\big)\Big)\,d\xi\\
&= \frac{1}{\varepsilon T}\int_{\theta-\varepsilon^2 T-\varepsilon T}^{\theta} \tilde{f}(e,\xi)\,d\xi\\
&\le \frac{\varepsilon^2 T+\varepsilon T}{\varepsilon T}u_e\\
&= (1+\varepsilon)u_e.
\end{aligned}$$

Thus, by dividing flow $\hat{f}$ by $(1+\varepsilon)$ we get a feasible flow that satisfies a $(1+\varepsilon)$-fraction of the demands D in $\mathcal{N} = (G=(V,E),u,\tau',S^+,S^-)$ with a time horizon of $(1+\varepsilon)T+\varepsilon T+\varepsilon^2 T = (1+\varepsilon)^2 T$. □

Value-approximate Earliest Arrival Flows. We can combine the linear program (β^*-EAT) with the technique of time condensation to show how a β^*-value-approximate earliest arrival flow can be approximated, where β^* is the best possible approximation value. Similarly to (β^*-EAT) corresponds to a time-expanded network, we can define an LP that corresponds to a geometrically condensed time-expanded network. Using the smoothing technique and the lemmas that allow us to transfer

flows between the original network and the condensed time-expanded networks, we can show that a flow in the condensed linear program approximates an optimal flow by additionally losing a bit in the time approximation.

Theorem 4.40. *Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ with with supplies and demands b be an instance for the earliest arrival transshipment problem and $\varepsilon > 0$ with $\frac{1}{\varepsilon}$ integral. Then, a $\big(1 + \mathcal{O}(\varepsilon)\big)$-time- and $(1 + \varepsilon)\beta^*$-value-approximate earliest arrival transshipment can be computed in the continuous time model, where β^* is the best value-approximation possible for the given instance. This can be done with a running time polynomial in the input size and ε^{-1}.*

Proof. We use geometrically condensed time-expanded variant of (β^*-EAT) to compute the optimal approximation value. Without loss of generality, we assume a sufficiently fine discretization of time. A discussion, how the discretization can be chosen is contained in the proof of Lemma 5.1 of [FS07]. The condensed time-expanded network $\mathcal{N}^L$ contains layers that are geometrically spread over $[0, T[$, with more layers near 0 and less layers near T. The geometrical spread ensures that the resulting network has the necessary precision for an FPTAS, but is still polynomial in the input size and $\frac{1}{\varepsilon}$. The resulting network has the following form, with $\theta_0, \dots, \theta_r \in L$ denoting the time points, at which the time layers are placed.

$$
\begin{aligned}
\max \quad & \beta', \\
\text{s.t.} \quad & \sum_{e \in \delta_v^+} x_e - \sum_{e \in \delta_v^-} x_e \leq \max\{0, b_v\} && \text{for all } v \in V, \\
& \sum_{e \in \delta_v^+} x_e - \sum_{e \in \delta_v^-} x_e \geq \min\{0, b_v\} && \text{for all } v \in V^L, \\
& x_{e_i} \leq u_e(\theta_{i+1} - \theta_i) && \text{for all } e_i \in E^L, \\
& |x|_\theta \geq \beta' p^*(\theta) && \text{for all } \theta \in L, \\
& x_e \geq 0 && \text{for all } e \in E^L.
\end{aligned}
$$

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

Let $D \leq \sum_{v \in S^+} b_v$ be an amount of demands and T be the minimal time to send a $\frac{1}{\beta^*} d$ fraction of them, where β^* is the minimal value such that a β^*-value-approximate flow exists. We show the following: There is a flow that sends $\frac{1}{(1+\varepsilon)} \frac{1}{\beta^*} d$ flow units in a time window of $\big(1 + \mathcal{O}(\varepsilon)\big)T$.

We consider the sub-network of $\mathcal{N}^L$ that covers $(1 + \varepsilon)^2 T$ time steps. We define a sub-list of these time layers by setting $k' \coloneqq \min\{i \mid \theta_i \geq (1 + \varepsilon)^2 T\}$ and $L' \coloneqq (0 = \theta_0, \theta_1, \dots, \theta_{k'})$. Then, $\mathcal{N}^{L'}$ is the increasing time-expanded network that is obtained from $\mathcal{N}^L$ by removing all nodes v^θ with $\theta \geq k'$ in upper time layers together with their incident arcs.

Let $\Delta \coloneqq \theta_{k'} - \theta_{k'-1}$. By definition of T there is a β^*-value-approximate flow that sends $\frac{D}{\beta}$ units of flow in $\mathcal{N}$ up to time T.

Let $k - 1$ and k be two consecutive points in time. By definition of L we have $\theta_k = 2^j(p + q)$ and $\theta_{k-1} = 2^j(p + q - 1)$ for some $j \in \{0, \dots, \ell - 1\}$ and $q \in \{1, \dots, p\}$. Then the following holds.

$$
\theta_{k'} - \theta_{k'-1} = 2^j(p + q) - 2^j(p + q - 1) = 2^j \leq 2^j + \frac{2^j q}{p} = \frac{2^j(p + q)}{p} = \frac{\theta_{k'}}{p}.
$$

Let $\Delta \coloneqq \theta_{k'} - \theta_{k\_1}$. If we assume without loss of generality that $\theta_{k'} \leq 2T$, by definition of p and the above calculation we can bound Δ by

$$\Delta = \theta_{k'} - \theta_{k-1} \leq \frac{\theta_{k'}}{p} \leq \frac{2T}{\left\lfloor \frac{2n}{\varepsilon^2} \right\rfloor} \leq \frac{\varepsilon^2 T}{n}.$$

There is a flow of value D in $\mathcal{N}$ by assumption. Lemma 4.39 implies that there is a flow in network $\mathcal{N} = (G = (V, E), u, \tau', S^+, S^-)$ in which all transit times are increased by δ. By Lemma 4.37, this implies that there is a flow of value $\frac{1}{(1+\varepsilon)} \frac{1}{\beta^*} D$ in $\mathcal{N}^{L'}$.

Thus, an optimal solution of the condensed linear program at least makes sure that a flow value of $\frac{1}{(1+\varepsilon)} \frac{1}{\beta^*} D$ has been sent until time layer $k' - 1$. The corresponding flow over time in network $\mathcal{N}$ then sends at least $\frac{1}{(1+\varepsilon)} \frac{1}{\beta^*} D$ units of flow by time $\theta_{k'}$ by Lemma 4.36. Thus, this flow is a $(1 + \varepsilon)$-β^*-value-approximate flow which violates time constraints by at most $1 + \mathcal{O}(\varepsilon)$.

□

5 Negative Travel Times

Network flows over time have been studied in many settings, all of which include non-negative travel times. However, there has been no study of flows in scenarios where the input instances already contain arcs with negative travel times. We consider different cases that occur if negative travel times are allowed. We establish a connection to flow problems where arcs are only available after a release date and can be used only until a deadline. For these problems we classify which instances can still be solved in polynomial time. For the case of quickest transshipments we give a simple approximation algorithm that computes temporally repeated paths and exceeds the optimal time horizon at most by a factor of $(2+\varepsilon)$.

As an application we present the Matching over Time Problem that can be solved for bipartite graphs by a reduction to the Maximum Flow over Time Problem. The reduction generates instances with negative travel times and arc release dates. Those instances do not provide a polynomial time algorithm. We complement the result by stating that the Matching over Time Problem is $\mathcal{NP}$-hard.

Publication Remark. A preliminary version of the results in this chapter has been published as extended abstract in [BKM+12].

So far many generalizations of network flow over time problems have been proposed and analysed. The generalizations share the property that they all use integral or sometimes rational transit times on the arcs. In this chapter we study a generalization of network flows over time problems by allowing travel times to be negative. We discuss how negative travel times can be incorporated into flows over time and how they influence the complexity of the Maximum Flow over Time Problem and Quickest Transshipment Problem.

Despite of the general motivation to complete the landscape of flow over time problems there is also a more practical motivation to review negative travel times. Residual graphs for flows over time typically contain arcs with negative transit times (more precise, the backwards arc have the negative travel time of the original arc). The case is usually not considered independently due to the strong connection between arcs and their reverse arcs. However, this is a special variant of a more general task: We are given a network together with a feasible flow over time and the task is to send as much flow as possible between some new sources and sinks. To the best of the author's knowledge, the only work that uses negative travel times is by Tjandra [Tja03]. He studies a bicriteria version of the shortest paths problem with time varying properties and also allows negative travel times on arcs.

If arcs are allowed to have negative travel times, instances may become harder and we will see that even the Maximum Flow over Time Problem becomes $\mathcal{NP}$-hard under these circumstances. However, the actual hardness of instances can vary a lot, depending on the absolute value of the negative travel times. In this chapter we assume that arcs may have negative travel times, however, flow is not allowed to be on arcs before time zero. Problem instances become hard to solve if there are source-sink paths that cannot be used in the complete available time interval, i. e., if flow starting

at time zero would flow into the negative time range. Negative travel times do not hinder solutions in time-expanded networks, but in contrast to the case of non-negative travel times, one cannot derive fully polynomial-time approximation schemes by using time condensation.

Matchings over Time. Recently, a lot of research has been done on how other combinatorial problems can be extended to temporal settings. Works on this topic include the extension of the abstract flow problem to abstract flows over time [KMP14] and the more general framework by Adjiashvili et al. [ABW+14]. With the Matching over Time Problem, we present another temporal version of the traditional matching problem. In this variant, nodes connected by an arc can be matched at some point in time (probably more than once). An arc prescribes a time interval that has to pass between the two points in time at which the adjacent nodes are matched. If we consider this time span as transit time, we can model the situation as a network flow over time in bipartite graphs, where waiting is not allowed.

Matchings over time are, for example, important whenever jobs need to be assigned to pairs of processors with the additional constraint that exactly a given amount of time has to lie between the two processing steps. In this model, each processor can be identified with a node, and the jobs with the edges of G. The delay $\tau_e(v)$ models some setup time needed for processor v to be able to process job e. In particular, whenever job $e = \{v, w\}$ needs to be processed on v for at least τ_e time steps before w can start working on it, this could be modelled by setting $\tau_e(v) = 0$ and $\tau_e(w) = \tau_e$. In this setting a maximum matching over time maximizes the number of processed jobs. The model can easily be extended to hypergraphs, e. g., graphs in which an edge connects more than two nodes. However, we leave this setting as an interesting direction for future research.

Flow Storage in Intermediate Nodes. The possibility of waiting in intermediate nodes and its influence is an important property in the context of flows over time. For the traditional Maximum Flow over Time Problem and the Quickest Transshipment Problem optimal solutions do not need flow to wait in intermediate nodes. The same is true for many more network flow problems. However, this is not the case for the Multi-commodity Flow over Time Problem, where the possibility of waiting induces a different maximum flow value [GS12b]. We have seen similar results for networks with arc release dates and deadlines in Section 2.3. We see a relation between release dates/deadlines and negative travel times: If node storage in intermediate nodes is allowed, the maximum flow value at a given time horizon may be higher in networks with negative travel times. Also, the problem becomes $\mathcal{NP}$-hard if waiting is not permitted.

Ouline of the Chapter. In Section 5.1 we introduce negative travel times, study the influence of waiting and discuss how the hardness of flow over time problems changes. We will see that the Quickest Transshipment Problem with general travel times can be solved in polynomial time if waiting in intermediate nodes is allowed. The problem does not necessarily become hard when negative travel times are introduced, we discuss which classes permit polynomial time solutions in Section 5.2. We see connections to the classical variant of Maximum Flow over Time and prove that the introduction of "a little" negativity does not destroy the structure of the problem. In Section 5.3 we show that the Quickest Transshipment Problem with negative travel times can be approximated if we exceed the time horizon by a factor of 2. We achieve the result by an application of an approximation algorithm by Fleischer and Skutella [FS07], which computes temporally repeated solutions. The Matching over Time Problem is introduced in Chapter 5.4. We use a

reduction of the problem to Maximum Flow over Time that uses both negative travel times and arc release dates to solve it for bipartite graphs.

5.1 Model Definition and Hardness

If negative travel times are allowed on arcs, not only the upper bound T on the available time, but also the lower bound 0 becomes a limitation for flow on arcs. In Definition 2.1 we take care of this by forbidding flow on arcs before time $\theta < |\tau_e|$ for negative τ_e on an arc e. This means that there can be no flow before time 0 on any arc. This fits to the applications where we are either given a network containing some flow already and to the setting of the Matching over Time Problem which we describe later in Section 5.4.

One might also think of an alternative possibility to define flows with negative travel time. Consider the path formulation of network flows and assume that it is still allowed to use any path directly from the beginning. Flow on these paths then may reach into times before time 0, but not arbitrarily (if paths are simple). This model is essentially equal to the traditional model of non-negative travel times and will therefore not further discuss it.

We call a dynamic network whose transit times τ_e may be negative a network with **general transit times**. Recalling from Definition 2.1 a feasible flow in a network with general transit times omits capacities of arcs, satisfies (weak) flow conservation, and respects $f_i(e, \theta) \coloneqq 0$ for all

$$\theta \in]0, 0 - \min\{0, \tau_e\}[\cup]T - \max\{0, \tau_e\}, T[$$

in the continuous setting. In the discrete setting no flow is allowed on arcs for all

$$\theta \in \{n \in \mathbb{N} \mid n < 1 - \min\{0, \tau_e\}\} \cup \{n \in \mathbb{N} \mid n > T - \max\{0, \tau_e\}\}.$$

The Influence of Waiting. The possibility of storing flow units in intermediate nodes is a relaxation of the flow conservation constraints. Any feasible flow that does not wait in intermediate nodes is still a feasible flow, if waiting is allowed. In most typical flow problems waiting is not necessary and optimal solutions without waiting exist. Ford and Fulkersons classical algorithm for the maximum flow over time problem generates temporally repeated optimal solutions that do not need waiting at intermediate nodes [FF58]. For the Quickest Transshipment Problem, Hoppe and Tardos [HT00] have introduced a generalized notion that still allows for a temporally repeated solution without waiting in intermediate nodes. Also for the Earliest Arrival Flow it is possible to compute optimal solutions that do not need waiting in intermediate nodes. The same holds for many additional variants of network flow problems.

In contrast, there are flow over time problems that are affected if flow storage is not allowed. The Multi-commodity Flow over Time Problem exhibits the property that letting flow wait allows to send flow faster in total [GS12b; HHS07]. Only recently, Groß and Skutella [GS15] showed a tight bound of 2 for the speed-up factor of waiting. Similar, by Lemma 2.13 and Theorem 2.15 we know that the maximum amount of flow that can be sent within a given time horizon differs if storage of flow is allowed or not. Furthermore, if waiting is not permitted, computing a maximum flow for a given time horizon becomes $\mathcal{NP}$-hard. We will show that similar results hold for negative travel times.

Example 5.1. *As an example for the implications of forbidding waiting, consider the instance depicted in Figure 5.1. It consists of two arcs* $e_1 = (s, v)$ *and* $e_2 = (v, t)$ *with travel times* $\tau_{e_1} = 0$ *and* $\tau_{e_2} = -1$ *and capacities* $u_{e_1} = 1$ *and* $u_{e_2} = 2$*, and the time is discrete. As a feasible flow over time is not allowed to send flow into any arc before time* 0*, arc* $e = (v, t)$ *with a negative travel time of* -1 *can only be used from time* 1 *on. If the shortest path distance* $\operatorname{dist}(s, v)$ *is smaller and waiting is allowed it is possible to send additional flow earlier to* v *that waits before continuing to the sink. Within a time horizon* $T = 3$ *a maximum flow with value* 2 *can be sent if flow waits at node* v *for one time step. If waiting is not allowed, the path* (s, v, t) *cannot be used at time 0. Thus, the flow value arriving at the sink up to time* T *is 1 at most.*

Cycles. If we require flow units not to wait, automatically flow runs in cycles in optimal solutions. As waiting is not allowed, flow can simulate waiting by travelling a cycle and coming back later. This behaviour is already known from other flow over time problems that allow for better solutions if flow storage in intermediate nodes is forbidden [GS15]. An example of such a scenario is given in Figure 5.2b. Flow starting at time 0 at the source can use the cycle at v to wait one time unit. This is necessary, because arc (v, t) can only be used from time 2 onwards. Thus, waiting allows to send 2 units of flow instead of one unit of flow within a time horizon $T = 2$ in the continuous model. An instance containing a negative cycle is depicted in Figure 5.2a. It is possible to send 2 units of flow within a time horizon of $T = 4$. One unit of flow can be sent by using the non-simple path (s, u, v, w, v, w, t). This path can be used for one time step to send one unit of flow in total. It uses arc (v, w) twice, once at time 3 and once at time 2. A second unit of flow can be sent using (s, u, w, t) also from time 0 on. Both paths cannot be used longer as otherwise they would violate the time horizon bound.

5.1.1 Hardness

There is a strong connection between network flows with arc release dates and deadlines as described in Chapter 2.3.1 and network flows over time with general transit times. We can transform a network with arcs and release dates into a network with general transit times without mortal arcs such that flows in the two networks correspond to each other. Basically it is enough to introduce additional preceding arcs for an arc having positive arc release date or deadline smaller than the time horizon.

Connection to Release Dates and Deadlines. By adding arcs with negative travel times we can enforce availability of some paths only after or before a given point in time. Consider a pair of edges e_1e_2 with $\tau_{e_1} + \tau_{e_2} = 0$. Let P be a path in the network containing e_1e_2 as consecutive edges and let e_2 be the only arc with negative travel time. Then, e_2 can only be used after time $|\tau_{e_2}|$ which models a scenario with an arc release date. The maximum flow value $\sum_{e \in P} \tau_e - |\tau_{e_2}|$ that can be sent over P is reduced by the absolute value of the travel time of e_2.

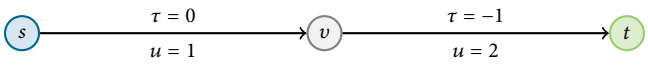
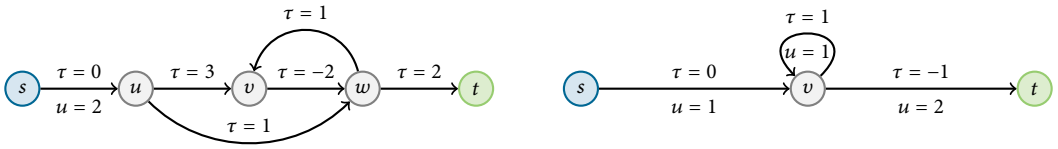


Figure 5.1: A network allowing for different maximum flow value if waiting in intermediate nodes is allowed.



(a) A negative cycle.

(b) A negative cycle.

Figure 5.2: Influence of cycles.

Lemma 5.2. *Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network with additional arc release dates r_e and deadlines d_e.*

There is a dynamic network $\mathcal{N}'$ with general travel times without release dates and deadlines such that any feasible flow f in $\mathcal{N}$ corresponds to a feasible flow f' in $\mathcal{N}'$, and vice versa.

Proof. For each arc $e = (v, w)$ we introduce three additional nodes v_1, v_2 and v_3 in $\mathcal{N}'$ and replace e by the sequence $e_1 = (v, v_1)e_2 = (v_1, v_2)e_3 = (v_2, v_3)e_4 = (v_3, w)$ with the same capacity u_e. We set the travel times of the new arcs to $\tau_{e_1} := -r_e$, $\tau_{e_2} := T - d_e + r_e$, $\tau_{e_3} := -T + d_e$, and $\tau_{e_4} := \tau_e$. An example of the construction is depicted in Figure 5.3a.

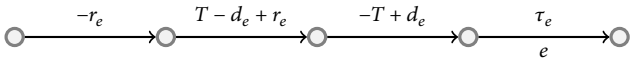
The total transit times of the arc sequence is $-r_e + T - d_e + r_e - T + d_e + \tau_e = \tau_e$. Thus we can exchange flow between feasible flows f in $\mathcal{N}$ and f' in $\mathcal{N}'$, if the arc sequence and e are used at the same time. This is the case, as because of $\tau_{e_1} = -r_e$, no flow can enter the arc sequence before the release time r_e. Flow entering after the deadline d_e would leave arc e_2 after the time horizon T.

Thus, for a pair of corresponding flows f and f' in $\mathcal{N}$ and $\mathcal{N}'$, respectively, we have $f(e, \theta) = f'(e_4, \theta)$. □

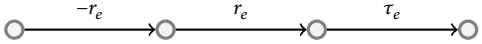
As a consequence, we see that flow problems with negative travel times becomes $\mathcal{NP}$-hard if waiting in intermediate nodes is forbidden.

Corollary 5.3. *The Maximum Flow over Time and the Quickest Transshipment Problem become $\mathcal{NP}$-hard if negative travel times are allowed and flow cannot be stored in intermediate nodes.*

Proof. We can reduce decision variants of both problems to their variants with arc release dates and deadlines which are $\mathcal{NP}$-hard by Theorem 2.15. For the reduction we convert instances with deadlines and release dates according to Lemma 5.2. □



(a) Transit times are depicted on top of the arcs. All arcs have the same capacity u_e.



(b) The reduction can be simplified if no deadlines are specified.

Figure 5.3: Reduction from instances with release dates and deadlines to negative travel times.

So, hardness follows directly from the hardness of the Maximum Flow over Time Problem with mortal arcs. Moreover, by a simple direct reduction to the Partition Problem we can also see that the Maximum Flow over Time Problem is inapproximable if the time horizon cannot be relaxed.

Lemma 5.4. *Assuming $\mathcal{P} \neq \mathcal{NP}$, the value of a maximum flow over time with general travel times without waiting in intermediate nodes is not approximable for any $c > 0$ within given time horizon T, if waiting is not allowed.*

Proof. We reduce Partition to the Maximum Flow over Time with negative travel times. Let $a_1, \dots, a_n$ be a partition instance and $A = \frac{1}{2}\sum_{i=1}^{n} a_i$. We create a graph as depicted in Figure 5.4. Arcs e_i have transit time a_i and arcs $\overline{a_i}$ have zero travel time for $i = 1, 2, \dots, n$.

For time horizon $T := A + 1$ each path can only be used at most once because the first two arcs need the maximal possible time. The last arc can only be used if the travel time of the beginning of the path sums up to A, e. g., the decision of using edges e_i or $\overline{e_i}$ corresponds to a solution for the Partition Problem. Thus, within time T it is possible to send exactly one unit of flow if and only if the Partition instance is a `yes`-instance.

The inapproximability now follows from the gap producing technique. Assume the existence of a c-approximation algorithm. For any `yes`-instance, the solution value is greater than $\frac{1}{c}$, while any `no`-instance has an optimal value of 0, allowing us to decide Partition. □

The reduction tells us, that we have at least to relax feasibility if we want to approximate the maximum flow over time problem with general transit times.

5.1.2 Solving the Problem with Waiting

If waiting is allowed, we can reduce the Maximum Flow over Time Problem with general transit times to the Quickest Transshipment Problem. This follows the same ideas as the reduction of the variant with arc release dates and deadlines we have seen in Section 2.3. We replace each arc e with a negative travel time $-\tau_e$ by the construction depicted in Figure 5.5. Therefore, we get an additional pair of source and sink and revert the direction of the arc. In the following two lemmas we show that we can compute flows with equal flow values at existing terminal nodes in the new network, and also we can transform any feasible transshipment satisfying the demands of the new terminals back into a feasible flow in the original network.

The following intuition stands behind introducing the additional pair of terminals: It is possible to send the supply $u_e \cdot (T - \tau_e)$ from the new source to the sink using the arc exactly in the given time horizon T. The value equals the maximal possible amount of flow that can travel along e. If flow would use the arc with negative travel time at some time θ, this flow can be redirected to the new

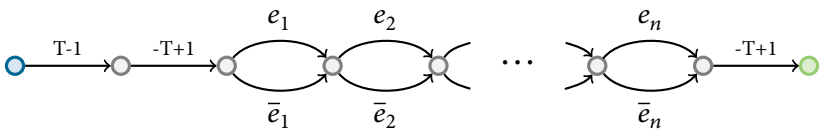


Figure 5.4: Graph used to reduce Maximum Flow over Time with general travel times to Partition.

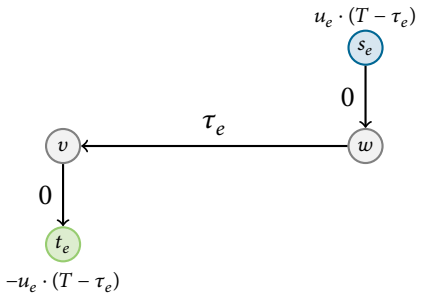


Figure 5.5: The construction that replaces arc e in the reduction to Quickest Transshipment.

sink thus freeing the flow units at the source at time $\theta - \tau_e$, hence simulating the arc with negative travel times.

Lemma 5.5. *Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network, $e = (v, w)$ an arc in $\mathcal{N}$ with negative travel time, f a flow in $\mathcal{N}$ that satisfies the balances b within the time horizon T. Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network, $e = (v, w)$ an arc in $\mathcal{N}$ with negative travel time, f a flow in $\mathcal{N}$ that satisfies the balances b within the time horizon T. Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network, $e = (v, w)$ an arc in $\mathcal{N}$ with negative travel time, f a flow in $\mathcal{N}$ that satisfies the balances b within the time horizon T.*

Let $\mathcal{N}'$ be a network in which $e = (v, w)$ is replaced by the path $((s_e, w), (w, v), (v, t_e))$ with two new terminal nodes s_e and t_e having supplies and demands $u_e \cdot (T - \tau_e)$ and $-u_e \cdot (T - \tau_e)$, respectively.

Then, there exists a flow f' in $\mathcal{N}'$ that satisfies the same balances and additionally satisfies the balances at the new terminals s_e and t_e. If the original flow did not use flow storage, f' does not use flow storage.

Proof. We have to send flow from the new source and to the new sink whenever possible and define f' as follows.

$$f'((s_w, w), \theta) := \begin{cases} u_e & \text{for } \theta \in [0, T - \tau_e[\\ 0 & \text{for } \theta \in [T - \tau_e, T[\end{cases}$$

$$f'((v, t_v), \theta) := \begin{cases} u_e & \text{for } \theta \in [\tau_e, T[\\ 0 & \text{for } \theta \in [0, \tau_e[\end{cases}$$

$$f'((w, v), \theta) := \begin{cases} u_e - f(e, \theta + \tau_e) & \text{for } \theta \in [0, T - \tau_e[\\ 0 & \text{for } \theta \in [T - \tau_e, T[\end{cases}$$

The definition of f'((w,v),⊠) is due to the fact that flow in f entering e at time θ can be redirected to t_v and the flow arriving at w at time $\theta - \tau_e$ can be taken from s_w. For the other arcs $a \in E \setminus \{e\}$ we have equal flow value $f'(a, \theta) := f(a, \theta)$ for $\theta \in [0, T[$.

As f' equals f at all nodes but v and w, we only have to check validity of the flow at these nodes. The arc capacities are respected by definition. For flow conservation at v it is enough to check that the balance of the summands for the three edges whose flow values are changed (or new) are zero.

At time θ we have the following accumulated change in flow conservation:

$$\begin{aligned}
&\left(\int_0^{\theta-\tau_e} f'((w,v),\xi)\,d\xi\right) - \left(-\int_0^{\theta} f(e,\xi)\,d\xi + \int_{\tau_e}^{\theta} u_e\,d\xi\right)\\
&= \int_0^{\theta-\tau_e} u_e - f(e,\xi+\tau_e)\,d\xi + \int_{\tau_e}^{\theta} f(e,\xi)\,d\xi - \int_{\tau_e}^{\theta} u_e\,d\xi\\
&= \int_0^{\theta-\tau_e} u_e - f(e,\xi+\tau_e)\,d\xi + \int_0^{\theta-\tau_e} f(e,\xi+\tau_e)\,d\xi - \int_0^{\theta-\tau_e} u_e\,d\xi\\
&= 0.
\end{aligned}$$

The same calculation can be done for w. Here, we have to take care that we do not integrate after time $T-\tau_e$:

$$\begin{aligned}
&\left(\int_0^{\theta+\tau_e} u_e - \int_{\tau_e}^{\theta} f(e,\xi)\,d\xi\right) - \left(\int_0^{\theta} f'((w,v),\xi)\,d\xi\right)\\
&= \int_0^{\theta} u_e\,d\xi - \int_{\tau_e}^{\theta+\tau_e} f(e,\xi)\,d\xi - \int_0^{\theta} u_e - f(e,\xi+\tau_e)\,d\xi\\
&= \int_{\tau_e}^{\theta+\tau_e} u_e\,d\xi - \int_{\tau_e}^{\theta+\tau_e} f(e,\xi)\,d\xi - \int_{\tau_e}^{\theta+\tau_e} u_e - f(e,\xi)\,d\xi\\
&= 0.
\end{aligned}$$

There is no flow on each of the new arcs after time $T-\tau_e$, and thus the extension of the integrals to T is feasible and the result remains 0. □

In contrast to the result proven in Lemma 5.5, it is not possible to create the equivalent flow in the original network from a feasible flow in the modified network. The reason is the following: It may be that flow arrives before time $|\tau_e|$ at v, which already then is redirected to t_e. Only if waiting is allowed it is possible to modify the flow in a way such that the corresponding flow in the original network uses e with the negative transit time at a later point in time.

Lemma 5.6. *Let $\mathcal{N}'$ be a dynamic network in which one arc e is replaced by the construction from Figure 5.5. Then it is possible to send flow satisfying the same balances in a network $\mathcal{N}$ in which e with negative travel time $-\tau_e$ is present if waiting in intermediate nodes is allowed.*

Proof. In a feasible transshipment in $\mathcal{N}'$, flow may arrive at node v before time τ_e which is then sent forward to the sink t_v. However, it is clear that the total amount of flow that arrives at v and travels to t_v is limited by $u_e \cdot (T-\tau_e)$ and that the capacity of e is sufficient to send this flow.

First we create a new transshipment f^* with changed flow values to satisfy certain requirements. We set $f^*((v,t_v),\theta) \coloneqq 0$ for $\theta \in [0,\tau_e[$ and $f^*((v,t_v),\theta) \coloneqq u_e$ for $\theta \in [\tau_e, T[$. This is always possible, as the amount of flow has to be sent anyway and flow units maybe have to wait. In the same manner we define $f^*((s_w,w),\theta) \coloneqq u_e$ for $\theta \in [0, T-\tau_e[$ and 0 otherwise. We also send the flow on arcs $e \in \delta^+(w)$ as early as possible. The flow f^* remains feasible if waiting is allowed.

Based on the new flow f^* we now define $f(e,\theta) \coloneqq u_e - f^*((w,v),\theta-\tau_e)$ for $\theta \geq \tau_e$ and all other flow values on arcs as in f^*. Now we have a flow of the type as we created in Lemma 5.5 which sends flow from s_w to t_v as early as possible, the same equations hold and the total flow balance at v and w does not change. □

With these two lemmas it is possible to show the following theorem:

Theorem 5.7. *Let $\mathcal{N} = (G = (V, E), u, \tau, s, t)$ be a dynamic network. The Maximum Flow over Time Problem with negative travel times in $\mathcal{N}$ can be solved in (strongly) polynomial time, if waiting is allowed.*

Proof. We create a network $\mathcal{N}'$ by replacing each arc e with negative transit time $-\tau_e$ by the path $((s_e, w), (w, v), (v, t_e))$ with two new terminal nodes s_e and t_e having supplies and demands $u_e \cdot (T - \tau_e)$ and $-u_e \cdot (T - \tau_e)$, respectively.

For given supply $b_s := b$ of the original source and a demand $b_t := -b$ of the original source we can compute a quickest transshipment in $\mathcal{N}'$. To find the maximum value b such that a transshipment exists we use binary search. The algorithm becomes strongly polynomial, if we use Meggido's parametric search framework from Theorem 1.1 instead.

Once the maximum b is found, the value is returned as maximum flow. If the actual flow is desired, we create a flow over time according to the proof of Lemma 5.6. □

5.2 Polynomial Special Cases

The hardness reductions in Lemma 5.4 and Figure 5.4 and the construction of feasible flows respecting negative travel times from extended graphs as described in Lemma 5.6 all show the same problem if waiting is not allowed:[1] It may be possible that flow takes a path too early and thus has to wait before continuing on the path. Clearly, this problem cannot occur on instances with non-negative transit times.

With this intuition we can specify the amount of "negativity" that is still allowed for some arcs, such that instances can still be solved in polynomial time. Following from our considerations, it shall not be possible to reach arcs before the time they are usable, e. g., for an arc e with negative travel time $-\tau_e$ a shortest path to tail(e) has at least a travel time of $|\tau_e|$. Taking a path based view on flows gives us a second restriction. Consider an s-t-path in a classical dynamic network. The path can be used immediately from time 0 on until time $T - \sum_{e \in P} \tau_e$. In the classical network scenario, any maximum network flow within time horizon T can also be solved by computing a flow in the network based on the reverse graph $\overleftarrow{G}$. To maintain this property, the constraint on shortest paths to a node v must also hold in $\overleftarrow{G}$. Paths satisfying this property can then be used to the maximum time $T - \sum_{e \in P} \tau_e$. This motivates the following definition.

Definition 5.8. Let $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ be a dynamic network with general transit times τ_e. We say that the network has **almost non negative transit times**, if for all $P \in \mathcal{P}$ and $v \in P$ the following two conditions hold

$$\tau(P_{[s,v]}) \geq 0 \tag{N1}$$

$$\tau(\overleftarrow{P}_{[v,t]}) \geq 0. \tag{N2}$$

◁

Property (N1) requires all s-v-Paths in the network to have a positive transit time, i. e., each path can be used directly from the beginning on and flow does not have to wait at intermediate nodes. (N2) does exactly the same but for opposite directions, hence a dynamic network satisfies (N2) if

[1] This is also the same problem for instances with mortal edges as described in Lemma 2.13.

and only if the network based on the reverse graph $\overleftarrow{G}$ satisfies (N1). Figure 5.6 shows a network satisfying (N1), but not (N2) because path (t, w, v, s) in the reversed graph can only be used from time 1 on.

We now define the (non-negative) reward that specifies how long a path in a network with negative travel times can be used. For positive transit times, for a path P the reward equals $T - \tau(P)$. This is the same value that is used in the minimum cost flow approach in Algorithm 2.1. For a path P we define $\tau^-(P) := -\min_{v \in P}\{\tau(P_{[s,v]})\} \geq 0$ as the first point in time that P can be used to send flow without violating the constraints. Similarly we define $\tau^+(P) := \max_{v \in P}\{\tau(P_{[s,v]})\} \geq 0$. The value denotes the time interval in which we cannot use the path before the upper bound T. The **reward** of a path is then defined as $r_T(P) := \max\{0, T - \tau^-(P) - \tau^+(P)\}$. Notice that the maximum reward can be gained if the transit times are almost non-negative.

Observation 5.9. *In a dynamic network whose transit times are almost non-negative the reward satisfies $r(P) = T - \tau(P)$.*

Proof. Because of (N1), all s-v-sub-paths have non-negative sum of travel time and therefore $\tau^-(P) = 0$. If (N2) holds, the maximum is achieved for the complete path. Hence, we have $\tau^+(P) = \tau(P)$ in this case. □

The observation indicates that in such (relatively constrained) networks optimal solutions that can be expressed as temporally repeated flows do still exist. Two networks that do not allow for temporally repeated solutions are depicted in Figure 5.7. An example that does neither satisfy (N1) nor (N2) is depicted in Figure 5.7a, the example in Figure 5.7b satisfies (N1) but not (N2). The paths using arcs with transit times different from 0 cannot be used in the complete time interval and thus optimal solutions enforce that the zero travel time path has to be used in different time intervals. For the instance in Figure 5.7b the zero travel time path through the middle is used at the beginning up to time $\frac{T}{4}$ and again after time $\frac{3}{4}T$ in an optimal solution. Notice that the instance in Figure 5.7a allows a solution in which all paths are only used in a single time interval. However, the paths are not used with their maximum reward. In the following we will prove our intuition that exactly in networks with almost non-negative transit times a maximum flow over time (without waiting in intermediate nodes) can be computed in polynomial time.

Reduced Costs Transit Time Conversion. An implementation of the Successive Shortest Paths Algorithm can benefit from so-called *reduced costs*. If all costs on arcs are replaced with reduced costs, they become non-negative. Thus, more efficient algorithms such as Dijkstra's algorithm can be used to compute the shortest paths. We will use the same approach to transform the transit times in our networks. We will see that a flow in the network with reduced costs transit times corresponds to a flow in the original network. Additionally, the transformation has the advantage that the shortest paths have a total length of 0. Therefore the technique can be used to reduce the size of time-expanded networks.

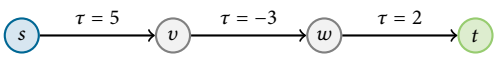
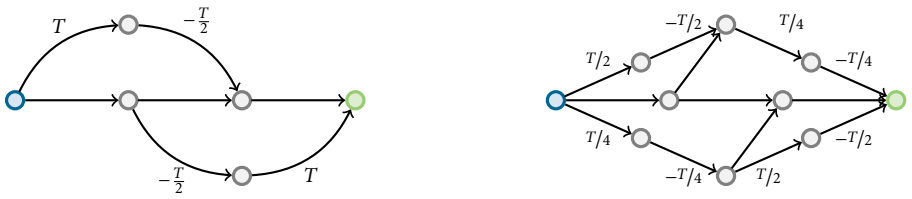


Figure 5.6: A network satisfying property (N1), but not property (N2).



(a) In an optimal solution the zero transit time path is used for one time step.

(b) The zero transit time path is activated twice in an optimal solution.

Figure 5.7: Instances without a temporally repeated solution. Arcs are assumed to have unit capacity, transit times are as denoted near the arcs. All other arcs have zero transit time.

Algorithm 5.1: Reduced Costs Transit Time Conversion

Input: $\mathcal{N} = (G = (V, E), u, \tau, s, t)$ with almost non-negative transit times.

Output: New travel times $\tau' \geq 0$.

1. Compute shortest paths distances $\text{dist}(s, v)$ from the source s to all nodes $v \in V$.
2. Compute transit times $\tau'_e := \tau_e + \text{dist}(s, v) - \text{dist}(s, w)$ for each arc $e = (v, w)$.
3. Return τ'.

The dynamic network $\mathcal{N}' = (G = (V, E), u, \tau', s, t)$ with new reduced costs transit times τ' as computed by the algorithm satisfies the following property.

Lemma 5.10. *Let $\pi_v := \text{dist}(s, v)$ be a feasible potential and $\tau'_e := \tau_e + \pi_v - \pi_w$ for each arc $e \in E$ be new travel times. The distance $\tau((s, \dots, v))$ for each s-v-path in $\mathcal{N}'$ equals $\tau((s, \dots, v)) - \pi_v$ for each $v \in V$.*

Proof. Let P be an s-v-path in $\mathcal{N}'$. Then for the total transit time of a path

$$\tau(P) = \sum_{e \in P} \tau'_e = \sum_{e \in P} \left(\tau_e + \pi_{\text{head}(e)} - \pi_{\text{tail}(e)}\right) = \left(\sum_{e \in P} \tau_e\right) + \pi_s - \pi_v$$

holds. With $\pi_s = 0$ the lemma follows. □

As a consequence, a temporally repeated flow in $\mathcal{N}'$ with a time horizon reduced by π_t has an equal value as the same flow if sent temporally repeated in the original network with the larger time horizon. Notice, that the flow not necessarily is feasible due to the transformed transit times.

Observation 5.11. *Let $\mathcal{N} = (G = (V, E), u, \tau, s, t)$ be a dynamic network with almost non-negative transit times and $\mathcal{N}'$ the network with reduced costs transit times τ' and $\mathcal{P}$ a set of paths.*

Then the value of a flow resulting from temporally repetition of the paths in $\mathcal{P}$ in $\mathcal{N}$ with time horizon T equals the value of the flow resulting from temporally repetition of the paths in $\mathcal{P}$ in $\mathcal{N}'$ with time horizon $T - \pi_t$.

Proof. The transit times are almost non-negative. Together with Lemma 5.10 the value of the temporally repeated flow is then

$$\sum_{P\in\mathscr{P}'}(T-\pi_t)-\tau'(P)=\sum_{P\in\mathscr{P}'}(T-\pi_t)-(\tau(P)-\pi_t)=\sum_{P\in\mathscr{P}'}T-\tau(P).$$

□

Thus, the value of a maximum flow in the original network can be computed by applying Algorithm 2.1 by Ford and Fulkerson to generate a temporally repeated flow in the new network if this is feasible. The following lemma guarantees that any (edge) flow over time can be transferred between the two networks.

Lemma 5.12. *Let $\mathcal{N}=(G=(V,E),u,\tau,s,t)$ be a dynamic network with almost non-negative transit times and let f' be an edge flow in the network $\mathcal{N}'$ with transit times τ' computed as reduced costs by Algorithm 5.1. Let π_v be a shortest path potential for nodes $v\in V$. Define f as*

$$f((v,w),\theta):=\begin{cases}0 & \theta\in[0,\pi_v[,\\ f'((v,w),\theta-\pi_v) & \theta\in[\pi_v,T[.\end{cases}$$

Then, f is a feasible flow over time in $\mathcal{N}$.

Proof. It remains to show that f is a feasible flow. Obviously capacities are respected because f' is a feasible flow and there is no flow left on arcs after time T. We have to show that flow conservation holds. Let v^* be a node and π_{v^*} its potential. For $\theta\geq\pi_{v^*}$ the following holds.

$$\begin{aligned}
&\sum_{e=(v,w)\in\delta^-(v^*)}f(e,\theta-\tau_e)-\sum_{e\in\delta^+(v^*)}f(e,\theta)\\
=&\sum_{e=(v,w)\in\delta^-(v^*)}f'(e,\theta-\tau_e-\pi_v)-\sum_{e\in\delta^+(v^*)}f'(e,\theta-\pi_{v^*})\\
=&\sum_{e=(v,w)\in\delta^-(v^*)}f'(e,\theta-\tau_e-\pi_v-\pi_{v^*}+\pi_w)-\sum_{e\in\delta^+(v^*)}f'(e,\theta-\pi_{v^*})\\
=&\sum_{e=(v,w)\in\delta^-(v^*)}f'\big(e,\theta-\pi_{v^*}-(\tau_e+\pi_v-\pi_w)\big)-\sum_{e\in\delta^+(v^*)}f'(e,\theta-\pi_{v^*})\\
=&\sum_{e=(v,w)\in\delta^-(v^*)}f'(e,\theta-\pi_{v^*}-\tau'_e)-\sum_{e\in\delta^+(v^*)}f'(e,\theta-\pi_{v^*})
\end{aligned}$$

The last equality follows from the flow conservation at time $\theta-\pi_{v^*}$ at node v^*. For earlier times there is no flow reaching v^* and also not leaving. Flow conservation holds for f, because f' is a feasible flow by the precondition. Hence, f is a feasible flow over time. □

Putting the pieces together, we can compute a maximum flow over time in networks with almost non negative transit times.

Theorem 5.13. *The values of a maximum flow over time in $\mathcal{N}$ with time horizon T and $\mathcal{N}'$ with reduced costs transit times and time horizon $T-\pi_t$ are equal if the original transit times τ are almost non-negative travel times. A maximum network flow can be computed in strongly polynomial time.*

Proof. By Lemma 5.12 feasible flows between $\mathcal{N}$ and $\mathcal{N}'$ correspond to each other. We therefore can compute a maximum flow over time in $\mathcal{N}'$. Together with Observation 5.11 it follows that a temporally repeated flow over time in $\mathcal{N}'$ is also feasible for $\mathcal{N}$.

The computation of the reduced transit times by Algorithm 5.1 can be done using the Moore-Bellman-Ford-Algorithm in strongly polynomial running time in $\mathcal{O}(|V| \cdot |E|)$. The temporally repeated flow can be computed by a single minimum cost flow computation in strongly polynomial time in $\mathcal{O}(|E| \log |E|(|E| + |V| \log |V|))$. □

It is necessary to have almost non-negative transit times on the arcs to apply the above technique, i. e., the properties (N1) and (N2) both have to be satisfied. Otherwise it is still possible to compute the reduced costs to get non-negative transit times, however feasible flows can not be transferred between networks with the two types of transit times.

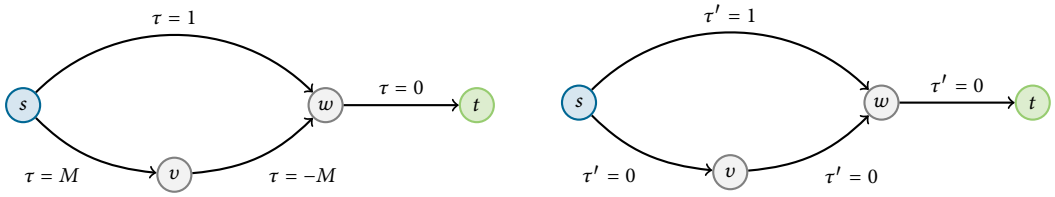
Example 5.14. *As an example, we consider the network depicted in Figure 5.8. The network consists of the nodes $V = \{s, v, w, t\}$ and the arcs $E = \{e_1 = (s, w), e_2 = (s, v), e_3 = (v, w), e_4 = (w, t)\}$ with transit times $\tau_{e_1} = 1, \tau_{e_2} = M, \tau_{e_3} = -M$ and $\tau_{e_4} = 0$ for some integral $M < T$. The actual value of a maximal flow depends of the capacities on the arcs. The optimal solution based on a minimum cost flow always uses the path (e_2, e_3, e_4) independently of the capacities. If the capacity of e_4 is not 1, the flow is not feasible in the original network.*

The same happens in the class of instances we used in the $\mathcal{NP}$-hardness proof in Lemma 5.4 that is depicted in Figure 5.4. Consider two instances for the Partition Problem consisting of the elements $\{1, 2, 3\}$ and $\{2, 2, 2\}$ which are feasible and infeasible, respectively. After the transformation of transit times is applied on the graphs resulting from the reduction the two instances cannot be distinguished.

Reducing the Time Horizon. Applying the reduced costs transit time transformation also has practical implications on instances with non-negative transit times. The reduction can be applied to reduce the necessary time horizon, because the reduction decreases the travel time of the shortest path to 0. Also, the reduction remains feasible if an approximation scheme that increases the necessary transit time by $(1 + \varepsilon)$ is applied.

Lemma 5.15. *Let $\mathcal{N} = (G = (V, E), u, \tau, s, t)$ be a dynamic network with almost non-negative transit times. Let A be a $(1+\varepsilon)$-approximation algorithm of the time horizon, i. e., A solves the problem with non negative travel times for a time horizon of $(1+\varepsilon)T$ instead of T for the Quickest Flow/Maximum Flow over Time/Maximum Earliest Arrival Flow Problem.*

Then solving the problem using algorithm A and transforming back the solution creates a solution



(a) The shortest path *svwt* can be used in the first $T - M$ time steps.

(b) The (shortest) path *svwt* has a transit time of 0 and can thus be used in all time steps.

Figure 5.8: The application of the cost reduction on a graph that does not satisfy (N2).

exceeding the time horizon by a factor of $(1+\varepsilon)$ *of the optimal time horizon of the original problem.*

Proof. Assume T is the optimum time horizon. Let d be the length of a shortest s-t-path in $\mathcal{N} = (G = (V, E), u, \tau, s, t)$. In the transformation from the solution in the transformed network $\mathcal{N}' = (G = (V, E), u, \tau', s, t)$, we elongate the transit time of each path by d. Thus, the time horizon by approximating in the transformed network is $(1+\varepsilon)(T')+d = (1+\varepsilon)(T-d)+d = T-d+\varepsilon T-\varepsilon d+d = T+\varepsilon T-\varepsilon d \leq T+\varepsilon T = (1+\varepsilon)T$. □

Corollary 5.16. *The* Quickest Flow Problem *and the* Maximum Earliest Arrival Transshipment Problem *with a single sink can be approximated if transit times are almost non-negative.*

Proof. We compute reduced costs transit times by invoking Algorithm 5.1. The Quickest Flow Problem can be solved using a $(1+\varepsilon)T$-time-approximate algorithm, e. g., the one by Fleischer and Tardos [FT98]. The Earliest Arrival Transshipment Problem can be approximated using the algorithm from Fleischer and Skutella [FS07]. The solutions can be transferred back without losing time by Lemma 5.15. □

If the transit times are non-negative, the same reduction can be applied and achieves practical improvements for algorithms based on temporally repeated flows. The transit time reduction is implemented in the software suite zet and used for the examples in Section 3.

5.3 Approximating

By Corollary 5.4 we know that the Maximum Flow over Time Problem with general transit times can not be approximated. However, it is possible to approximate the flow value if we additionally relax the time constraint. Thus we can compute a transshipment that exceeds the time horizon by a factor of $(2+\varepsilon)$ similar to an algorithm by Fleischer and Skutella for the Quickest Transshipment Problem [FS07]. To show the bound we have to use length bounded network flows. For the computation of those flows we need length bounded shortest paths.

5.3.1 Length Bounded Shortest Paths

A simple scenario motivating length bounded shortest paths might be planning tours with a rented car. The renter might e. g., have signed a kilometer limit of T for the driven distance and has to pay a penalty if the distance is exceeded. Now, the renter wants to plan a tour from the base location s to some destination t. Because time is very valuable, the trip should be as fast as possible. So the target is the following: From all s-t-paths that are no longer than T find the one with the shortest travel time.

Formalizing the scenario we are given a graph $G = (V, E)$ with arc length ℓ_e and costs c_e for each arc $e \in E$, and a length bound T. For a path P in G, we denote the **length** by $\ell(P) \coloneqq \sum_{e\in P} \ell_e$, and the cost by $c(P) \coloneqq \sum_{e\in P} c_e$. A ***k*-length bounded path** is a path P satisfying

$$\ell(P) = \sum_{e\in P} \ell_e \leq k.$$

The Length Bounded Shortest Path Problem consists of finding a T-length bounded path with minimal costs. The problem is sometimes also referred to as Restricted Shortest Path Problem.

Problem:	Length Bounded Shortest Path
Instance:	Graph $G = (V, E)$ with lengths ℓ_e, costs c_e for each $e \in E$ and a length bound $T > 0$.
Task:	Find a T-length bounded path P minimizing the cost $c(P) \coloneqq \sum_{e \in P} c_e$.

The Length Bounded Shortest Path Problem has been studied already in the 1960s. With non-negative arc lengths on general graphs, or on acyclic graphs the problem can be easily solved in pseudo-polynomial time using dynamic programming [Jok66; WG65]. However, the problem is $\mathcal{NP}$-complete.

Observation 5.17 (Hardness of the Length Bounded Shortest Path Problem [GJ79])**.** *For a given length T and a cost value C it is $\mathcal{NP}$-hard to decide if a T-length bounded path with cost of at most C exists.*

Proof. The problem can be reduced to the Partition Problem[2] using the graph depicted in Figure 5.9. For a given Partition instance $a_1, \ldots, a_n$ we set the length bound to $\left(T \coloneqq \frac{1}{2}\sum_{i=1}^{n} a_i\right) + 1$. Each path has to use some of the upper arcs with length $\ell_{e_i} \coloneqq a_i$ but no costs and some of the lower arcs with no length but costs $c_{e_i} \coloneqq a_i$. Thus, an instance for the Partition Problem is feasible if and only if the shortest path has exactly length and cost T. □

The Length Bounded Shortest Path Problem problem can be approximated efficiently and fully polynomial-time approximation schemes have been developed by Hassin [Has92] and Warburton [War87]. Hassin's FPTAS is based on scaling the (pseudo-polynomial) dynamic program. Since then, the algorithms have been improved and current research tries to find fast algorithms solving the problem. State of the art $(1 + \varepsilon)$-approximation algorithms have runtimes of $\tilde{\mathcal{O}}(nm)$ [LR01; GRK+12]. The latest improvement of Bernstein [Ber12] reduces the time complexity for a fully polynomial approximation scheme in the undirected case to $\tilde{\mathcal{O}}\left(m(2/\varepsilon)^{\mathcal{O}\left(\sqrt{\log(n)}\log\log(n)\right)}\right)$.

[2] A similar reduction to the Knapsack Problem due to Megiddo is presented in [HZ80]. This reduction uses the knapsack size as length bound.

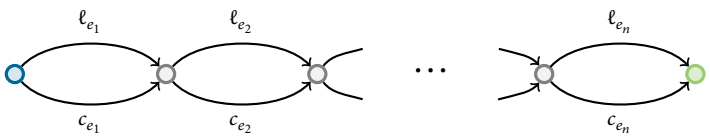


Figure 5.9: The graph used for the reduction from the Partition Problem to the Length Bounded Shortest Path Problem.

Solving Length Bounded Shortest Path by Dynamic Programming. In the following we will shortly describe the dynamic program for the Length Bounded Shortest Path Problem based on different length bounds as presented by Joksch [Jok66]. The main ingredient of the dynamic program is the Bellman equation

$$\min\left\{f(v,\theta-1),\min_{e=(u,v)\in E:\theta-\ell_e\geq 0}\{f(u,\theta-\ell_e)+c_e\}\right\}. \tag{5.1}$$

It stores the minimum cost at which we are able to reach a given node v at a specific time θ. This is either possible by taking over costs from the earlier time step $\theta-1$ or by reaching v via an arc (u,v) such that its travel time is not too long.

Algorithm 5.2: Dynamic Program for Length Bounded Shortest Path

---

Input: Acyclic graph $G=(V,E)$ with lengths $\ell_e\in\mathbb{N}_0$, costs $c_e\in\mathbb{R}$ for $e\in E$, and a length bound $T\in\mathbb{N}$.

Output: A shortest (in terms of costs c_e) T-length bounded path.

1. Initialize the table $f(v,\theta)$ with

$$\begin{aligned} f(s,0) &:= 0, \\ f(s,\theta) &:= \infty \qquad \text{for } \theta=1,2,\dots,T, \\ f(v,-1) &:= \infty \qquad \text{for } v\in V\setminus\{s\}. \end{aligned}$$

2. Solve table $f(v,\theta)$ dynamically using the equation

$$f(v,\theta) := \min\left\{f(v,\theta-1),\min_{e=(u,v)\in E:\theta-\ell_e\geq 0}\{f(u,\theta-\ell_e)+c_e\}\right\}$$

   for $v\in V\setminus\{s\},\theta\in\{0,1,\dots,T\}$.

3. Compute a shortest path by backtracking in the table f.

The optimal cost of a T-bounded path is stored in the table at position $f(t,T)$ after step 2 and the optimum shortest path can be reconstructed backwards until the source is reached. The total runtime of the algorithm is $\mathcal{O}(|E|\cdot T)$. It can be extended to support negative arc lengths easily by changing the Bellman equation. It is also possible to extend the algorithm to the case of general graphs (containing cycles) if the arc lengths remain non-negative. However, the introduction of both negative arc lengths and cycles leads to non-simple paths as optimal solutions.

Adaptation to Negative Travel Times without Waiting. We will use the length bounded shortest path problem as subroutine in an approximation algorithm for quickest flow with negative travel time without intermediate node storage. We will use the path returned as solution by a T-bounded shortest path instance to augment flow. For this purpose, we have to slightly change the algorithm. The Bellman equation (5.1) allows the cost of a shortest path to be taken from the previous time step as $f(v,\theta-1)$, or by using an arc (u,v). However, taking the solution from an earlier time step means

in fact waiting at a node to reach another node at a later point in time. This is only allowed in the source and sink. To account for this we have to change the equation for intermediate nodes; for the sink it is still allowed to take over earlier solutions. Additionally, we have to take care not to exceed the allowed time steps due to negative travel times.

This update the dynamic program from Algorithm 5.2 as follows to use it in the setting with negative arc lengths without waiting. The algorithm accepts arcs with negative length, but only accepts them if using them would not lead to paths entering the negative time range or exceed the length bound T. This is ensured by the modified Bellman equation

$$\min\left\{\min_{e=(u,v)\in E:\theta-\ell_e\in[0,T[}\{f(u,\theta-\ell_e)+c_e\}\right\}, \tag{5.2}$$

for intermediate nodes which only takes into account arcs that reach vertex v exactly at time θ that are used within within the interval $[0,T]$. The equation for the sink t remains unchanged. This leads to the following Algorithm 5.3.

Algorithm 5.3: Dynamic Program for the Length Bounded Shortest Path with negative arc lengths

---

Input: Acyclic graph $G=(V,E)$ with lengths $\ell_e \in \mathbb{Z}$, costs $c_e \in \mathbb{R}$ for $e \in E$, and a length bound $T \in \mathbb{N}$.

Output: A shortest (in terms of costs c_e) T-length bounded path.

1. Initialize the table $f(v,\theta)$ with

$$\begin{aligned} f(s,0) &:= 0, \\ f(s,\theta) &:= \infty \qquad \text{for } \theta = 1,2,\dots,T, \\ f(v,-1) &:= \infty \qquad \text{for } v \in \setminus\{s\}. \end{aligned}$$

2. Solve the table $f(v,\theta)$ dynamically using the equations

$$f(v,\theta) := \min\left\{\min_{e=(u,v)\in E:\theta-\ell_e\in[0,T[}\{f(u,\theta-\ell_e)+c_e\}\right\}$$

for $v \in V \setminus \{s,t\}, \theta \in 0,1,\dots,T$, and

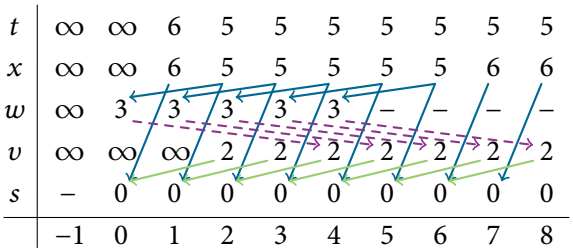
$$f(t,\theta) := \min_{(v_i,t)\in E}\left\{f(t,\theta-1), \min_{e=(v_k,t):\theta-\ell_e\in[0,T[}\{f(v_k,\theta-\ell_e)+c_e\}\right\}$$

for $\theta \in 0,1,\dots,T$.

3. Compute a shortest path by backtracking in the table f.

Example 5.18. *As an example consider the graph depicted in Figure 5.10 with node set $\{s,v,w,x,t\}$. The graph contains two possible s-t-paths; the upper path (s,x,t) with total cost 6 and the lower path (s,v,w,x,t) with total cost of 5. Due to the requirement that waiting is not allowed the path can only*

be taken from time 2 on and requires a time horizon of at least 4 due to the negative arc lengths. The table computed by the dynamic program from Algorithm 5.3 is the following.



t	∞	∞	6	5	5	5	5	5	5	5
x	∞	∞	6	5	5	5	5	5	6	6
w	∞	3	3	3	3	3	–	–	–	–
v	∞	∞	∞	2	2	2	2	2	2	2
s	–	0	0	0	0	0	0	0	0	0
	-1	0	1	2	3	4	5	6	7	8

Arcs denote which entries are considered for the computation of entries for a given node and time. The optimal cost of 5 is only possible to reach if the table entry at index $(w, 4)$ is available which is referred to by the first dashed arc, thus enforcing a time horizon of $T = 4$.

Approximating Length Bounded Shortest Path. We now derive an approximation algorithm for the Length Bounded Shortest Path Problem in the following way: For a given bound T, we compute a shortest path that may be a little longer, more precisely, the length is bounded by $(1+\varepsilon)T$. Due to the relaxation, all former shortest paths are still valid solutions, thus the cost of the returned path is at most the cost of a T-length bounded shortest path.

We solve the dynamic program by modifying arc lengths in a way that resembles time condensation but in the table of the dynamic program. First, we prove a lemma that allows us to compress the table in some situations. Assume all arcs have a length that is a multiple of $\Delta > 1$. In the table computed by the dynamic program only columns with indices that are multiples of Δ will contain other values than infinity (with exception of the row belonging to the target node). If all arc lengths are divided by Δ, the table also shrinks by a factor of Δ, and we can ignore the redundant rows.

Lemma 5.19. *Let $G = (V, E)$ with lengths $\ell_e \in \mathbb{Z}$, costs $c_e \in \mathbb{R}$ for $e \in E$, and a length bound $T > 0$ be an instance for the T-length bounded shortest path problem with general lengths having the property that all arc length $\ell_e = \ell'_e \cdot \Delta$ are multiples of Δ. It is possible to compute a T-length bounded path in time $\mathcal{O}(|E| \cdot \frac{T}{\Delta})$*

Proof. Calculate an $\frac{T}{\Delta}$-length bounded shortest path on an instance with path lengths ℓ'_e.

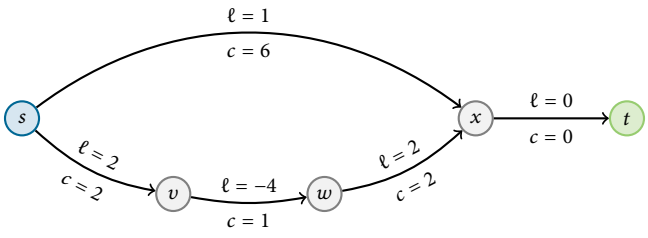


Figure 5.10: Example instance for the Length Bounded Shortest Path Problem with negative arc lengths. For a time horizon $T \in \{1, 2, 3\}$ the upper path is the optimal as solution, for larger time horizons the lower path is also available and used in the optimal solution.

Consider any T-length bounded path P. Than the length of P is

$$\sum_{e \in P} \ell'_e \cdot \Delta = \Delta \cdot \sum_{e \in P} \ell'_e \leq T.$$

Division by Δ shows that P is $\frac{T}{\Delta}$-length bounded with the new arc lengths. □

To compute $(1+\varepsilon)$-length bounded shortest paths we round up all arc length to a multiple of $\frac{\varepsilon T}{n}$. The length of each (simple) path can increase at most by a factor of εT, as it cannot contain more arcs than the number of nodes in the graph, which also necessitates the increase of the length bound. Due to the scaling, the table of the dynamic program becomes sparse, and hence we can apply the previous lemma. As a consequence we get the following theorem.

Theorem 5.20. *For an instance of the* Length Bounded Shortest Path Problem*, it is possible in polynomial time to find a $(1+\varepsilon)T$-bounded path having at most the costs of a shortest T-bounded path.*

Proof. Define $\Delta := \frac{\varepsilon T}{n}$. We scale all edge length to the next multiple of Δ by setting $\ell'_e := \left\lceil \frac{\ell_e}{\Delta} \right\rceil \cdot \Delta$. Any T-length bounded path P is then bounded by

$$\sum_{e \in P} \ell'_e = \sum_{e \in P} \left\lceil \frac{\ell_e}{\Delta} \right\rceil \cdot \Delta \leq \sum_{e \in P} \ell_e + \sum_{e \in P} \Delta \leq T + n \cdot \Delta = T + \frac{n\varepsilon T}{n} = (1+\varepsilon)T.$$

Executing the dynamic program 5.3 with bound $(1+\varepsilon)T$ returns a path with costs at most of the costs of a shortest T-bounded path. All transit times are multiples of Δ, thus by Lemma 5.19 we can scale the instance again to have time horizon of $\frac{(1+\varepsilon)T}{\Delta}$ and return the solution of this instance. The new length bound is $\frac{(1+\varepsilon)T}{\Delta} = \frac{(1+\varepsilon)Tn}{\varepsilon T} = \frac{(1+\varepsilon)}{\varepsilon} n$ which is polynomial in the input size and $\frac{1}{\varepsilon}$. □

5.3.2 Quickest Flows with Negative Travel Times

Using the FPTAS for T-bounded shortest paths, we develop a simple $\big((2+\varepsilon)T\big)$-time-approximation algorithm for Quickest Transshipment with general transit times $\tau_e \in \mathbb{Z}$ and bounded costs $C \in \mathbb{R}_{\geq 0}$. The result uses techniques developed by Fleischer and Skutella [FS07].

Even for Maximum Flow over Time, it is not possible to find temporally repeated solutions. However, if only paths P are used that are short enough such that the maximum reward $r_T(P)$ can be gained in a temporally repeated solution, it is possible to satisfy all demands with only doubling the necessary time horizon.

Definition 5.21 (T-length bounded flow)**.** Let f_i be a static multi-commodity flow satisfying node balances b_i at terminals $S_i^+ \cup S_i^-$ for each commodity $i \in K$. The flow f_i is called ***T*-length bounded**, if for each commodity the flow f_i can be decomposed into a path flow $\left(x_P^i\right)_{P \in \mathscr{P}^i}$, such that all flow carrying paths P with $x_P^i > 0$ satisfy $r_T(x_P^i) > 0$ and are available within time horizon T.

Let $G = (V, E)$ be a directed graph with costs c_e, capacities u_e, and arc length $\tau_e \in \mathbb{R}_{\geq 0}$ for arcs $e \in E$. Let $s, t \in V$ be the source and sink and d a demand. A **length bounded flow** is a path flow x_P for $P \in \mathscr{P}$ that can send all demands from the source to the sink. ◁

Problem: Length Bounded Flow

Instance: A dynamic network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$ with general transit times, demands and supplies b_v, costs c_e and a length bound $T > 0$.

Task: Find a T-length bounded path flow x that minimizes the costs

$$c(x) \coloneqq \sum_{P \in \mathcal{P}} \sum_{e \in P} c_e \cdot x_P.$$

Linear Programming Formulation. We propose a linear programming formulation to solve the Length Bounded Flow Problem. Without loss of generality we assume that each commodity only uses one source s_i and sink t_i. Otherwise, create supersources and supersinks for each commodity as in the extended network (cf. Definition 1.5).

Let $\mathcal{P}^{s_i t_i}$ be the set of all source-sink paths for commodity i. We then define

$$\mathcal{P}_i^T \coloneqq \{P \in \mathcal{P}^{s_i t_i} \mid r_T(P) > 0\}$$

to be the set of paths that are usable within time horizon T.

We extend the path based flow formulation from Definition 1.7 to set up a linear program for the multi-commodity case. For each commodity, flow can be sent on paths from $\mathcal{P}_i^T$ and the sum of flow must satisfy the demands $d_i = -b_{t_i}$. The costs on paths are defined as $c_{P,i} \coloneqq \sum_{e \in P} c_{e,i}$ This leads us to the following linear program to compute a length bounded multi-commodity minimum cost flow.

$$\begin{aligned}
\min \quad & \sum_{i \in K} \sum_{P \in \mathcal{P}_i^T} c_{P,i} x_P^i, && && \text{(MCF-LB)}\\
\text{s.t.} \quad & \sum_{i \in K} \sum_{P \in \mathcal{P}_i^T : e \in P} x_P^i && \leq u_e && \text{for all } e \in E,\\
& x_P^i && \geq 0 && \text{for all } i \in K, P \in \mathcal{P}_i^T.
\end{aligned}$$

By dualization and some algebraic rearranging we get the corresponding dual, which does not need an exponential number of variables any more but instead incorporates an exponential number of constraints.

$$\begin{aligned}
\max \quad & \sum_{i \in K} d_i z_i - \sum_{e \in E} u_e y_e, && && \text{(NP-LB)}\\
\text{s.t.} \quad & \sum_{e \in P} y_e + c_{e,i} && \geq z_i && \text{for all } P \in \mathcal{P}_i^T,\\
& z_i && \geq 0 && \text{for all } i \in K.
\end{aligned}$$

We will now combine the results to compute a length bounded flow that violates the allowed bound by at most a factor of $(1+\varepsilon)$ using the technique from Fleischer and Skutella [FS07].

Lemma 5.22. *Let x_P^i be a path decomposition of a T-length bounded multi-commodity flow with commodities $i \in K$ with costs $c(x)$.*

For every $\varepsilon > 0$, there exists a feasible $(1+\varepsilon)T$-length bounded static multi-commodity flow x' with lower costs $c(x') \leq c(x)$. The flow x' can be computed in polynomial time in the input size and $\frac{1}{\varepsilon}$.

Proof. If there exists a polynomial solvable separation problem that we can use to find violating constraints of (NP-LB), we could use Theorem 1.3 to solve the dual program in polynomial time. The only type of constraints is violated if there is a path in $\mathscr{P}_i^T$ such that the costs of the path is larger than z_i. This is equivalent to check whether the shortest T-length bounded (with probably negative arc lengths) has a cost of less than z_i. Unfortunately this is $\mathcal{NP}$-hard, but Theorem 5.20 allows us to find an approximate $(1+\varepsilon)T$-length bounded path whose cost is are bounded from above by the cost of a shortest path in $\mathscr{P}_i^T$.

We can now use the equivalence of optimization and separation to find a solution of a relaxed version of (NP-LB) that may contain additional constraints for paths of length at most $(1+\varepsilon)T$. If we turn the dual solution into a solution for the corresponding (relaxed) variant of the primal linear program (MCF-LB), we get a path based flow with minimum costs that uses paths in $\mathscr{P}_i^{(1+\varepsilon)T}$ for each commodity. As the solution space is relaxed, the solution has no higher costs than the optimum solution of (MCF-LB). □

Scaling of Flows. Central point of the approximation algorithm is the fact that any static T-length bounded path flow satisfying some demands d can be converted to a flow over time that satisfies demands with value $T \cdot d$ by doubling the time horizon. The idea is straight forward: The length of each flow carrying path is bounded by T, so the original demands can be satisfied within time T. If the flow is sent for T time units, the satisfied demands are also scaled by T.

Lemma 5.23. *Let x be a T-length bounded multi-commodity path flow satisfying a fraction $\frac{b_i}{T}$ of the demands for each commodity with total costs $c(x) \leq \frac{C}{T}$.*

Then there exists a path flow x' with time horizon $2T$ that satisfies all supplies and demands b_i with costs at most C.

Proof. We send flow of value x_P^i into path P for time T beginning at the earliest point in time. For defining the actual flow over time f_i, we define a temporary flow as

$$\hat{f}_i(P,e,\theta) \coloneqq \begin{cases} x_P^i & \text{if } \theta \in [\tau^-(P) + \tau([P_{s_i,\mathrm{tail}(e)}]), \tau^-(P) + \tau([P_{s_i,\mathrm{tail}(e)}]) + T[, \\ 0 & \text{otherwise.} \end{cases}$$

Flow can enter P first at time $\tau^-(P)$ waiting in intermediate nodes is not allowed. The flow arrives at arc e after time $\tau([P_{s_i,\mathrm{tail}(e)}])$. Using this notation, we can define

$$f_i(e,\theta) \coloneqq \sum_{P \in P_i : e \in P} \hat{f}_i(P,e,\theta)$$

to be the solution flow having the desired properties. f satisfies

$$\sum_{i \in K} f_i(e,\theta) = \sum_{i \in K} \sum_{P \in P_i : e \in P} \hat{f}_i(P,e,\theta) \leq \sum_{i \in K} \sum_{P \in P_i^T : e \in P} x_P^i \leq u_e$$

and therefore respects capacities. f also satisfies strict flow conservation. Let v be an intermediate node, i. e., on any path P using node v, there are arcs e, e' with $\text{head}(e) = v = \text{tail}(e')$. These arcs by definition satisfy $\hat{f}_i(P, e, \theta - \tau_e) = \hat{f}(P, e', \theta)$. In total

$$\begin{aligned}\sum_{i\in K}\sum_{e\in\delta^-(v)} f_i(e,\theta-\tau_e) &= \sum_{i\in K}\sum_{e\in\delta^-(v)}\sum_{P\in P_i:e\in P} \hat{f}_i(P,e,\theta-\tau_e)\\ &= \sum_{i\in K}\sum_{e'\in\delta^+(v)}\sum_{P\in P_i:e'\in P} \hat{f}_i(P,e',\theta) = \sum_{i\in K}\sum_{e'\in\delta^+(v)} f_i(e',\theta)\end{aligned}$$

holds. The demands of source s_i are satisfied due to

$$\sum_{e\in\delta^+(s_i)}\int_0^{2T} f_i(e,\xi)\,d\xi = \sum_{e\in\delta^+(s_i)}\sum_{P\in\mathcal{P}_i^T:e\in P}\int_{\tau^-(P)}^{\tau^-(P)+T} x_P^i\,d\xi = T\cdot\frac{1}{T}b_{s_i} = b_{s_i}.$$

The same computation holds for the costs, so the costs of f are bounded by C. □

Algorithm 5.4: Length Bounded Approximation for the Quickest Transshipment Problem

Input: A dynamic network $\mathcal{N} = (G = (V, E), u, \tau, S^+, S^-)$, supplies and demands for the sources and sinks $b : V \to \mathbb{R}$, costs $c_{e,i}$ together with a cost bound C, a time horizon $T \in \mathbb{N}$ and the precision $\varepsilon > 0$.

Output: A flow over time satisfying all supplies and demands within time horizon $(2 + \varepsilon)T$ that is bounded by costs C, if exists. Otherwise, there is no flow within time horizon T.

1. Compute a static $(1 + \varepsilon)T$-length bounded path flow x that satisfies a $\frac{1}{T}$-fraction of the supplies and demands with costs at most $c(x) \le \frac{C}{T}$ if such a flow exists, otherwise return without solution.
2. Compute a flow with time horizon $(2 + \varepsilon)T$ using Lemma 5.23 and return it.

Using a binary search frame work we can compute a flow over time that exceeds the optimal time horizon by just a factor of $2 + \varepsilon$.

Theorem 5.24. *A solution for the Quickest Transshipment Problem (with multiple commodities) with bounded costs on instances without negative cycles that does not need flow storage and that exceeds the optimal makespan by at most a factor $(2 + \varepsilon)$ for $\varepsilon \in]0, 4]$ can be computed in polynomial time.*

Proof. Let T^* be the minimal makespan that is necessary to ship all supplies. For an arbitrary $\delta > 0$ we can use Algorithm 5.4 to find a $(1 + \delta)$-approximation of the timespan in time $\mathcal{O}(\log(\frac{T^*}{\delta}))$ by dividing the time interval in small steps of length δ and calling the algorithm in a binary search framework. After the binary search we get a value T with

$$T^* \le T \le (1 + \delta)T^*.$$

If we call Algorithm 5.4 with the approximative length bound T and precision δ we get a flow over time with makespan

$$(2 + \delta)T \le (2 + \delta)(1 + \delta)T^* \le (2 + 3\delta + \delta^2)T^*.$$

To achieve the desired approximation guarantee we have to chose δ such that

$$(2 + 3\delta + \delta^2)T^* \leq (2 + \varepsilon)T^*.$$

The inequality holds for $\delta \coloneqq \frac{\varepsilon}{4}$. □

Corollary 5.25. *The Maximum Flow over Time Problem with general transit times without negative cycles can be approximated such that for a given time horizon T at least as much flow can be sent as is possible for time $\frac{T}{2+\varepsilon}$ for $\varepsilon \in]0, 4]$.*

Proof. We use binary search to find the maximum amount of flow b that can be sent from sources to sinks within time T using the algorithm described in Theorem 5.24. A feasible flow has a time horizon of T and sends at least as much flow as was possible to send within a time of $\frac{T}{2+\varepsilon}$. □

Tightness. The approximation factor given by Theorem 5.24 is tight if we are restricted to use temporally repeated solutions. Consider the example with k sources and a single sink depicted in Figure 5.11. Any temporally repeated flow based on a static flow loses a factor 2 in the time horizon to send the same amount of flow. An optimal flow can send k units of flow within a time horizon of $T = k$ if one flow unit is sent from each source to the sink. The k units share the arc (w, t) at disjoint time steps. However, any static flow can send at most one unit of flow over the limiting bottleneck arc, thus needing additional $k - 1$ time steps to send the k units of flow.

Extensions. In the proof of Lemma 5.4 we used the class of instances depicted in Figure 5.4 to prove that the Maximum Flow over Time Problem can not be approximated if the network contains negative travel times. However, the instances do neither satisfy (N1) nor (N2). Another type of networks that can be used to prove $\mathcal{NP}$-hardness of the problem is depicted in Figure 5.12. This class of instances satisfies (N1), e. g., all paths can be used directly from the beginning on. However, yes-instances of Partition only increase the maximum flow value form T to $T + 1$ which does not allow to prove an inapproximability factor. Therefore, it is an open question if there exists an approximation algorithm for the maximum flow value within a given time horizon for instances whose transit times satisfy exactly one of (N1) or (N2).

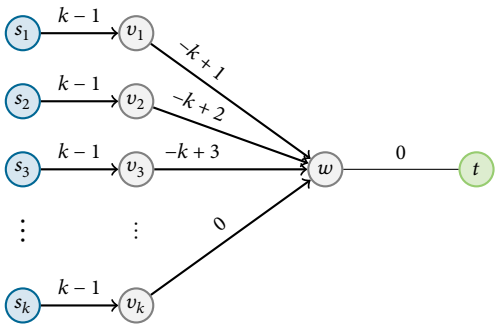


Figure 5.11: An example showing that the bound proven in Theorem 5.24 is tight.

Better Approximation Factors. The approximation algorithm in this section is based on temporally repetition of a static (length bounded) flow. This allows to approximate the Quickest Transshipment Problem and to compute a time-approximate maximum flow. For problems with non-negative transit times better approximation results are achieved by using condensed time-expanded networks. We conjecture that it is not possible to achieve good approximation results by using only simple forms of time condensation such as geometric time condensation as used in Section 4.4. The approximation bounds rely on the fact that optimal solutions do not need non simple paths, but such paths are necessary as we have seen. A strong hint in that direction is the network depicted in Figure 5.13. Networks with non-negative transit times have the property that a (single-commodity) flow sending D units of flow within a time horizon of T can be scaled such that it sends δD units of flow within a time horizon of δT for any $\delta > 0$. This property is not true for the network in the figure. It is possible to send 3 units of flow within a time horizon of $T = 3$, however it is not possible to send additional flow within a time horizon of $T = 4$.[3]

Sequence Rounded Time Condensation. A promising direction to achieve good approximate solutions seems to be the sequence rounded time condensation technique which was introduced by Groß and Skutella [GS12b] to approximate the Maximum Multi-commodity Flow over Time Problem that also requires non simple paths in optimal solutions. For this problem they also show that the scaling of flow does not work. To apply their technique one first has to show that the condensed network remains polynomially small if more sequences are added which is necessary for negative transit times. Second, one has to prove that smoothing of flows in the condensed network still results in a feasible flow in the original network. The approximation algorithm by Groß and Skutella also relies on solving a dynamic program to find violated constraints. As last ingredient one therefore has to show that the appropriate Separation Problem is still solvable in polynomial time for negative sequences.

5.4 Matchings over Time

Temporal extensions of combinatorial optimization problems have not been studied in many areas of combinatorial optimization. The most famous example, flows over time, is the main topic of this thesis. Another field which produces problems that naturally have a temporal dimension is scheduling theory, where time often is inherent to problems (see e. g., [Pin12] for an introduction and overview to scheduling). Recently, Adjiashvili et al. [ABW+14] introduced a general model for time-expanded versions of packing problems. We now introduce the Matching over Time

[3] This property is proven in Lemma 4.8 in [FS07].

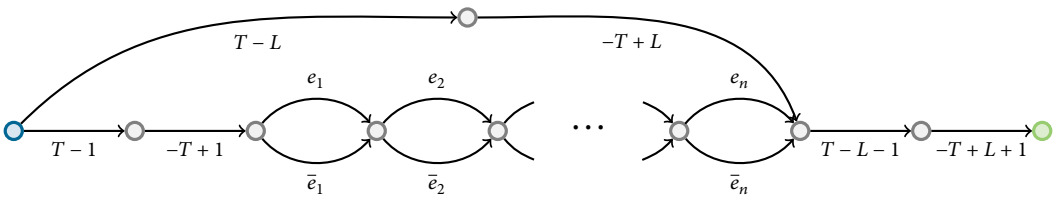


Figure 5.12: A direct reduction for maximum flow over time with general travel times to partition.

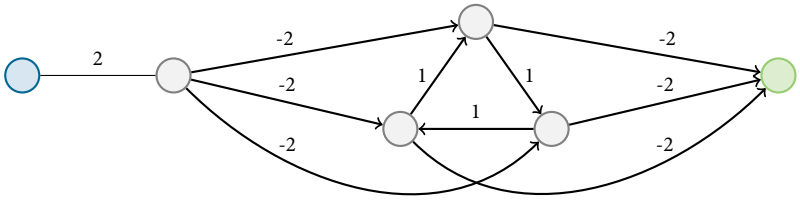


Figure 5.13: Viaolation of Lemma 4.8.

Problem, a generalization of the well known Matching Problem that lies in the intersection of network flows and scheduling and see how network flows can be used to solve a special case of the problem.

For our version of matchings over time we assume that each edge $e = \{v, w\}$ has different delays $\tau_e(v)$ and $\tau_e(w)$ at both incident nodes. If the arc is activated at some time θ, after the respective delay times it is matched once to v at time $\theta + \tau_e(v)$ and to node w at time $\theta + \tau_e(w)$. As we want to compute a matching, at each point in time only one edge can be matched to a node. Notice that this idea allows that arcs can be activated in arbitrary time steps and that nodes can be matched to different arcs in each time step. In contrast to this definition, Adjiashvili et al. [ABW+14] propose a different variant to extend the matching problem into the temporal dimension. They assume that arcs can be matched to nodes without any delay, however an edge e has an assigned production time τ_e that requires the edge to be matched at times $\{\theta, \theta + 1, \dots, \theta + \tau_e - 1\}$, if it is activated at time θ. We now define *matchings over time* more formally.

Definition 5.26 (Matching over Time)**.** Let $G = (V, E)$ be an undirected graph, $T \in \mathbb{N}$, and let $\tau_e(v)$ be delays for each edge $e \in E$ and node v incident to e.

A **matching over time** is a set of edge-time tuples $M = \{(e, \theta) \mid e \in E, \theta \in \{1, 2, \dots, T\}\}$ that satisfies the following requirements. For an edge-time tuple (e, θ) edge $e = \{v, w\}$ saturates node v at time $\theta + \tau_e(v)$ and at node w at time $\theta + \tau_e(w)$. The matching over time has to obey the time horizon, e. g.,

$$\max\{\theta + \tau_e(v), \theta + \tau_e(w)\} \leq T$$

holds for all $(e = \{v, w\}, \theta) \in M$. For a feasible matching we require that at each time θ, at most one edge is incident to each node $v \in V$, e. g., M should satisfy

$$|\{(e, \xi) \in M \mid e \in \delta(v) \wedge \xi + \tau_e(v) = \theta\}| \leq 1.$$

◁

Problem:	Matching over Time
Instance:	An undirected graph $G = (V, E)$, a time horizon T, and delays $\tau_e(v)$ for each edge $e = \{v, w\} \in E$.
Task:	Compute a matching over time M that maximizes $\lvert M \rvert$.

Simple Cases. We first consider a simple case that is easy to solve. If all delays $\tau_e(v) \equiv 0$ are zero for all edges $e \in E$ and nodes $v \in V$ the matchings are independent in each time step and the cardinality of M can be maximized by computing a static maximum matching M^s on graph $G = (V, E)$ and set $M := \{(e, \theta) \mid e \in M^s \wedge \theta \in \{1, 2, \dots, T\}\}$. A solution of this type resembles temporally repeated flows in the sense that a solution is repeated for all feasible time steps. However, we will see that such solutions do not exist in general and that the problem is $\mathcal{NP}$-hard already for bipartite graphs.

time-expanded Matchings. Generally, a matching over time can be computed in an undirected time-expanded graph that is defined the following way (cf. also Definition 2.4). For a given time horizon T we create T copies of every node and use

$$V^T := \{v^\theta \mid v \in V, \theta \in \{1, 2, \dots, T\}\}$$

as node set. The edge copies connect node copies with respect to their delay time, e. g.,

$$E^T := \left\{ e^\theta = (v^{\theta+\tau_e(v)}, w^{\theta+\tau_e(w)}) \;\middle|\; \begin{array}{l} e = (v, w) \in E, \\ \max\{\theta + \tau_e(v), \theta + \tau_e(w)\} \leq T \end{array} \right\}.$$

Lemma 5.27. *Let $G = (V, E)$ be an undirected graph, T a time horizon and $\tau_e(v)$ delays for each edge $e \in E$. Then a static matching M^s in the time-expanded graph $G^T = (V^T, E^T)$ corresponds to a matching over time M in G with same value $|M^s| = |M|$ and vice versa.*

Proof. Let M^s be a static matching in G^T. Define $M := \left\{(e, \theta) \mid e^\theta \in M^s\right\}$. Then M is a feasible matching over time. Because e^θ is only present in E^T if $\max\{\theta+\tau_e(v), \theta+\tau_e(w)\} \leq T$, each edge-time tuple (e, θ) obeys the time horizon. For the matching property consider a node copy v^θ. Because M^s is a matching, we have $\left|\left\{e^\theta \in M^s \mid e^\theta \in \delta(v^\theta)\right\}\right| = 1$. Assume now, that two edges in M are matched to v at time θ, e. g., there exist tuples (e_1, ξ_1) and (e_2, ξ_2) in M with $\xi_1 + \tau_{e_1}(v) = \theta = \xi_2 + \tau_{e_2}(v)$. However, this is not possible because then $e_1^{\xi_1}$ and $e_2^{\xi_2}$ would be present in M^s violating the matching property at v^θ. The other direction follows by the same arguments. □

Example 5.28. *As an example we consider a simple instance on the vertex set $V = \{a_1, a_2, b_1, b_2\}$ and the edge set $E = \{e_1 = (a_1, b_1), e_2 = (a_1, b_2), e_3 = (a_2, b_2)\}$ as depicted in Figure 5.14a. Arc (a_1, b_1) is available from the beginning and has a delay of 1 at b_1, arc (a_1, b_2) has delays of 3 and 2 at a_1 and b_2, respectively. Finally, arc (a_2, b_2) is only available from time 3 onwards because both delay times are equal.*

Temporally Repeated Matchings. The maximum matching over time in the example has a value of 7 and one possible solution is depicted in Figure 5.14c. Observe, that the solution in the figure is not temporally repeated in the sense that the solution does not only contain edges that are matched at every possible time. However, the matching over time $M = \{(e_1, 1), (e_1, 2), (e_1, 3), (e_1, 4), (e_3, 1), (e_3, 2), (e_3, 3)\}$ has this property. Analogously to maximum network flows such a solution can be constructed by a static weighted matching. Let us define edge weights $w_e := T - \max\{\tau_e(v), \tau_e(w)\}$. Then the weight represents how often an arc can be used in a matching over time. For a time horizon of $T = 5$, in the example we have $w_{e_1} = 4$, $w_{e_2} = 2$, and $w_{e_3} = 3$ and the maximum weighted static matching with this weights is $M^s = \{e_1, e_2\}$ generating the above matching over time M.

However, in contrast to flows over time there are not always temporally repeated optimal matchings. The instance depicted in Figure 5.14b does not allow for an optimal temporally repeated solution. Assume a time horizon $T = 2$. Then the maximum weighted matching is $M^s = \{e_2, e_4\}$ with a total weight of 4. However, the maximum matching over time $M = \{(e_1, 0), (e_2, 1), (e_3, 0), (e_4, 0), (e_5, 0)\}$ has a value of 5.

Bipartite Graphs. Static matchings can be reduced to static network flows by a well known reduction, see for example the text book by Burkard, dell'Amico, and Martello [BdAM09]. We now apply a similar technique in the temporal setting. Let an instance for the MATCHING OVER TIME PROBLEM be given including a bipartite graph $G = (V = A \cup B, E)$, a time horizon T and arc delays $\tau_e(v)$. We define the corresponding **matching over time network** $\mathcal{N}'$ as follows. As vertex set

$$V' \coloneqq V \cup \{s\} \cup \{t\}$$

we take the original vertices together with an additional source and sink. We direct all arcs from set A to set B and connect the source to nodes in A and the nodes in B with the sink, e. g.,

$$E' \coloneqq \{(a,b) \in E \mid a \in A, b \in B\} \cup \{(s,a) \mid a \in A\} \cup \{(b,t) \mid b \in B\}.$$

We define the parameter for arcs $e = (a,b)$ for all $a \in A, b \in B$ as follows. The transit time is set to $\tau'_e \coloneqq \tau_e(b) - \tau_e(a)$, which might be negative. Additionally, we set the release date of e to $r_e \coloneqq \min\{\tau_e(a), \tau_e(b)\} + 1$. The transit times and release dates of arcs connected to the source or sink are set to zero and we do not impose deadlines. All edge capacities u'_e are unit capacities.

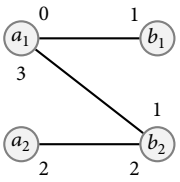
Lemma 5.29. *Let $G = (V = A \cup B, E)$ be a bipartite graph, T be a time horizon and $\tau_e(v)$ be edge delays for all edges $e \in E$ and v incident to e. Let $\mathcal{N}' = (G = (V', E'), u', s, t)$ be the corresponding matching over time network. Then, a discrete maximum flow over time with arc release dates and negative travel times that without waiting in intermediate nodes in $\mathcal{N}'$ corresponds to a matching over time with the same value.*

Proof. Let f be a feasible discrete network flow over time respecting release dates, arc capacities and possibly negative travel times. Because the capacity is limited to 1 on all arcs and the network structure guarantees that the path decomposition only contains paths of the form (s, a, b, t) for some $a \in A, b \in b$. Especially, at most one flow unit can reach node a at time θ. Because the flow does not use waiting in intermediate nodes, the flow unit immediately continues along arc $e = (a,b)$. Based on this observations we define a matching over time

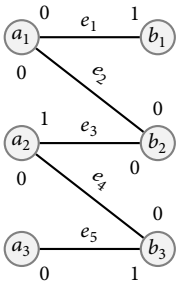
$$M \coloneqq \{(e, \theta + \min\{\tau_e, 0\} - r_e + 1) \mid f(e,\theta) = 1\}.$$

We verify that the activation dates are feasible by checking the definition. For arcs with positive travel times $\tau_e \geq 0$ the activation time reduces to $\theta - r_e + 1 \geq 1$ because no flow is on e before the release date. For arcs with negative travel time $\tau_e < 0$, we get $\theta + \tau_e - r_e + 1 = \theta + \tau_e(w) - \tau_e(v) - \tau_e(w) + 1 = \theta - \tau_e(v) + 1 \geq 1$ as start date, because no flow is on e before $\tau_e(v)$ due to the negative travel time and the release date.

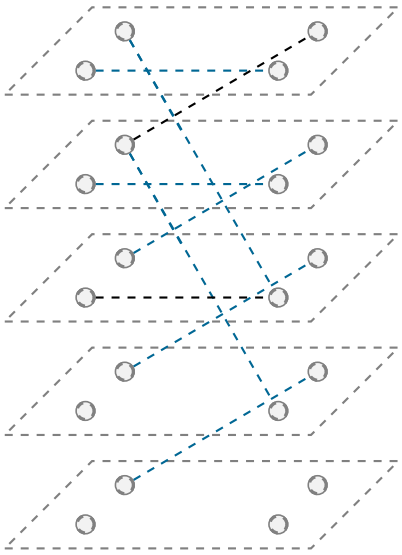
If flow is on $e = (a,b)$ at time θ, the arc is matched to a at time $\theta \leq T$ by definition of M and feasibility of the flow over time with time horizon T. Similarly, the edge is matched with b at time $\theta + \tau_e(w) \leq T$, as can be verified by inserting the definition. Thus, the matching over time M obeys



(a) The numbers at the nodes incident to edges denote their respective delays.



(b) An instance that does not allow for a temporally repeated solution.



(c) A maximum matching over time for the given instance.

Figure 5.14: The MATCHING OVER TIME PROBLEM instance from Example 5.28 and a time-expanded solution.

the time horizon. Finally, also at any point in time at most one edge is matched to any node. This follows for $a \in A$ by definition of M as the inflow is limited by one. In the same way, a node b with two matched arcs would imply two units of flow arriving at b at the same time which is not possible due to flow conservation and the fact that waiting in intermediate nodes is not allowed. □

Polynomial Solvability. In contrast to the results on network flows with release dates in Section 2.3 and on flows over time with negative travel times earlier in this chapter the reduction in Lemma 5.29 does not provide a polynomial algorithm in general. If all arcs in the generated network have no release dates and positive transit times, a solution can be computed using Algorithm 2.1. Notice, that for those instances the solution is also temporally repeated and thus can be represented in polynomial size. It is easy to check whether an instance for the MATCHING OVER TIME PROBLEM

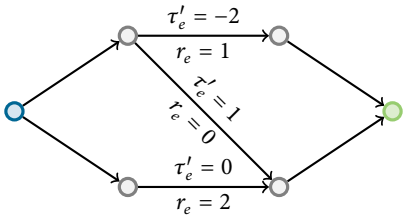


Figure 5.15: The generated matching over time network for the instance depicted in Figure 5.14a. The release dates and transit times not depicted are zero, the capacities are 1 for all arcs.

allows for such a solution. This is the case if and only if either $\tau_e(a) = 0$ for all $a \in A$ holds, or $\tau_e(b) = 0$ for all $b \in B$. There is also no polynomial algorithm in general, unless $\mathcal{P} = \mathcal{NP}$.

Theorem 5.30. *The* MATCHING OVER TIME PROBLEM *is weakly $\mathcal{NP}$-hard, even if the input graph is bipartite.*

Proof. Let $a_1, a_2, \dots, a_n$ be the values of the PARTITION instance in sorted order, i. e., $a_1 \leq a_2 \leq \cdots \leq a_n$. We define a time horizon $T := \frac{A}{2} + 1$. For the reduction we define a bipartite graph such as depicted in Figure 5.16. We use $2n + 2$ nodes v_i and w_i for $i = 0, 1, \dots, n + 1$. For each number a_i in the PARTITION instance we connect vertices v_i and w_i with two parallel arcs e_i and f_i. We also create arcs $z_i := (v_i, w_{i+1})$ for $i = 0, 1, \dots, n + 1$ and two arcs $g := (v_0, w_0)$ and $h := (v_{n+1}, w_{n+1})$. The edges f_i are equipped with delays $\tau_{f_i}(v_i) := 0$ and $\tau_{f_i}(w_i) := a_i$ and we set $\tau_g(v_0) := \tau_h(v_{n+1}) := 0$. All other edge delays are set to zero.

The idea behind the reduction is as follows. We would like to use edges g and h and define a perfect matching for a `yes`-instance. The former arc is connected with w_0 at time 0, while the latter is connected at v_{n+1} at time T. We try to find a path starting in g and ending in h that skips exactly the $\frac{A}{2}$ time steps in between. Only arcs e_i have a different delay, thus skipping time refers to selecting an element a_i into a solution of the PARTITION PROBLEM, and we show that we reach time layer T if and only if the PARTITION instance is feasible.

We start by observing the following fact. Let $\theta \in \{2, 3, \dots, T - 1\}$ be a time step and for $i < j$ nodes w_i and v_j are being matched by arcs e_i and f_j at times $\theta - a_i$ and θ, respectively. We then can match all nodes (besides the unreachable nodes v_0 and w_{n+1}) at time θ. To achieve this we match nodes w_0 to v_i by activating arcs $z_1, \dots, z_{i-1}$ at time θ. w_i and v_j are already matched by assumption. We match the intermediate nodes by activating arcs $e_{i+1}, \dots, e_{j-1}$. Finally, the last nodes $w_{j+1}, \dots, v_{n+1}$ can be matched by taking arcs $z_j, \dots, z_k$. On the other hand, if no vertices are matched by time θ, we can match all of those vertices by activating arcs $z_0, \dots, z_n$ at time θ.

Now, let $I \subseteq \{1, 2, \dots, n\}$ denote the indices of values in a feasible solution for a `yes`-instance with the following property: Whenever two or more elements a_i have the same value, the one(s) with smaller index are chosen. This is possible without loss of generality. Let π be a permutation, such that $I = \{\pi_1, \pi_2, \dots, \pi_k\}$. We define levels for each element in the solution by $L(i) := \sum_{j<i} a_{\pi_j} + 1$. We add $(f_{\pi_i}, L(i))$ to the matching over time. Consider a point in time $\theta = L(i) = L(i - 1) + a_{\pi_i}$ for $i = 1, \dots, k$. Due to our matching of f_{π_i} and $f_{\pi_{i-1}}$ two nodes $w_{\pi_{i-1}}$ and v_{π_i} are matched at time θ. Because of the assumed ordering of the elements we have $\pi_{i-1} < \pi_i$ and therefore our observation allows us to match all other nodes at time θ. At time $\theta = 1$ exactly one node v_{π_1} is already matched. We can match nodes $v_1, \dots w_{\pi_1 - 1}$ and $w_{\pi_1}, \dots, v_{n+1}$ by matching appropriate e_j and z_j. At time T exactly one node w_{π_k} is matched already. By selecting appropriate edges e_j and z_j we can match the nodes $w_0, \dots, v_{\pi_k}$ and $v_{\pi_k + 1}, \dots, w_n$. For the remaining time steps with $\theta \neq L(i)$ for all i from 2 to T, we can match all nodes by taking arcs z_j as described above. Now, the only unmatched nodes are v_0

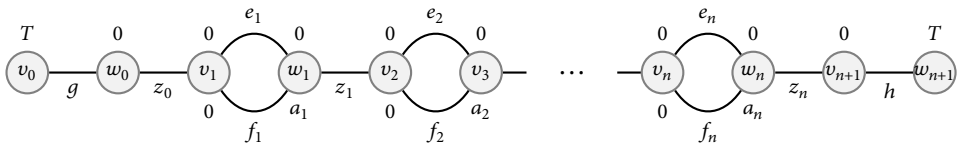


Figure 5.16: The reduction of the MATCHING OVER TIME PROBLEM to the PARTITION PROBLEM.

at time T, w_0 at time 1, v_{n+1} at time T, and w_{n+1} at time 1. These last two node pairs can be matched by adding $(g, 1)$ and $(h, 1)$ to the matching over time. Thus we have matched all nodes at all possible times.

Conversely, every perfect matching that matches all nodes to edges at all possible times defines a solution of a given PARTITION instance. Therefore $(g, 1)$ has to be in the matching and z_1 cannot be matched at time 0. We cannot match all nodes at time 1 by only taking arcs e_j, because then node v_{n+1} would remain unmatched. Therefore some arc f_j has to be selected at time 1. We define $\pi_1 := j$. The remaining arcs at time 1 (apart from w_{n+1}) can only be matched by selecting arcs $z_{\pi_1}, \dots, z_n$. In particular, no other edge f_j can be contained in the matching over time. Assume now the first i edges $f_{\pi_1}, \dots, f_{\pi_i}$ have been matched already. At level $L(i)$ node w_{π_i} is matched. Observe that it is only possible to match all node copies $v_0, \dots, v_{\pi_i}$ by including arcs $z_0, \dots z_{\pi_{i-1}}$ into the matching. If another edge e_j or f_j with $j < \pi_i$ is included in the matching at least w_0 remains unmatched. Also, not all elements can be matched by selecting only arcs e_j for $j > \pi_i$. Therefore, exactly one arc f_j is contained in the matching over time for time $L(i)$. Define $\pi_{i+1} := j$. We continue in this way until we have $L(k) = T$. The edges in the last point in time can be matched using the same arguments, but because no further arcs of type f_j can be selected, node v_{n+1} is not matched by arc z_n, but instead h has to be contained in the matching. The elements $\{a_{\pi_1}, \dots, a_{\pi_k}\}$ form a feasible solution of the PARTITION instance. □

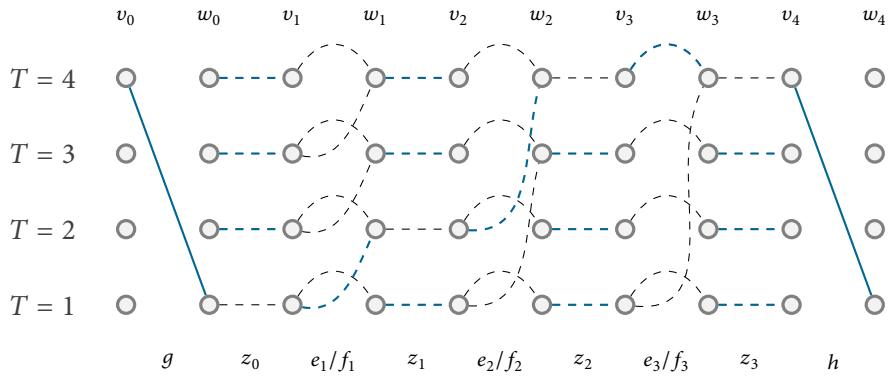


Figure 5.17: An example of the reduction for a PARTITION instance with $a_0 = 1$, $a_1 = 2$ and $a_3 = 3$. The graph corresponds to the time-expanded matching graph. The matching arcs are the solid arcs.

6 Abstract Flows over Time

Abstract networks and abstract flows are a generalization of classical network flows by Hoffman [Hof74]. The model generalizes the notion of paths that replaces the underlying network structure. Paths are a linearly ordered sets satisfying the *switching axiom*. Hoffman used the model to analyse the minimal structural properties necessary to retrieve a Max-Flow=Min-Cut-Theorem.

We present an algorithm that computes lexicographically abstract maximum flows. The algorithm is then used to prove existence of abstract earliest arrival flows. The result is complemented by an algorithm that computes value-approximate earliest arrival flows.

Publication Remark: Parts of this chapter are joint work with Jannik Matuschke and Britta Peis and appeared in [KMP14].

The concept of *abstract flows* was introduced by Hoffman [Hof74] when he reviewed the first proof of the Max-Flow=Min-Cut-Theorem by Ford and Fulkerson [FF56] again. He noticed that the (non-constructive) proof does not make use of many of the typical structural properties of networks. This is in contrast to their (constructive) proof given later [FF62], which also leads to the class of augmenting path algorithms. As typical structures we see for example the possibility to switch between two s-t-paths P and Q that share an arc e: There is a path starting with arcs on P until e and then continues with the arcs of Q after e. The same is true for a path starting in Q that switches to P after e.

Ford and Fulkerson's proof uses a relaxed version of this property which motivates the so-called *switching axiom*: For two paths that cross in a common element, there is another path that only contains elements of the beginning of P and the end of Q. However, the path is not required to use all of the elements (not even the common element e) and it is totally possible that the elements appear in different order. In the general model there still is an Abstract-Max-Flow=Min-Cut-Theorem, which states that abstract flows remain totally dual integral.

Similar to the proof of the Max-Flow=Min-Cut-Theorem in [FF56], the results in [Hof74] are of a theoretical nature and do not allow for a direct polynomial algorithm. In the abstract flow model we cannot search through the set of paths because it is too large in general. Instead, a polynomial abstract flow algorithm may call a separation oracle. It was a long open question, whether a polynomial algorithm computing an (integral) maximum abstract flow exists until McCormick [McC96] presented a polynomial primal-dual algorithm for the unweighted case. The algorithm was extended to the general case with additional weights on paths by Martens and McCormick [MM08]. Martens [Mar07] applies the algorithms to compute unsplittable abstract network flows.

In the spirit of abstract flows, further abstractions based on uncrossing axioms have been proposed. For the case of *lattice polyhedra*, corresponding totally dual integral results have been established by Hoffman and Schwarz [HS78], Gröflin and Hoffman [GH82] and Hoffman [Hof78]. Similar results on switchdec polyhedra are due to Gaillard [Gai97]; see also the survey due to Schrijver [Sch84].

Abstract Flows over Time. Despite the level of abstraction in Hoffman's model, many interesting results from classical theory of flows can be taken over into the abstract setting. *Abstract flows over time* are a generalization of abstract flows in which elements are assigned a travel time that is necessary for flow to pass along the element. In this setting it is possible to derive similar results as Ford and Fulkerson [FF58] showed for the Maximum Flow over Time Problem problem. Subsequently, we are interested whether there are also abstract flows over time that have the earliest arrival property. An earliest arrival transshipment is a flow that also satisfies supplies and demands that are given for first and last elements on paths. If the paths do not all start and end in the same element, earliest arrival transshipments do not exist since they are a generalization of network flows over time.

Outline of the Chapter. In Section 6.1 we begin with an introduction into the field of abstract flows and abstract flows over time. We present two possibilities to define time-expanded abstract networks and show some basic structural properties of abstract networks. Lexicographically maximum abstract flows are introduced in Section 6.2. We develop an algorithm that computes such flows for sequences of sources (or sinks) that adhere to a certain order. In Section 6.3 we introduce the abstract version of earliest arrival flows. We show that such flows exist in general and can be computed using the lexicographically maximum flow algorithm. We then add a notion of supplies and demands for source elements and sink elements and show that 2-value-approximate abstract earliest arrival flows exist.

6.1 Introduction to Abstract Flows

In this section we introduce Hoffman's abstract network flow model and extend it to a temporal setting by adding transit times. Furthermore we introduce an abstract version of time expansion and show some properties of abstract networks.

Definition 6.1 (Abstract network)**.** Let E be a finite ground set of **elements** and $\mathscr{P} \subseteq \mathcal{P}(E)$ be a family of **paths**. In an **abstract path system** $(E, \mathscr{P})$ each path $P \in \mathscr{P}$ has a linear order $\leq_P$ of its elements and weights $r_P : \mathscr{P} \to \mathbb{Q}_{\geq 0}$.

For a given path P and two elements $a <_P e$ we say a is **left** of e, and for $a >_P e$ we say that a is **right** of e. The elements up to e and beginning at e are defined as

$$P_{[\to e]} \coloneqq \{a \in P \mid a \leq_P e\}, \qquad \text{and} \qquad P_{[e\to]} \coloneqq \{a \in P \mid a \geq_P e\}.$$

Observe that these sub-paths include element e. We denote the elements on P that are left of e and right of e by

$$P_{[\to e[} \coloneqq \{a \in P \mid a \leq_P e\}, \qquad \text{and} \qquad P_{]e\to]} \coloneqq \{a \in P \mid a \geq_P e\}.$$

The **first** element $\operatorname{first}(P)$ of a path P is the element e such that $e \leq_P a$ for all $a \in P$. Correspondingly $\operatorname{last}(P)$ denotes the **last** element e, which satisfies $e \geq_P a$ for all $a \in P$.

Two paths P and Q cross at element e if $e \in P \cap Q$. An abstract path system satisfies the **switching axiom** if for any two paths P and Q crossing in element e there is a path that only uses elements at the beginning of P and at the end of Q and vice versa for elements at the beginning of Q and at the

end of P. More formally, both sets $\left\{R \in \mathscr{P} \mid R \subseteq P_{[\to e]} \cup Q_{[e\to]}\right\}$ and $\left\{R \in \mathscr{P} \mid R \subseteq Q_{[\to e]} \cup P_{[e\to]}\right\}$ are non-empty. We then define the switch paths of P and Q to be

$$P \times_e Q \in \arg\max\{r_P \mid R \in \mathscr{P}, R \subseteq P_{[\to e]} \cup Q_{[e\to]}\} \tag{6.1}$$

and

$$Q \times_e P \in \arg\max\{r_P \mid R \in \mathscr{P}, R \subseteq Q_{[\to e]} \cup P_{[e\to]}\}. \tag{6.2}$$

An abstract path system that satisfies the switching axiom together with capacities $u_e : E \to \mathbb{Q}^+$ on the edges is an **abstract network**. ◁

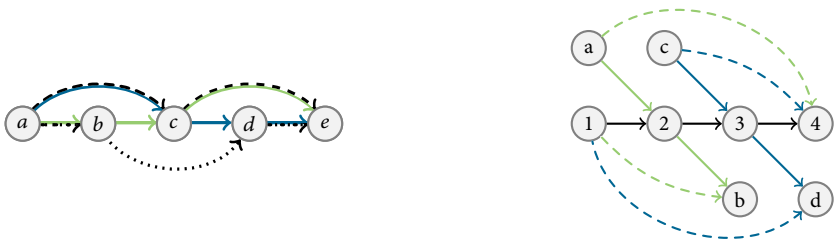
The notation generalizes classical networks as defined in Section 1.2. For a given network $\mathcal{N} = (G, u, s, t)$ the set of arcs E together with the set of all s-t-paths defines an abstract network if we assume $r_P \equiv 1$ for all paths. The switch of two paths P and Q at e is then the classical intersection of paths where we start with P until e and continue with the arcs on Q. Notice that the definition of an abstract path system does not require any relation between the order of the elements on paths. If P and Q both contain elements e and f, it is totally possible that $e <_P f$ on P but $e >_Q f$. Also, it is not necessary that the switch path $P \times_e Q$ uses all of the elements in $P_{[\to e]} \cup Q_{[e\to]}$.

Example 6.2. *We consider two examples of abstract networks. Figure 6.1a depicts the network $(E, \mathscr{P})$ with elements $E = \{a, b, c, d, e, f\}$ and paths $\mathscr{P} = \{(a, c, e), (a, c, d, e), (a, b, c, e), (a, b, d, e)\}$. We depict the elements as circles and paths as edges connecting them, but the presentation should not be confused with regular graphs. All paths start and end in the same networks, such that the network very much resembles classical networks. However, there is a difference. Element c is in the intersection of $P := (a, c, d, e) \cap Q := (a, b, c, e)$, but the only possible choice for the switch is $P \times_c Q = (a, b, d, e)$ and not (a, b, c, d, e) which does not exist in the network.*

As a second example we consider the abstract network depicted in Figure 6.1b with ground set $\{1, 2, 3, 4, a, b, c, d\}$ and the paths $\{(1, 2, 3, 4), (a, 2, b), (c, 3, d), (1, b), (1, d), (a, 4), (c, 4)\}$. The network features paths starting in more than one element and therefore resembles a network with multiple sources and sinks. Notice that both paths $(a, 2, b)$ and $(c, 3, d)$ cross with path $(1, 2, 3, 4)$ but there is no path starting with a and ending with d.

Abstract Flows. An **abstract flow** is an assignment $x_P : \mathscr{P} \to \mathbb{Q}_{\geq 0}$ that respects the capacities such that

$$\sum_{P \in \mathscr{P} : e \in P} x_P \leq u_e$$



(a) An abstract network. (b) Another abstract network.

Figure 6.1: Two abstract networks that are not equivalent to a classical network.

holds for each element $e \in E$. We define the Weighted Abstract Flow Problem with the objective of maximizing the (weighted) flow value.

Problem: Weighted Abstract Flow

Instance: An abstract network $(E, \mathscr{P})$, capacities u_e and path weights r_P.

Task: Find an abstract flow x that maximizes

$$\sum_{P \in \mathscr{P}} r_P \cdot x_P.$$

The special case with unit weights $r \equiv 1$ is the Abstract Flow Problem.

While the unweighted Abstract Flow Problem is a generalization of the Maximum Flow Problem (in the path formulation), the weighted version of the problem is a generalization of the Minimum Cost Flow Problem. The problem can be written as a linear program in the following way.

$$\begin{aligned} \max \quad & \sum_{P \in \mathscr{P}} r_P \cdot x_P, && && \text{(WAF)} \\ \text{s.t.} \quad & \sum_{P \in \mathcal{P}: e \in P} && \leq u_e && \text{for all } e \in E, \\ & x_P && \geq 0 && \text{for all } P \in \mathcal{P}. \end{aligned}$$

The dual of the linear program is the Weighted Abstract Cut Problem which is specified in the following LP formulation. We denote the special case with unit weights $r_P \equiv 1$ by Abstract Cut Problem.

$$\begin{aligned} \min \quad & \sum_{e \in E} y_e u_e, && && \text{(WAC)} \\ \text{s.t.} \quad & \sum_{e \in P} y_e && \geq r_P && \text{for all } P \in \mathscr{P}, \\ & y_e && \geq 0 && \text{for all } e \in E. \end{aligned}$$

Supermodularity. In contrast to the case of classical network flows the pair of linear and dual program is not totally dual integral. However, if we restrict the possible values for the weights, integer optimum solutions exist in case of integral weights and capacities. The path weights r_P are **supermodular** if

$$r_{P \times_e Q} + r_{Q \times_e P} \geq r_P + r_Q$$

holds for two crossing paths $P, Q \in \mathscr{P}$ with $e \in P \cap Q$. If the weights are supermodular, the abstract version of the Max-Flow=Min-Cut-Theorem holds.

Theorem 6.3 (Abstract Max-Flow=Min-Cut [Hof74]). *Let $(E, \mathscr{P})$ be an abstract network with integral capacities $u_e \in \mathbb{Z}^+$ on the elements and supermodular weights r_P on paths $P \in \mathscr{P}$. Then, the dual linear programs* (WAF) *and* (WAC) *have integer optimum solutions.*

Let $(E, \mathscr{P})$ be an abstract network. We consider all elements $t \in E$ that are either first or last elements on any path $P \in \mathscr{P}$ to be a **source** or **sink**, respectively. We denote both source elements and sink elements as **terminal** elements.

6.1.1 Maximum Abstract Flow

Abstract networks are typically huge in size. We consider the number of paths $|\mathscr{P}|$ as possibly exponential in the number of elements $m := |E|$ of the ground set. Therefore, the running time function of an efficient algorithm cannot include $|\mathscr{P}|$ as parameter. However, as an algorithm has to access paths somehow, we also allow the algorithm to use oracle calls. For a given set of elements, a call to the oracle returns (at most) one path that only uses elements contained in the set, possibly under certain additional restrictions. Under this conditions, a polynomial-time algorithm solving the Abstract Flow Problem is due to McCormick [McC96]. In the following we briefly describe the algorithm's basic ideas.

Oracles. Let $y \in \mathbb{Q}_+^m$ be a dual solution of (WAC). The oracle $\mathbf{O}_{\text{sep}}$ returns a **violating path** $P \in \mathscr{P}$ with $\sum_{e \in P} y_e < r_P$ together with the order $<_P$ of elements and the path weight r_P, or verifies that every path $P \in \mathscr{P}$ is dual feasible, i. e., $\sum_{e \in P} y_e \geq r_P$ holds for each path. The oracle solves the Separation Problem for (WAF) and by the equivalence of optimization and separation [GLS88] it can be used to solve the Weighted Abstract Flow Problem in strongly polynomial time. However, the approach does not lead to integral solutions even if the linear programs (WAF) and (WAC) are totally dual integral.

In the unweighted case with $r_P \equiv 1$ for all paths $P \in \mathscr{P}$ we can use a simpler variant of the oracle. Let $Q \subseteq E$ be a set of elements. The oracle $\mathbf{O}$ returns a violating path $P \in \mathscr{P}$ together with the order $<_P$ of its elements such that $P \subseteq Q$, or verifies that there is no path contained in Q.

Computing an Abstract Maximum Flow. McCormick's algorithm is a primal-dual generalization of the classical augmenting path maximum flow algorithm. The structure of an abstract path system is not as simple as in classical networks, such that using a labelling algorithm in some kind of residual network is not possible. Instead, the Abstract Maximum Flow Algorithm maintains a candidate set for an abstract minimum cut as solution of the dual problem. The algorithm then calls $\mathbf{O}$ to verify that the candidate set is in fact a feasible dual solution. If this is the case, the primal solution is a maximum abstract flow. Otherwise, the oracle returns a violating path. The returned violated paths are then combined to an augmenting structure which allows to improve the flow value.

Definition 6.4 (Augmenting Structure). Let $(E, \mathscr{P})$ be an abstract network and x be an abstract flow. An **augmenting structure** is a sequence $(Q_1^+, P_1^-, Q_2^+, P_2^-, \dots, Q_{k+1}^+)$ of k negative paths $P_i^- \in \mathscr{P}$ and $k+1$ positive paths $Q_i^+ \in \mathscr{P}$ for some $k \in \mathbb{N}_0$ satisfying the following properties.

There are k elements e_i^- such that $e_i^- \in Q_i^+ \cap P_i^-$ for $i = 1, \dots, k$ and also k elements e_i^+ such that $e_i + \in P_i^- \cap Q_{i+1}$ for $i = 1, \dots, k$. The elements satisfy $e_i^- <_{Q_i^+} e_i^+$ and $e_{i+1}^- <_{Q_{i+1}^+} e_i^+$. The initial and final parts of the paths overlap, i. e., $Q_{i\,[\to e_i^-]}^+ \subseteq P_{i\,[\to e_i^-]}^-$ for for the initial elements and $Q_{i+1\,[e_i^+ \to]}^+ \subseteq P_{i\,[e_i^+ \to]}^-$ holds for the final elements for $i = 1, \dots, k$.

The flow value on negative paths can be reduced, i. e., $x_{P_i} > 0$ for $i = 1, \dots, k$. On the middle part of positive paths flow can be augmented, i. e., the elements between e^-_{i+1} and e^+_i on paths Q^+_i and also the final part of Q^+_1 and the initial part of Q^+_{k+1} are not saturated. More formally, any element $e \in Q^+_{1\,[e^-_1 \to]}$, $e \in Q^+_{k+1\,[\to e^+_k]}$ or $e \in Q^+_i \setminus Q^+_{i\,[\to e^-_i]} \setminus Q^+_{i\,[e^+_{i-1} \to]}$ has strictly positive capacity left. ◁

The idea of an augmenting structure is to augment flow on positive paths and reduce flow on negative paths. Elements on initial and final parts of paths occur in both types of paths, such that their flow value is not changed in the process. However, on the middle elements of negative paths P^-_i, the flow value is reduced and on the middle parts of positive paths Q^+_i the flow value is increased. The elements e^+_i and e^-_i are contained in the candidate set L of a cut. Then, the augmenting structure proves that the candidate set was not feasible.

The framework of McCormick's algorithm is given in Algorithm 6.1. All details on the computation of the algorithm are omitted because we only use the algorithm as a sub-routine in the remainder of the chapter. The above test of dual feasibility is performed in step 2a). For a thorough description of the algorithm see [McC96].

Algorithm 6.1: Abstract Maximum Flow Algorithm

Input: Abstract network $(E, \mathscr{P})$, possibly an initial solution x_0.

Output: Abstract maximum flow x.

1. Initialize $x \coloneqq x_0$, if an initial solution is given. Otherwise initialize x as the zero flow.
2. While x is not optimal:
   a) Compute an augmenting structure AS. If no such structure exists, return x.
   b) Determine $\delta \in \mathbb{N}$ such that all paths in AS can be augmented/decreased by δ.
   c) For each path $P^+ \in AS$, set $x_{P^+} \coloneqq x_{P^+} + \delta$.
   d) For each path $Q^- \in AS$, set $x_{Q^-} \coloneqq x_{Q^-} - \delta$.

6.1.2 Abstract Flows over Time

Analogously to classical networks we add a temporal dimension to abstract flows and define the time expansion of abstract path systems. Let τ be non-negative transit times assigned to the elements $e \in E$. The transit time for an element denotes the time that is necessary for flow to pass along it. Let e and f with $e <_P f$ be two consecutive elements on path P. Flow travelling through e at time θ then continues travelling along f at time $\theta + \tau_e$.

In the following we introduce two variants of time expansion of an abstract network. The ground set of both path systems consists of temporal copies of the original elements (for a given time horizon). The first variant consists of copies of the paths in the original network starting at different times (therefore using different temporal copies of the elements). This construction, however, does not yield an abstract network because the switching property is not satisfied. We then allow flow to wait before flowing through elements and add more paths to reflect this. The resulting path system then is actually an abstract network.

Temporal Paths. Let $(E, \mathscr{P})$ be an abstract network with capacities $u_e \in \mathbb{Q}^+$ and **transit times** $\tau_e \in \mathbb{Z}^+$ on the elements $e \in E$ and let $T \in \mathbb{Z}^+$ be a **time horizon**. Similar to the time expansion of classical networks (cf. Definition 2.4) we create T copies of the elements which we identify with the intervals $]0,1],]2,2], \dots,]T-1, T]$ and define the **time-expanded ground set** as

$$E^T := \{e^\theta \mid e \in E, \theta \in \{1, 2, \dots, T\}\}.$$

Flow that is sent along a path P starting at time θ enters element $e \in P$ at time $\theta + \sum_{a \in P_{[\to e[}} \tau_a$. For any $P \in \mathscr{P}$ and $\theta \in \{1, 2, \dots, T\}$ we define the **temporal path** P_θ to consist of the element copies that flow travels through if sent into P at time θ, i. e.,

$$P_\theta := \left\{ e^\xi \in E^T \;\middle|\; e \in P, \theta + \textstyle\sum_{a \in P_{[\to e[}} \tau_a = \xi \right\}.$$

The order of the elements in P_θ is the same as in P thus we have $a^\xi <_{P_\theta} e^{\xi'}$ if and only if $a <_P e$. Finally, the time-expanded paths

$$\mathscr{P}_\theta^T := \left\{ P_\theta \;\middle|\; P \in \mathscr{P}, \theta \in \{1, 2, \dots, T\}, \theta + \textstyle\sum_{e \in P} \tau_e \leq T \right\}$$

are all temporal paths that arrive early enough, i. e., at the latest at time T.

Crossing of Temporal Paths. The abstract path system $(E^T, \mathscr{P}_\theta^T)$ is not an abstract network because it does not satisfy the switching properties (6.1) and (6.2). At first glance, this might be surprising because in classical network flows, time-expanded networks are indeed networks again by construction. However, the definition of $\mathscr{P}_\theta^T$ ignores the fact that it introduces additional intersections of paths at different times. It is possible that there is no path satisfying the switching axiom (6.1).

Example 6.5 (Time-expanded Path System that is not an Abstract Network). *Consider the ground set $E := \{s, a, b, t\}$ and set of paths $\mathscr{P} := \{P, Q, R, S\}$ with $P = (s, a, b, t)$, $Q = (s, b, a, t)$, $R = (s, a, t)$, and $S = (s, b, t)$. Observe that $(E, \mathscr{P})$ is an abstract network satisfying both switching properties* (6.1) *and* (6.2).

Assume now that the elements have unit transit times $\tau_e \equiv 1$. Then, the temporal paths P_0 and Q_1 intersect in the temporal copy b^2 of element b. To satisfy the requirements for abstract networks there must be a path contained in $P_{0[\to b^2]} \cup Q_{1[b^2 \to]} = \{s^0, a^1, b^2, a^3, t^4\}$ which is not the case.

Time-expanded Abstract Networks. We now extend the path system by adding further paths that solve the conflict shown in Example 6.5. The problem, which is introduced by simply taking all temporal copies as path system, is that the copies of elements may be used at the wrong time. This is the case even if there is a path satisfying the switching axiom in the underlying static abstract network. In the example, every path could serve as switch $P \times_b Q$ (if the path weights are set accordingly). If we want to use a temporal copy of P as possible path, after using copy b^2 we have to use t^3 as next element, however, the crossing of P_0 with Q_1 requires us to use element t^4. We will solve this problem by introducing waiting periods σ for each path that allow for flow to hold on for an arbitrary number of time steps before travelling through the next element.

Definition 6.6 (Time-expanded Abstract Network). Let $(E, \mathscr{P})$ be an abstract network with capacities $u_e \in \mathbb{Q}^+$ and transit times $\tau_e \in \mathbb{Z}^+$ and let $T \in \mathbb{Z}^+$ be a time horizon. We again use the time-expanded ground set $E^T := \{e^\theta \mid E \in E, \theta \in \{1, 2, \dots, T\}\}$. Let $P \in \mathscr{P}$ be a path and

$\sigma : P \to \{1, 2, \dots, T\}$ be waiting periods for each of the elements of P. Flow travelling along P that adheres to the waiting pattern σ waits σ_a time units before flowing through an element $a \in P$ and thus enters element $e \in P$ at time $\sum_{a \in P_{[\to e[}} (\sigma_a + \tau_a) + \sigma_e$. The **temporal path with intermediate waiting** P_σ is defined to be

$$P_\sigma \coloneqq \left\{ e^\xi \in E^T \;\middle|\; e \in P, \sum_{a \in P_{[\to e[}} (\sigma_a + \tau_a) + \sigma_e = \xi \right\}.$$

The order of the elements contained in P_σ is $a^\xi <_{P_\sigma} e^{\xi'}$ if and only if $a <_P e$. The set of time-expanded paths

$$\mathscr{P}_\sigma^T \coloneqq \left\{ P_\sigma \;\middle|\; P \in \mathscr{P}, \sigma \in \{1, 2, \dots, T\}^P, \sum_{e \in P} (\sigma_e + \tau_e) \le T \right\}$$

consists of all temporal paths with intermediate waiting that arrive within the time horizon T. Then $(E^T, \mathscr{P}_\sigma^T)$ forms the **time-expanded abstract network** of $(E, \mathscr{P})$. ◁

It can easily be seen that $\mathscr{P}_\theta^T \subseteq \mathscr{P}_\sigma^T$ because each path P_θ is equivalent to a path P_σ with $\sigma \coloneqq (\theta, 0, \dots, 0)$. We now show that the time-expanded abstract network is actually an abstract network if the underlying static network satisfies the following property. Let $P, Q \in \mathscr{P}$ be two paths with a common element $e \in P \cap Q$ and $R \coloneqq P \times_e Q$. An abstract network **preserves the order** if for any pair of elements $a, b \in R \cap P_{[\to e]}$ with $a <_P b$ or $a, b \in R \setminus P_{[\to e]}$ with $a <_Q b$, also $a <_R b$ holds.

Lemma 6.7. *Let $(E, \mathscr{P})$ be an abstract network that preserves the order on the paths. Then the path system $\mathscr{P}_\sigma^T$ is an abstract network.*

Proof. Let P_π and Q_ρ be two paths in $\mathscr{P}_\sigma^T$ with waiting patterns π and ρ that intersect at element e^θ. Then the underlying paths cross in the static network at element e and there is a path $R \coloneqq P \times_e Q$. Because the network preserves the order of the paths, R contains elements of P and Q in the same order but probably leaves out some of the elements. We can define waiting periods before the elements on R according to the transit times of the skipped elements on P and Q such that the temporal copies of elements used by R, P and Q, respectively, are the same. Let $R = e_1 <_R e_2 <_R \cdots <_R e_k$ be the elements of R in their respective order. We then define σ appropriately such that it skips the omitted elements by waiting.

$$\sigma(e_1) \coloneqq \begin{cases} \sum_{a \in P_{[\to e_1[}} (\tau_a + \pi_a) + \pi_{e_1} & \text{if } e_1 \in P, \\ \sum_{a \in Q_{[\to e_1[}} (\tau_a + \rho_a) + \pi_{e_1} & \text{if } e_1 \in Q, \end{cases}$$

and for $i = 2, \dots, k$

$$\sigma(e_i) \coloneqq \begin{cases} \sum_{\{a \in P \mid e_{i-1} <_P a <_P e_i\}} (\tau_a + \pi_a) + \pi_{e_i} & \text{if } e_i \in P, \\ \sum_{P_{]e_{i-1} \to]}} (\tau_a + \pi_a) + \sum_{Q_{[\to e_i[}} (\tau_a + \rho_a) + \rho_{e_i} & \text{if } e_{i-1} \in P, e_i \in Q, \\ \sum_{\{a \in Q \mid a_{i-1} <_Q a <_Q a_i\}} (\tau_a + \rho_a) + \rho_{e_i} & \text{if } e_{i-1}, e_i \in Q \in P. \end{cases}$$

Due to the precondition that the underlying static network preserves the order of elements σ is well defined and $R_\sigma \in \mathscr{P}_\sigma^T$ is a path in the time-expanded abstract path system that satisfies the switching property for $P_\pi \times_{e_\theta} Q_\rho$. Analogously it can be shown that there is a path $Q_\rho \times_{e^\theta} P_\pi$ contained in $\mathscr{P}_\sigma^T$. □

Abstract Maximum Flow over Time. An **abstract flow over time** is an assignment $x : \mathscr{P}_\sigma^T \to \mathbb{R}_{\geq 0}$ that assigns a flow value to each path such that

$$\sum_{\left\{P_\sigma \in \mathscr{P}_\sigma^T \middle| e^\theta \in P_\sigma\right\}} x_{P_\sigma} \leq u_e$$

holds for all temporal copies of elements e^θ in the time-expanded ground set E^T. Hence, a flow over time respects the capacities of elements at each point in time. The value $|x|$ of a flow over time is defined as $\sum_{\{P_\sigma \in \mathscr{P}_\sigma^T\}} x_{P_\sigma}$. Analogously to the case of classical flows over time we can define an **abstract cut over time** as a subset $C \subseteq E^T$ of the time-expanded ground set such that

$$P_\sigma \cap C \neq \varnothing$$

for all paths $P_\sigma \in \mathscr{P}_\sigma^T$, i. e., every path in the time-expanded network is covered by the cut. The **capacity** of a cut over time is $\sum_{\{e^\theta \in C\}} u_e$.

Problem:	Abstract Maximum Flow over Time
Instance:	An abstract network $(E, \mathscr{P})$, capacities $u_e \in \mathbb{Q}^+$, path weights $r_P \in \mathbb{Z}^+$ and a time horizon $T \in \mathbb{Z}^+$.
Task:	Find an abstract flow over time x maximizing the flow value $\|x\|$.

Theorem 6.8 (Abstract-Max-Flow=Min-Cut-over-Time-Theorem). *Let $(E, \mathscr{P})$ be an abstract network, τ_e transit times for elements and T a time horizon. Then, the values of an abstract maximum flow over time and the capacity of a minimum abstract cut over time are equal. There exists a maximum flow that does not use waiting, i. e., $x_P = 0$ for all $P \in \mathscr{P}_\sigma^T \setminus \mathscr{P}_\theta^T$.*

The theorem is joint work with Jannik Matuschke and Britta Peis. A proof can be found in [Mat13] and also in [KMP14].

6.1.3 Structural Properties

If the paths in an abstract network satisfy certain constraints, we can show that the switching operation preserves the order of the elements. We start by showing that we can always choose the path resulting from an application of $\times_\cdot$ in such a way that the two subpaths used for its construction are not mixed.

Lemma 6.9. *Let $P, Q \in \mathscr{P}$, $e \in P \cap Q$, then there is a path $R \subseteq P_{[\to e]} \cup Q_{[e\to]}$ such that $a \in R \cap P_{[\to e]}$ and $b \in R \setminus P_{[\to e]}$ implies $a <_R b$.*

Proof. Let $P, Q \in \mathscr{P}$ be two paths with common element $e \in P \cap Q$. Let R to be a path contained in $P_{[\to e]} \cup Q_{[e\to]}$ such that $\left|R \setminus P_{[\to e]}\right|$ is minimal. Assume now, that there is $a \in R \cap P_{[\to e]}$ and $b \in R \setminus P_{[\to e]}$ with $b <_R a$. Let $R' := P \times_a R$. Observe that $R' \subset P_{[\to e]} \cup Q_{[e\to]}$ and $R' \setminus P_{[\to e]} \subset R \setminus P_{[\to e]}$ as $b \notin R'$, which is a contradiction to the choice of R. □

We call a sub-path $Q \subset P$ **order preserving** if Q is a strict subset of P and the order $<_Q$ is identical to $<_P$ on the elements that occur in both paths.

Lemma 6.10. *Let $\mathcal{P}$ be a set of paths such that no sub-path preserves the order. Then, there are no paths $P, Q \in \mathcal{P}$ such that $Q \subset P$.*

Proof. By contradiction assume there are two paths $P, Q \in \mathcal{P}$ with $Q \subset P$. Let P^* be such that $|P^*|$ is minimal among all possible choices of such a P.

For $Q \subset P^*$ define $b(Q) \in Q$ to be the maximal element with respect to $<_Q$ such that $p <_{P^*} b(Q)$ for all $p \in Q_{[\to b(Q)[}$, i. e., until element $b(Q)$ the order of Q is identical to that of P. Because no sub-path preserves order, $b(Q)$ cannot be the last element of Q. So let $a(Q) \in Q$ be the successor of $b(Q)$ in Q. Notice, that this implies $a <_{P^*} b$ by definition of $b(Q)$. Among all paths $Q \subset P^*$, choose Q^* such that $b^* := b(Q^*)$ is maximal with respect to $<_{P^*}$, i. e., Q^* is the path that has the longest common part with P^*. Let $a^* := a(Q^*)$.

Now let $R := Q^* \times_{b^*} P^*$. Notice that $a^* \notin R$, because $a^* >_{Q^*} b^*$, and therefore $R \subset P^*$. The order $<_R$ is identical to $<_{Q^*}$ on the initial part $Q^*_{[\to b^*]} \subset R$ of R. To see this, we have to show that for all $c, d \in R \cap Q^*_{[\to b^*]}$ with $c <_{Q^*} d$ also $c <_R d$ holds.

To see this, assume there are c and d with $c <_{Q^*} d$ and $d <_R c$. Let $R' := R \times_d Q^*$. Notice that $c \notin R'$. By Lemma 6.9 we choose R such that $R_{[\to d]} \subset Q^*$ without loss of generality. Thus $R' \subset Q^* \subset P^*$ which contradicts the choice of P^*.

By definition of $b(Q^*)$, the order $<_{Q^*}$ is identical to $<_{P^*}$ on $Q^*_{[\to b^*[}$ and thus $<_R$ is identical to $<_{P^*}$ on $Q^*_{[\to b^*]} \subset R$. This implies that $a(R)$ and $b(R)$ cannot be both contained in $Q^*_{[\to b^*[}$. Thus, $a(R) \in P^*_{[b^* \to]}$, which implies that $b(R) \in P^*_{]b^* \to]}$ as $a(R) <_{P^*} b(R)$. However, this means $b(R) >_{P^*} b^*$ contradicting the choice of Q^* maximizing b^*. □

Lemma 6.11. *Let $(E, \mathcal{P})$ be an abstract network such that no sub-path preserves the order and let $R \coloneqq P \times_e Q$.*

If $a, b \in R \cap P_{[\to e]}$ and $a <_P b$, then $a <_R b$. If $a, b \in R \setminus P_{[\to e]}$ and $a <_Q b$, then $a <_R b$, i. e., the network preserves order.

Proof. If $a \in P \times_e Q \cap P_{[\to e]}$ and $b \in P \times_e Q \setminus P_{[\to e]}$, then by Lemma 6.9 we can assume without loss of generality that $a <_{P \times_e Q} b$.

For the first statement, we assume by contradiction that there exist $a, b \in P_{[\to e]} \cap R$ with $a <_P b$ but $b <_R a$. Then, by our above assumption, there is no $c \in R \setminus P_{[\to e[}$ with $c <_R a$. Thus $R_{[a\to]} \subseteq P$ and $R \times_b P \subset P$, which is a contradiction to Lemma 6.11.

For the second statement, we assume by contradiction that there exist $a, b \in R \setminus P_{[\to e]}$ with $a <_Q b$ but $b <_R a$. Then, by assumption, there is no $c \in R \cap P_{[\to e]}$ with $c >_R b$. Thus $R_{[a\to]} \subseteq Q$ and $Q \times_a R \subset Q$, again contradicting Lemma 6.11. □

Corollary 6.12. *Let $(E, \mathcal{P})$ be an abstract network, such that no sub-path preserves the order. Then the path system $\mathcal{P}^*_T$ is an abstract network.*

Proof. This directly follows from Lemma 6.11 and Lemma 6.7. □

6.2 Lexicographic Maximum Abstract Flows

We derive a similar notation for lexicographically maximum flows in the abstract setting as for the classical flows in Definition 1.10. Using this notation we show existence of *lexicographically maximum abstract flows* and present an algorithm to compute such flows. However, in contrast to the

classical case the lexicographical order of the terminals cannot be arbitrary but has to respect certain constraints if more than one terminal node is contained in a path.

Lexicographical Order of Abstract Flows. For a terminal element e we define the outflow (inflow) to be the amount of flow on paths that start (end) with e. Let $(E, \mathscr{P})$ be an abstract network and let x be an abstract flow. The **outflow** $|x|_s^+$ of a source element s is then

$$|x|_s^+ := \sum_{P \in \delta_s^+} x_P,$$

and the **inflow** $|x|_t^-$ into a sink element t is defined to be

$$|x|_t^- := \sum_{P \in \delta_t^-} x_P.$$

Definition 6.13 (Lexicographically Maximum Abstract Flow)**.** Let $(E, \mathscr{P})$ be an abstract network and $s_1, s_2, \dots, s_k$ be an order of the source elements. Let x^1 and x^2 be two maximum abstract flows in $(E, \mathscr{P})$. We say that x^1 is **lexicographically larger** than x^2 if there exists either an $\ell \in \{0, 1, \dots, k-1\}$ such that $|x^1|_{s_{\ell+1}}^+ > |x^2|_{s_{\ell+1}}^+$ and the outflow values $|x^1|_{s_i}^+ = |x^2|_{s_i}^+$ are equal for $i = 1, 2, \dots, \ell$, or all outflows $|x^1|_{s_i}^+ = |x^2|_{s_i}^+$ are equal for $i = 1, \dots, k$. We then also write $x^1 \geq_L x^2$.

If $t_1, t_2, \dots, t_k$ is an order of the sinks we have $x^1 \geq_L x^2$ if either $|x^1|_{s_{\ell+1}}^- > |x^2|_{s_{\ell+1}}^-$ and $|x^1|_{s_i}^- = |x^2|_{s_i}^-$ holds for some $\ell \in \{0, 1, \dots, k-1\}$ and $i = 1, \dots, \ell$, or $|x^1|_{s_i}^- = |x^2|_{s_i}^-$ holds for all $i = 1, 2, \dots, k$.

A **lexicographically maximum abstract flow** x^* is maximum among all of those flows, i. e., $x^* \geq_L x$ for all feasible abstract flows x. ◁

For classical flows Minieka [Min73] and Megiddo [Meg74] show that lexicographically maximum flows exist for any order of the sources and sinks. In the abstract model the more general setting allows that sources may appear in a different order on several paths. We will therefore restrict the possible order. We call a sequence of terminals **compatible** if the terminal elements respect their rank if more than one terminal of the same type appears on a path. For sources $s_1, s_2, \dots, s_k$ we require that sources with higher rank appear right of sources with lower rank, e. g.,

$$P \in \mathscr{P}, s_i \neq s_j \in P : j < i \implies s_i \leq_P s_j. \tag{6.3}$$

In contrast to sources, a compatible sequence of sinks $t_1, t_2, \dots, t_k$ has to satisfy

$$P \in \mathscr{P}, t_i \neq t_j \in P : i < j \implies t_i \leq_P t_j, \tag{6.4}$$

e. g., sinks with higher rank appear earlier.

For a given compatible sequence of sources $s_1, s_2, \dots, s_k$ we define abstract networks $(E, \mathscr{P}_s^i)$ with increasing subsets of paths $\mathscr{P}_s^i \subseteq \mathscr{P}$ for $i = 1, 2, \dots, k$. We start with $\mathscr{P}_s^0 := \emptyset$ and then recursively define

$$\mathscr{P}_s^i := \mathscr{P}_s^{i-1} \cup \{P \in \mathscr{P} \mid s_i = \text{first}(P)\}.$$

Correspondingly, for a compatible sequence of sinks $t_1, t_2, \dots, t_k$ we define $\mathscr{P}_t^0 := \emptyset$ and

$$\mathscr{P}_t^i := \mathscr{P}_t^{i-1} \cup \{P \in \mathscr{P} \mid t_i = \text{last}(P)\}.$$

Each of the abstract path systems $(E, \mathscr{P}_s^i)$ and $(E, \mathscr{P}_t^i)$ for $i = 1, 2, \dots, k$ contains the paths starting and ending in the first i sources and sinks, respectively. All of those systems are again abstract networks and satisfy the switching axiom (6.1).

Lemma 6.14. *Let $(E, \mathscr{P})$ be an abstract network and $s_1, s_2, \dots, s_k$ a compatible sequence of sources. Then for each $i = 1, \dots, k$ the abstract path system $(E, \mathscr{P}_s^i)$ is an abstract network. The same is true for a compatible sequence $t_1, t_2, \dots, t_k$ of sinks and the systems $(E, \mathscr{P}_t^i)$.*

Proof. Let P and Q be two crossing paths in $\mathscr{P}_s^i$ with a common element $e \in P \cap Q$. We have to show that there are paths $R \subseteq P_{[\to e]} \cup Q_{[e\to]}$ and $R' \subseteq Q_{[\to e]} \cup P_{[e\to]}$ in $\mathscr{P}_s^i$.

Because $(E, \mathscr{P})$ is an abstract network, there is a switching path $R = P \times_e Q$ in $\mathscr{P}$. Then, $R \in \mathscr{P}_s^i$ if and only if the first element of R is one of the source elements $\{s_1, s_2, \dots, s_i\}$. The source element $\operatorname{first}(R)$ is one of the elements $P_{[\to e]} \cup Q_{[e\to]}$ by definition of the switching property. If the paths satisfy $\operatorname{first}(R) = \operatorname{first}(P)$, nothing remains to show. Otherwise, $\operatorname{first}(R)$ is contained in P or Q, but not the first element and $\operatorname{first}(R) = s_j$ for some $j < i$ by 6.3. The same holds for the path $R' = Q \times_e P$.

For a given compatible sequence $t_1, t_2, \dots, t_k$ of sink elements the same argumentation holds with $\operatorname{last}(R)$ instead of $\operatorname{first}(R)$. □

Lemma 6.15. *Let $s_1, s_2, \dots, s_k$ be a compatible sequence of sources and let x^i be a maximum abstract flow in the network $\mathscr{P}_s^i$. If Algorithm 6.1 is executed with the abstract network $\mathscr{P}_s^{i+1}$ and initial flow x^i as input, during the execution the flow value $|x|_{s_j}^+$ does not decrease for $j = 1, \dots, i$.*

Let $t_1, t_2, \dots, t_k$ be a compatible sequence of sources. Then during the execution of the algorithm on $\mathscr{P}_s^{i+1}$ the flow value $|x|_{t_j}^+$ does not decrease for $j = 1, 2, \dots, i$.

Proof. We prove the lemma by contradiction and assume there is a source s_j with $j < i + 1$ whose outflow is reduced by an augmenting structure AS. If there are more of those sources we take the most important one with respect to the given compatible sequence $s_1, s_2, \dots, s_k$.

Because the outflow value $|x|_{s_j}^+$ is reduced by the augmenting structure there must be a path P_ℓ^- on which flow is reduced with $\operatorname{first}(P_\ell^-) = s_j$, but this s_j is not the first element of any path on which flow is increased, especially $s_j \neq \operatorname{first}(Q_\ell^+)$. We denote the source of this path by $s_j' := \operatorname{first}(Q_\ell^+)$. Since in any augmenting structure $Q_\ell^+ \subseteq P_\ell^-$ on the beginning of the paths, there are two source elements s_j and s_j' on P_ℓ^- with $s_j <_{P_\ell^-} s_j'$. Hence, the source element s_j' has a strictly higher rank in the source order. The outflow $|x|_{s_j'}^+$ cannot increase because the initial flow was lexicographically maximum on the source elements $s_1, \dots, s_i$ and thus there must be another path $P_{\ell_2}^-$ in the augmenting structure starting at s_j'. By the same argumentation there is another source element $s_j'' := \operatorname{first}(Q_{\ell_2}^+)$ right of s_j' on P_{ℓ_2} which again has a higher rank in the source order. The procedure cannot be continued infinitely due to the finite amount of sources and thus leads to a source element with higher rank whose outflow increases, contradicting the assumption.

To show the result for a given sink sequence, we use the same argumentation for a given compatible sequence $t_1, t_2, \dots, t_k$ of sinks. □

Using the result of the lemma we can state the algorithm that computes a lexicographically maximum flow, either for a given sequence of sources or sinks. The algorithm successively computes abstract flows in increasing networks containing an increasing number of terminal elements. Consider the following Algorithm 6.2 formalizing the procedure.

Algorithm 6.2: Abstract Lexicographically Maximum Flow

Input: An abstract network $(E, \mathscr{P})$ with capacities u_e, weights $r_P \equiv 1$ and a compatible sequence of sources $\{s_1, s_2, \dots, s_k\}$ or sinks $\{t_1, t_2, \dots, t_k\}$.

Output: A lexicographic maximal flow x.

1. Set $i = 0$ and initialize $x^0 \equiv 0$ as the zero flow on all paths.
2. Set $i := i + 1$ and define the network $(E, \mathscr{P}_s^i)$ (or $(E, \mathscr{P}_t^i)$, if the input terminal sequence consists of sinks).
3. Compute a flow x^i using Algorithm 6.1 in $(E, \mathscr{P}_s^i)$ starting with solution x^{i-1}.
4. If $i = k$ return x^k, otherwise continue with 2.

Theorem 6.16. *Algorithm 6.2 computes a lexicographically maximum abstract flow in* $(E, \mathscr{P})$.

Proof. The algorithm works for source and sink element sequences. We prove the result for source elements, but the same argumentation holds for sinks.

We show that flow x^i is a lexicographically maximum abstract flow in $(E, \mathscr{P}_s^i)$ after step 3 in the algorithm. The statement of the theorem then follows for $i = k$. The statement is obviously true for the first iteration that computes a maximum abstract flow x^1 in $(E, \mathscr{P}_s^1)$ as the network only contains one source element.

Assume by induction that x^i is a lexicographically maximum abstract flow in $(E, \mathscr{P}_s^i)$. During the computation of x^{i+1} the inflow of no source element is reduced due to Lemma 6.15 and the flow is maximum. Assume now that x^{i+1} is not a lexicographically maximum abstract flow in the abstract network $(E, \mathscr{P}_s^{i+1})$. Then there is a flow x' that sends more flow out of the source s_j for some $j \in \{1, 2, \dots, i\}$. We define the restricted flow $\hat{x}$ by setting $\hat{x}_P := x'_P$ for each path $P \in \mathscr{P}^i$. The outflow of source element s_j is the same for x' and $\hat{x}$ and $\hat{x}$ is a feasible abstract flow in $(E, \mathscr{P}^i)$ that sends more flow out of s_j than does x^i. This is a contradiction to x^i being lexicographically maximum. □

Theorem 6.17. *Let* $(E, \mathscr{P})$ *be an abstract network with integral arc capacities* $u_e \in \mathbb{Z}^+$ *and constant path weights* $r_P \equiv 1$. *Let a compatible sequence of either sources* $s_1, s_2, \dots, s_k$ *or sinks* $t_1, t_2, \dots, t_k$ *be given.*

Then there exists an abstract maximum flow x *in* $(E, \mathscr{P})$ *that is lexicographically maximum on the given sequence. This flow can be computed in polynomial time if a separation oracle is given.*

Proof. The flow can be computed using Algorithm 6.2. The algorithm performs k iterations in each of which Algorithm 6.1 is called once as a subroutine. The algorithm can be implemented such that it has a polynomial runtime in $|E|$ by using a separation oracle and capacity scaling techniques [McC96]. □

Directions for Further Research. For classical network flows it is possible to compute flows that are lexicographically maximum not only for a sequence of sources or sinks, but also for arbitrary sequences of both types of terminals. This has first been shown by Minieka [Min73] and independently

by Megiddo [Meg74]. Because the existing algorithms have to be applied in the time expanded network, Hoppe and Tardos [HT00] present a new algorithm that is specialized for dynamic networks.

Unfortunately, these approaches cannot be applied to compute an abstract lexicographically maximum flow, assuming an extended notion of a compatible sequence of terminals of both type. For two given classical lexicographically maximum flows, one on the sources and the other on the sinks, Minieka's algorithm computes a flow having both patterns. This is achieved by changing the flow along cycles such that the flow values at the terminal nodes are not changed. Hoppe's extension uses an extended network with additional super terminals. In this network, flow is not only sent from sources to sinks but also shifted between sources using a minimum cost flow in the residual graph. Both approaches are based on the fact that augmenting path algorithms can be used to send flow between terminals *without changing the actual flow value*.

Hence, a more advanced algorithm than McCormick's Abstract Maximum Flow Algorithm is necessary to compute lexicographically maximum flows for sequences of both types of terminals. This is because Algorithm 6.1 checks dual feasibility by finding a violated constraint and computes a possible augmenting structure that improves the primal solution. Therefore the algorithm cannot be used to shift flow between different terminals without increasing the flow value.

6.3 Earliest Arrival Abstract Flows

In this section we consider a generalization of the earliest arrival flow problem in the abstract setting. First, we want to compute maximal abstract flows and show existence of such flows by applying the Abstract Lexicographically Maximum Flow Algorithm 6.2. If we additionally add supplies and demands, we show that value-approximate solutions exist.

6.3.1 Existence and Lexicographic Maximum Flow Algorithm

Let $(E, \mathscr{P})$ be an abstract path system with capacities $u_e \in \mathbb{R}^+$ and transit times $\tau_e \in \mathbb{R}_{\geq 0}$ on the elements and a time horizon $T \in \mathbb{N}$. Let x be an abstract flow in the time-expanded path system $(E^T, \mathscr{P}_\sigma^T)$. For any $\theta \in \{1, 2, \dots, T\}$ let $\mathscr{P}^\sigma := \{P \in \mathscr{P} \mid \sum_{e \in P}(\sigma_e + \tau_e) \leq \theta\}$ be the set of temporal paths that require not more than θ time units to travel through them. Then,

$$|x|_\theta := \sum_{P \in \mathscr{P}^\theta} x_P$$

defines the flow value that arrives until time θ. We call x an **abstract earliest arrival flow** if it maximizes the flow value at all points in time $\theta = 1, \dots, T$, i. e., for all θ it holds that $|x|_\theta \geq |x^\theta|$ where $|x^\theta|$ is the value of a maximum abstract flow over time with time horizon θ.

Problem:	Abstract Earliest Arrival Flow
Instance:	An abstract network $(E, \mathscr{P})$, an integral time horizon $T \in \mathbb{N}$ for $\theta \in \{1, 2, \dots, T\}$.
Task:	Compute an abstract flow over time that maximizes $\|x\|_\theta$ simultaneously for all points in time θ.

Foregoing Considerations. We want to use Algorithm 6.1 in the time-expanded abstract network to compute an abstract earliest arrival flow. By Lemma 6.7 we know that the time-expanded set system only is an actual abstract network if the paths preserve order. However, without loss of generality we can restrict ourselves to networks having this property. To see this, consider two paths P and Q in an abstract network such that $Q \subset P$. If P is order preserving, then P is not necessary in an optimal solution of the Abstract Earliest Arrival Flow Problem. If we allow flow to have waiting periods, any solution using P can be transformed such that Q is used instead. Thus, the value of the flow at a given point in time does not change and P is not used.

By Lemmas 6.9 and 6.11 we then can assume without loss of generality that there do not exist strict sub-paths $Q \subset P$ and that the order of elements is preserved on all paths. Hence, we can solve the Abstract Earliest Arrival Flow Problem for arbitrary abstract networks.

Theorem 6.18. *Let $(E, \mathscr{P})$ be an abstract network and $T \in \mathbb{N}$ be a time horizon. Then, there exists an abstract earliest arrival flow x in $(E, \mathscr{P})$ that may require waiting in intermediate nodes. The flow x can be computed by an application of Algorithm 6.2 in the time-expanded network $(E, \mathscr{P}_\sigma^T)$.*

Proof. We solve the Abstract Maximum Earliest Arrival Flow Problem for an instance $(E, \mathscr{P})$ with an application of the Abstract Lexicographically Maximum Flow Algorithm 6.2. Let T be a sufficiently large time horizon and $(E^T, \mathscr{P}_\sigma^T)$ be the time-expanded abstract network.

For each sink element $t \in E$ in the original abstract network we now get a sink for each point in time where a temporal copy arrives in the time-expanded network. Let $t = \text{last}(P)$ be a sink on P. Then for each $\theta \in [\tau(P), T[$ the elements (e, θ) are sinks for some temporal paths. We define an order $(e, 0), \dots, (e, 1), \dots, (e, T)$ on these sinks to compute a lexicographic maximal abstract maximum flow in the time-expanded abstract network.

There is no temporal path containing two sinks arriving at time θ due to the above lemma. Let P_σ, Q_σ be temporal paths with $\sum_{a \in P} \sigma(a) + \tau(a) = \sum_{a \in Q} \sigma(a) + \tau(a)$ having different last elements $\text{last}(P) \neq \text{last}(Q)$. Then $R \coloneqq P \times_Q last(P)$ would be a sub-path of P violating Lemma 6.10 □

Directions for Further Research. To compute abstract earliest arrival flows, Algorithm 6.2 is executed on a time-expanded abstract network that contains waiting. In the setting of classical network flows such a computation can be avoided. Minieka [Min73] shows that using the Successive Shortest Path Algorithm in the original network and sending flow along the generalized path decomposition temporally repeated leads to an earliest arrival flow.

This approach does not require waiting in intermediate nodes and sends flow only along temporal copies of the original paths. A similar algorithm to the Successive Shortest Path Algorithm is due to Martens and McCormick [MM08]. Their algorithm computes a maximum weighted abstract flow by using augmenting structures of decreasing total reward. By total reward we mean $\sum_{i=1}^{k+1} r_{Q_i^+} - \sum_{i=1}^{k} r_{P_i^-}$ for P_i^- and Q_i^+ being the paths contained in an augmenting structure. This resembles sending shortest paths just in a maximization setting; the shortest path has the most reward.

It is an interesting question whether the augmenting structures that are used by Martens and McCormick's algorithm can be used temporally repeated over the whole time horizon.

6.3.2 Abstract Flows over Time with Supplies and Demands

Classical flows with multiple sinks do not allow for earliest arrival flows, but the flow value can be approximated as we have seen in Section 4.3. We extend the notion of abstract earliest arrival flows

by adding supplies and demands for source elements and sink elements, respectively.

We define $S^+ := \{e \in E \mid \exists P \in \mathcal{P} : e = \text{first}(P)\}$ to be the set of source elements and similarly define $S^- := \{e \in E \mid \exists P \in \mathcal{P} : e = \text{last}(P)\}$ to be the set of sink elements. Let $b^+ : E \to \mathbb{Q}_{\geq 0}$ be supplies for source elements and let $b^- : E \to \mathbb{Q}_{\geq 0}$ be demands for sink elements. An abstract flow in $(E, \mathcal{P})$ satisfies given supplies and demands if

$$\sum_{P \in \mathcal{P}:\text{first}(P)=e} x_P = b_e^+ \qquad \text{and} \qquad \sum_{P \in \mathcal{P}:\text{last}(P)=e} x_P = b_e^-$$

holds for source and sink elements, respectively. Observe that it is possible that an element is both, a source element and a sink element.

Example 6.19. *Consider the abstract network with elements $E := \{d, e, f\}$ and paths $\mathcal{P} := \{P_1 = (d, e), P_2 = (e, f), Q = (d, f), R = (e)\}$. It is easy to verify that $(E, \mathcal{P})$ actually is an abstract network. Because $\text{last}(P_1) = e = \text{first}(P_2)$ element e is both source and sink. Define balances $b_d^+ := b_e^+ := 1$ and supplies $b_e^- := b_f^- := 1$. The transit times are set to $\tau_d := 1, \tau_e := 0, \tau_f := 1$ and the capacities are unit capacities $u \equiv 1$. It is possible to satisfy the supplies and demands within a time horizon of 2 by sending one unit of flow each on P_1 and P_2.*

For given supplies and demands we are interested in finding a flow over time that satisfies as much of the balances as early as possible.

Problem:	Abstract Earliest Arrival Transshipment
Instance:	An abstract network $(E, \mathcal{P})$, an integral time horizon $T \in \mathbb{N}$ for $\theta \in \{1, 2, \dots, T\}$, supplies b^+ and demands b^- for source elements and sink elements.
Task:	Compute an abstract flow over time that maximizes $\lvert x \rvert_\theta$ for all points in time θ.

Restrictions and Time Expansion. If we want to compute abstract flows in time-expanded networks in the presence of supplies and demands we have to impose two additional restrictions. First, we explicitly require that sub-paths preserve order. In Section 6.3.1 we assumed without loss of generality that in an abstract network $(E, \mathcal{P})$ there are no paths that are strictly contained in longer paths. Under this assumption Lemma 6.7 guarantees that the time-expanded network can be built. However, the argumentation does no longer hold. If a path $Q \subset P$ is strictly contained in P it is required to use both paths to satisfy the demands at $\text{last}(Q)$ and $\text{last}(P)$.

The second restriction is required because we have to extend our notion of time-expanded abstract networks by additional super elements. If we want to make sure that $\sum_{P \in \mathcal{P}:\text{last}(P)=e} x_P = b_e^-$ holds for an element e in the time-expanded network, we have to make sure that no more flow is sent on temporal copies of such paths. Let $e \in E$ a sink element. We can enforce the constraint for e by extending all paths $P \in \mathcal{P}$ with $\text{last}(P) = e$ by a super sink element t_e^*. Setting the capacity to $u_{t_e^*} := b_e^-$ ensures that no flow in an abstract time-expanded network can exceed the demands. We do the same by adding super source elements at the beginning of every path.

Definition 6.20. Let $(E, \mathscr{P})$ be an abstract network. Let $P, Q \in \mathscr{P}$ two paths with a common element $e \in P \cap Q$. Then the network is **terminal respecting**, if all paths $R \in P_{[\to e]} \cup Q_{[e\to]}$ satisfy

$$\text{first}(R) = \text{first}(P) \qquad \text{and} \qquad \text{last}(R) = \text{last}(Q).$$

◁

Our goal is to define an abstract variant of the General Greedy Value-approximate Earliest Arrival Flow Algorithm 4.2. The algorithm sends flow in increasing time-expanded networks and avoids reduction of flow sent in earlier time steps by removing backward arcs in the residual network. Thus, we have to specify a mechanism that avoids reduction of flow on abstract paths arriving in earlier time steps. We do this by including another element into paths before the super terminals.

Let e^θ be an element copy for time θ in the time-expanded network. We will extend all temporal paths P_σ that contain e^θ as last element by an element e_c^θ that counts the amount of flow arriving at element e at time θ. We will then use an algorithm given by Martens and McCormick [MM08] to fix the flow value on these elements. We now formalize the definition of the time-expanded abstract network with paths extended by counting elements and super terminals.

Definition 6.21 (Time-expanded Network with Extended Paths)**.** Let $(E, \mathscr{P})$ be an abstract network with source elements S^+ and sink elements S^-. Let $(E^T, \mathscr{P}_\sigma^T)$ be the time-expand network for some time horizon $T \in \mathbb{N}$.

We introduce additional super source elements s^*, super sink elements t^* and **counting elements** t_c. Combining all those elements with the original node copies, the time-expanded ground set is defined as

$$\widetilde{E}^T := E^T \cup \{s_e^* \mid e \in S^+\} \cup \{t_e^* \mid e \in S^-\} \cup \{e_c^\theta \mid e \in S^-, \theta \in \{1, 2, \dots, T\}\}.$$

We extend each original temporal path $P_\sigma = (e_1, e_2, \dots, e_n) \in \mathscr{P}_\sigma^T$ by the corresponding super terminals and a counting element. Let $s = e_1 = \text{first}(P_\sigma)$ and $t^\theta = e_n = \text{last}(P_\sigma)$ be the first and last element of P_σ, respectively. The **extended path**

$$\widetilde{P}_\sigma := (s^*, e_1, e_2, e_3, \dots, e_n, t_c^\theta, t^*)$$

contains three more elements, the super source as new first element, the super sink as last element and the counting element left of the super sink. The set of paths $\widetilde{\mathscr{P}}_\sigma^T$ consists of all extended paths $\widetilde{P}_\sigma$.

◁

Lemma 6.22. *Let $(E, \mathscr{P})$ be a terminal respecting abstract network preserving the order. Let τ_e be transit times for the elements and b^+ supplies for source elements and b^- demands for sink elements and $T \in \mathbb{N}$ a time horizon. Then, $(\widetilde{E}^T, \widetilde{\mathscr{P}}_\sigma^T)$ is an abstract network.*

Proof. Because $(E, \mathscr{P})$ preserves the order $(E^T, \mathscr{P}_\sigma^T)$ is an abstract network by Lemma 6.7. We have to show that the switching axioms (6.1) and (6.2) are satisfied by $(\widetilde{E}^T, \widetilde{\mathscr{P}}_\sigma^T)$. Consider two paths $\widetilde{P}_\sigma$ and $\widetilde{Q}_\sigma$ crossing in the element $e^\theta \in \widetilde{P}_\sigma \cap \widetilde{Q}_\sigma$. The switching axiom guarantees the existence of a path $R_\sigma = P_\sigma \times_{e^\theta} Q_\sigma$ in $(E^T, \mathscr{P}^T)$. Observe that $\text{first}(R_\sigma) = \text{first}(P_\sigma)$ and $\text{last}(R_\sigma) = \text{last}(Q_\sigma)$ holds. Therefore, $\widetilde{R}_\sigma$ and $\widetilde{P}_\sigma$ initiate with the same super source element and $\widetilde{R}_\sigma$ and $\widetilde{Q}_\sigma$ are extended by the same counting element and super sink. Hence, the extended path $\widetilde{Q}_\sigma$ is contained in $\widetilde{\mathscr{P}}_\sigma^T$. The same holds for $Q_\sigma \times_{e^\theta} P_\sigma$. □

Non-existence. As the Abstract Earliest Arrival Transshipment Problem is a generalization of the Earliest Arrival Transshipment Problem in the classical network flow model, abstract flows having the earliest arrival property do not exist. As a simple example consider the abstract network from Example 6.19. Besides the given feasible transshipment with time horizon $T = 2$ there is another feasible transshipment which at time $\theta = 1$ sends one unit of flow on path $R = (e)$ and another unit of flow on path $Q = (d, f)$. The first flow unit arrives at time 1 because R has zero transit time. The second flow unit arrives at time 3. This new solution sends one unit of flow earlier, but needs more time to send the second flow unit. Hence, no earliest arrival transshipment exists in the abstract network and the new solution is a 2-value-approximate flow.

Restricted Abstract Flows

We use a variant of the Abstract Flow Problem described by Martens and McCormick [MM08] to solve the weighted case. With the restricted variant we can approximate abstract earliest arrival flows using the framework from Section 4.3. For a given abstract maximum flow x, we may require for a saturated element e that in an augmentation step of Algorithm 6.1, the flow value through e remains unchanged.The problem can be represented by the following two modified versions of the linear programs for abstract maximum flow and abstract maximum cut.

Restricted Abstract Flows. Let $(E, \mathscr{P})$ be an abstract network with supermodular path weights r, let x be a feasible flow and let y be a dual solution. Let $\lambda \in \mathbb{Q}$ be given. Assume that the following conditions are satisfied by the given solutions. $\sum_{e \in P} y_e = r_P - \lambda$ holds for each path $P \in \mathscr{P}$ and $y_e \cdot \left(u_e - \sum_{P : e \in P} x_P\right) = 0$ holds for each element $e \in E$. The conditions are relaxed optimality conditions for (WAF) and (WAC) where λ specifies by how much solutions may deviate from the optimum.

A dual solution for x has to satisfy $y_e = 0$ for unsaturated elements. We denote the set of **restricted elements** by $\mathcal{R} \coloneqq \{e \in E \mid y_e > 0\} \subseteq E$. A solution of the Restricted Abstract Maximum Flow Problem is a maximum flow x under the condition that the flow through restricted elements remains unchanged. Such a flow can be computed as solution of the following linear program.

$$
\begin{aligned}
\max \quad & \sum_{P \in \mathscr{P}} x_P, && && \text{(RAF)}\\
\text{s.t.} \quad & \sum_{P \in \mathcal{P} : e \in P} x_P && \leq u_e \quad && \text{for all } e \in E,\\
\text{s.t.} \quad & \sum_{P \in \mathcal{P} : e \in P} x_P && = u_e \quad && \text{for all } e \in \mathcal{R},\\
& x_P && \geq 0 \quad && \text{for all } P \in \mathcal{P}.
\end{aligned}
$$

The dual linear program solves the Restricted Abstract Minimum Cut Problem.

$$
\begin{aligned}
\min \quad & \sum_{e \in E} \ell_e u_e, && && \text{(RAC)}\\
\text{s.t.} \quad & \sum_{e \in P} \ell_e && \geq r_P \quad && \text{for all } P \in \mathscr{P},\\
& \ell_e && \geq 0 \quad && \text{for all } e \in E \setminus \mathcal{R}.
\end{aligned}
$$

Application in the Time-expanded Network. Let x^i be an abstract flow over time in the time-expanded network with extended paths $(\widetilde{E}^i, \widetilde{\mathscr{P}}_\sigma^i)$ for a time horizon i. All paths use a counting element t_c^θ for some $\theta \in \{1, 2, \dots, i\}$. Let $v(t_c^\theta) \coloneqq \sum_{P: t_c^\theta \in P} x_P$ be the value of flow through the counting element. We define a new time-expanded abstract network by specifying new element capacities u', which remain $u'_e = u_e$ for all elements e that are not counting elements. The capacity of counting elements is restricted to the flow value through them, i. e., we set $u'_{t_c^\theta} \coloneqq v(t_c^\theta)$ for all counting elements t_c^θ in the time-expanded ground set.

The flow x^i remains feasible in the time-expanded abstract network with the new element capacities u'. Because all paths use exactly one of the counting elements and they are all saturated, we can define a feasible dual solution by setting

$$y_e \coloneqq \begin{cases} u'_e & e \text{ is a counting element} \\ 0 & \text{else} \end{cases} \qquad \text{for all } e \in \widetilde{E}^i.$$

The flow x and dual values y are feasible in the larger time-expanded network $\left(\widetilde{E}^{i+1}, \widetilde{\mathscr{P}}_\sigma^{i+1}\right)$ for a time horizon increased by 1. We want to use the algorithm by Martens and McCormick for the Restricted Abstract Maximum Flow Problem to augment flow without removing flow on the counting elements. To define an instance for the problem we need rewards. In the classical setting, the Triple-Optimization-Theorem [JR82] states, that an earliest arrival flow is equal to a minimum cost flow where the costs equal the transit times. We will use the same idea and define rewards such that they reflect the arrival time. The earlier a path arrives, the higher we set its reward. Paths arriving in the first time step, i. e., paths with zero travel time, have a reward of T. The reward decreases linearly with the arrival time and paths arriving at time T have a reward of 1. More formally, let $\widetilde{P}_\sigma$ be a path in the time-expanded network for time horizon T. Let $\text{last}(P) = t$ be the sink element of the underlying path and let θ be the arrival time of $\widetilde{P}_\sigma$ such that the path uses the sink element copy t^θ and the counting element t_c^θ. We then define the reward as

$$r\left(\widetilde{P}_\sigma\right) \coloneqq T - \theta + 1$$

Lemma 6.23. *The weight function $r\left(\widetilde{P}_\sigma\right) \coloneqq T - \theta + 1$ is supermodular.*

Proof. Let P, Q with $e \in P \cap Q$ and $R = P \times_e Q$ and $R' = Q \times_e P$. In the time-expanded network with extended paths the switched paths satisfy $\text{last}(R) = \text{last}(Q)$ and $\text{last}(R') = \text{last}(P)$. Therefore, R arrives at the same time θ as Q and R' arrives at the same time θ' as P. The statement of the lemma then follows with

$$r(P \times_e Q) + r(Q \times_e P) = r(R) + r(R') = T - \theta + 1 + T - \theta' + 1 = r(Q) + r(P).$$

□

We can now combine the results and derive a greedy value-approximation algorithm for abstract earliest arrival transshipments in increasing time-expanded networks.

Algorithm 6.3: Greedy Abstract Value-approximate Earliest Arrival Flow Algorithm

Input: An abstract network $(E, \mathscr{P})$ with transit times τ_e for the elements, supplies b_e^+ for sources and demands b_e^- for sinks.

Output: A 2-value-approximate abstract flow over time.

1. Set $i \coloneqq 1$ and compute an abstract flow x^1 in $(\widetilde{E}^1, \widetilde{\mathscr{P}}_\sigma^1)$. Define $\lambda \coloneqq 1$.
2. Let $(\widetilde{E}^{i+1}, \widetilde{\mathscr{P}}_\sigma^{i+1})$ be the abstract time-expanded network with extended paths for time horizon $i+1$. Define path weights $r\left(\widetilde{P}_\sigma\right) \coloneqq T - \theta + 1$, updated capacities

$$u'_e \coloneqq \begin{cases} v(t_c^\theta) & \text{if } e = t_c^\theta \text{ is a counting element} \\ u_e & \text{else} \end{cases} \quad \text{for all } e \in \widetilde{E}^{i+1},$$

   and dual values

$$y_e \coloneqq \begin{cases} r(\widetilde{P}_\sigma) - \lambda & e \text{ is a counting element on path } \widetilde{P}_\sigma \in (\widetilde{E}^i, \widetilde{\mathscr{P}}_\sigma^i) \\ 0 & \text{else} \end{cases},$$

   for all elements $e \in \widetilde{E}^{i+1}$.
3. Compute an abstract flow x^{i+1} in $(\widetilde{E}^{i+1}, \widetilde{\mathscr{P}}_\sigma^{i+1})$ that is a solution for the dual problems (RAF) and (RAC) with the RAMFMC-Algorithm by Martens and McCormick [MM08].
4. If x^{i+1} satisfies all balances, return f^{i+1}. Else, set $i \coloneqq i+1$ and continue with 2.

Before we prove the correctness of the algorithm we briefly review its behaviour when it is called with the instance from Example 6.19. Recall that the instance does not allow for an abstract earliest arrival transshipment.

Example 6.24. *The abstract network consists of the elements $E = \{d, e, f\}$ and paths $\mathscr{P} = \{P_1 = (d, e), P_2 = (e, f), Q = (d, f), R = (e)\}$. Transit times are $\tau_d = 1$, $\tau_e = 0$ and $\tau_f = 1$. Both source elements and sink elements have demands and supply of 2, respectively. We have already seen, that no earliest arrival flow exists.*

In the first iteration, an abstract flow x^1 in the network $(\widetilde{E}^1, \widetilde{\mathscr{P}}_\sigma^1)$ is computed. In the first time layer only element e^1 is available and any maximum flow uses the path $\widetilde{R} = (s_e^*, e^1, e_c^1, t_e^*)$ with counting element e_c^1.

Consider now the second time step. The reduced capacity for the counting element e_c^1 is set to $u'_{e_c^1} = 1$. Paths available at time 2 are the second copy of R as well as the first copies of P_1 and P_2 which both have a transit time of 1. Both of those arcs cannot be used to increase the flow value because both the supply and demand of $\text{last}(P_1) = e = \text{first}(P_2)$ are already satisfied and flow on e_c^1 cannot be reduced. Therefore, we have $x^2 = x^1$.

In the next iteration the first temporal copy of Q with transit time 2 is available. The last flow unit is sent along the extended path $\widetilde{Q} = (s_d^*, d^1, f^2, f_c^2, t_f^*)$. At the end of the iteration, the last flow is sent and the algorithm computed a 2-value-approximate abstract earliest arrival flow.

Observe that an augmenting structure consisting of the three paths

$$Q_1^+ = (s_e^*, e^1, f^1, f_c^2, t_f^*)$$
$$P_1^- = (s_e^*, e^1, e_c^1, t_e^*)$$
$$Q_2^+ = (s_d^*, d^1, e^2, e_c^2, t_e^*)$$

could be used in the second iteration to improve the flow that arrives until time 2. Because e_c^1 is a restricted element, the call of the RAMFMC-Algorithm in step 3 does not augment the flow value with this structure. This is also the reason why the algorithm computes a 2-value-approximate earliest arrival flow: Any augmenting structure that can improve a flow computed by RAMFMC has to remove flow on a counting element. We conclude with the following theorem.

Theorem 6.25. *Let $(E, \mathscr{P})$ be a terminal respecting abstract network that preserves the order. Let u_e be capacities on the elements, τ_e transit times on the elements and b^+ supplies and b^- demands. Then, Algorithm 6.3 computes a 2-value-approximate earliest arrival flow.*

Proof. We first have to show that y, u' and r are well defined such that the prerequisites for the application of RAMFMC-Algorithm in step 3 are satisfied. $\sum_{e\in P} y_e = r_P - \lambda$ holds by definition. For paths $\widetilde{P}_\sigma$ that strictly arrive before time $i+1$ the weights satisfy $r(\widetilde{P}_\sigma) > 1$ and therefore $y_e = r(\widetilde{P}_\sigma) - \lambda > 0$. Due to the restriction of the capacities to u', all counting elements e are saturated and have a positive dual value y_e. Consequently $y_e \cdot (u_e - \sum_{P:e\in P} x_P) = 0$ holds. Lastly, $x_P \cdot (\sum_{e\in P} y_e - r_P + \lambda) = 0$ is also satisfied. Hence, we can use the algorithm to compute optimum solutions to (RAF) and (RAC).

To show the approximation guarantee, by Lemma 4.17 we have to show that $|x^{i+1}| \geq \frac{1}{2}|x^{i+1,*}|$, where $x^{i+1,*}$ is the value of a maximum abstract flow over time in the time-expanded abstract network $\left(\widetilde{E}^{i+1}, \widetilde{\mathscr{P}}_\sigma^{i+1}\right)$.

Consider the computed flow x^{i+1} in the network $\left(\widetilde{E}^{i+1}, \widetilde{\mathscr{P}}_\sigma^{i+1}\right)$. We can use the Abstract Maximum Flow Algorithm 6.1 with x^{i+1} as an initial solution to compute the abstract maximum flow $x^{i+1,*}$ for time horizon $i+1$. Consider an iteration of the algorithm that augments the flow by a positive value. The augmenting structure used for the augmentation has to use a negative path P^- that actually has a counting element in the negative segment such that the element's flow value is decreased by the augmentation. This is because any other augmenting structure would increase the flow and is also valid for the Restricted Abstract Maximum Flow Problem value which is a violation to x^{i+1} being maximal.

Thus, for any flow unit that is increased in the process, flow on a counting element has to be decreased. However, the total flow value on the counting elements is a bound to the actual flow value because each path uses such an element. Hence, we can only send as much flow as has already been sent by x^{i+1}, and consequently $|x^{i+1}| \geq \frac{1}{2}|x^{i+1,*}|$. □

List of Figures

List of Algorithms

Notation Index

General Notation

$\mathbb{N}, \mathbb{N}_0$	natural numbers, and with zero included	
$\mathbb{Q}$	rational numbers	
$\mathbb{R}, \mathbb{R}_{\geq 0}$	real numbers, and the non-negative numbers	
$\mathbb{Z}$	integer numbers	
$\mathcal{P}(\cdot)$	power set of a set	
$\uplus$	disjoint union of two sets; $A \uplus B$ implies $A \cap B = \varnothing$	
$\subset$	strict subset	
$\subseteq$	subset or equal	

Graphs and Static Networks

$\mathcal{P}$	The class of problems polynomial algorithms.	9
$\mathcal{NP}$	The class of problems with non-deterministic polynomial algorithms.	9
$\text{dist}(v, w)$	shortest path distance between two nodes v and w	14
ε	approximation factor for approximation algorithms	10
e	an arc in a graph or network	13
f	an edge flow (over time)	15
$f(e)$	the flow value of edge flow e on arc e	15
$\delta^-(\cdot)$	incoming arcs of either a vertex or a set of vertices	13
$\delta^+(\cdot)$	outgoing arcs of either a vertex or a set of vertices	13
$G = (V, E)$	a graph with set of nodes and arcs	13
E	the set of directed arcs in a graph or network	13
$\text{tail}(e)$	tail (start node) of arc e	13
$\text{head}(e)$	head (end node) of arc e	13
V	the set of vertices in a graph or network	13
u, v, w	nodes in a graph	13
f^1, f^i	rank of flows in lexicographical order	23
$\overleftarrow{E}$	the set of reverse arcs of a residual network	19
$\mathcal{N}$	a static or dynamic network	15
$\mathcal{N}^*$	extended network with additional vertices and arcs	17
$\overleftarrow{e}$	the reverse arc belonging to arc e	19
t	a sink in a network	15
S^-	set of sinks	15
s	a source in a network	15
S^+	set of sources	15
b_v	supplies/demands for vertices	17
u_e	capacity of an arc e	15
$c(\cdot)$	cost of a flow	20
$f \in \mathcal{O}(g)$	$\mathcal{O}$-notation; means f asymptotically does not grow faster than g	8

$f \in \tilde{\mathcal{O}}(g)$	short form for $f \in \mathcal{O}(g \cdot \log(g))$	8
$\mathscr{P}$	a family of paths	14
P	a path in a graph	14
$\lvert f \rvert$	the value of flow f	15
x	a path based network flow	17
Flows over Time		
i	a commodity in a multi-commodity flow	27
K	set of commodities	27
$\mathrm{ex}_{f_i}(v, \theta)$	excess at v at time θ	27
$\mathrm{in}_{f_i}(v, \theta)$	inflow of commodity i into v at time θ	27
$\mathrm{out}_{f_i}(v, \theta)$	outflow of commodity i from v at time θ	27
e^θ	temporal copy of arc e starting at time θ	34
$\lvert f \rvert$	flow value	28
$\lvert f \rvert_\theta$	flow value at time θ	28
f	flow over time	27
$\mathcal{N}$	dynamic network	26
$\mathcal{N}^T$	time-expanded network with time horizon T	33
E^T	arcs in a time expanded network	34
S^{+T}	sources in a time expanded network	33
S^{-T}	sinks in a time expanded network	33
V^T	vertex set of a time expanded network	33
$\mathscr{P}$	set of paths	30
$\mathscr{P}_e$	set of paths using arc e	30
p_f	arrival pattern of a flow over time f	49
p^*	earliest arrival pattern	49
d	deadlines for arcs	42
r	release dates for arcs	42
S^+, S^-	set of sources and sinks	26
s^*	super source	33
t^*	super sink	33
τ	transit times for arcs	26
$\tau(P)$	travel time of path P	30
T^*	optimal time horizon for a transshipment over time	52
T	time horizon	27
θ	a time within the time horizon	27
v^θ	copy at time θ of node v	33
x	a path flow over time	30
Evacuation Simulation		
$\mathcal{C}$	the set of cells	61
$CA = (\mathcal{C}, N, \mathcal{E}, \mathrm{pot}_e)$	cellular automaton	61
c	a cell	61
pot_e	a potential for exit e	61
$\mathcal{E}$	a set of exits	61
$\mathcal{I}, i$	a set of evacuees and a single evacuee	61

M	state mapping of a cellular automaton	61
N	neighbourhood function	61
Approximate Earliest Arrival Flows		
α	an approximation factor for time-approximate earliest arrival flows	88
β	an approximation factor for value-approximate earliest arrival flows	88
Δ	a condensation factor	119
I_i	half open interval $[2^i - 1, 2^{i+1} - 1[$	92
L	list of points in time	118
$\mathcal{N}^L$	condensed time-expanded network	118
Ξ_x	the weighted sum of arrival times of a flow	94
Negative Travel Times		
c_e	arc costs	138
ℓ	arc lengths	138
M	a matching over time	149
M^s	a static matching	150
π_v	shortest path distance from a source to v	135
$\tau(P)$	travel time of path P	133
$\tau^+(P)$	the time interval in which P cannot be used such that flow arrives before T	134
$\tau^-(P)$	the (non-negative) time interval in which flow cannot be sent along P	134
$\tau_e(v)$	setup time for node v incident to arc e	149
Abstract Flows over Time		
$\geq_L$	lexicographic order of flows	165
$\leq_P$	order of elements in P	156
E^T	time-expanded ground set	161
P, Q, R	abstract paths	156
$P \times_e Q$	the path that exist due to the crossing axiom if two paths intersect	157
$P_{]e\rightarrow]}$	the elements of P beginning at e exclusive	156
$P_{[e\rightarrow]}$	the elements of P beginning at e inclusive	156
$P_{[\rightarrow e[}$	the elements of P up to e exclusive	156
$P_{[\rightarrow e]}$	the elements of P up to e inclusive	156
P_σ	temporal copy of path P with a waiting pattern σ	162
P_θ	temporal copy of path P starting at time θ	161
$\mathscr{P}^T$	the set of temporal path copies	161
$\mathscr{P}^T_\sigma$	the set of temporal path copies with intermediate waiting	162
$(E^T, \mathscr{P}^T_\sigma)$	time-expanded abstract network	162
$(\widetilde{E}^T, \widetilde{\mathscr{P}}^T_\sigma)$	the time-expanded network with extended paths	171
$\mathbf{O}$	a separation oracle for abstract flows	159
r_P	path weights	158
σ	a waiting pattern	162
$\lvert x\rvert^-_t$	inflow into sink t in the abstract flow x	165
$\lvert x\rvert^+_s$	outflow of source s in the abstract flow x	165

Subject Index

Bibliography

[ABW+14] David Adjiashvili, Sandro Bosio, Robert Weismantel and Rico Zenklusen. **Time-expanded Packings**. In: Javier Esparza, Pierre Fraigniaud, Thore Husfeldt and Elias Koutsoupias, editors, *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014)*. Volume 8572 of *Lecture Notes in Computer Science*, pages 64–76. Springer, Berlin, Heidelberg, 2014.

[ADS10] Giovanni Andreatta, Luigi De Giovanni and Giorgio Salamaso. **Fleet Quickest Routing on Grids: A Polynomial Algorithm**. *International Journal of Pure and Applied Mathematics*, 62(4): 419–432, 2010.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti and James B. Orlin. **Network Flows: Theory, Algorithms, and Applications**. Prentice Hall Inc., 1993.

[Aro89] Jay E. Aronson. **A Survey of Dynamic Network Flows**. *Annals of Operations Research*, 20(1): 1–66, 1989.

[BA99] Victor J. Blue and Jeffrey L. Adler. **Bi-directional Emergent Fundamental Pedestrian Flows from Cellular Automata Microsimulation**. *Transportation and Traffic Theory*, 14: 235–254, 1999.

[BdAM09] Rainer E. Burkard, Mauro Dell'Amico and Silvano Martello. **Assignment Problems**. Society for Industrial and Applied Mathematics, 2009.

[BDK93] Rainer E. Burkard, Karin Dlaska and Bettina Klinz. **The Quickest Flow Problem**. *ZOR - Methods and Models of Operations Research*, 37: 31–58, 1993.

[Bel58] Richard Bellman. **On a Routing Problem**. *Quarterly of Applied Mathematics*, 16: 87–90, 1958.

[Ber12] Aaron Bernstein. **Near Linear Time $(1+\varepsilon)$-Approximation for Restricted Shortest Paths in Undirected Graphs**. In: *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 189–201. Society for Industrial and Applied Mathematics, 2012.

[Ber78] Geoffrey N. Berlin. **The Use of Directed Routes for Assessing Escape Potential**. *Fire Technology*, 14(2): 126–135, May 1, 1978.

[BFA11] Umang Bhaskar, Lisa K. Fleischer and Elliot Anshelevich. **A Stackelberg Strategy for Routing Flow over Time**. In: *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 192–201. Society for Industrial and Applied Mathematics, 2011.

[BK04] Yuri Boykov and Vladimir Kolmogorov. **An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9): 1124–1137, 2004.

[BK07] Nadine Baumann and Ekkehard Köhler. **Approximating Earliest Arrival Flows with Flow-Dependent Transit Times**. *Discrete Applied Mathematics*, 155(2): 161–171, 2007.

[BKM+12] Sandro Bosio, Jan-Philipp W. Kappmeier, Jannik Matuschke, Britta Peis and Martin Skutella. **Flows over Time with Negative Transit Times and Arc Release Dates**. In: Andreas Brieden, Zafer-Korcan Görgülü, Tino Krug, Erik Kropat, Silja Meyer-Nieberg, Goran Mihelcic and Stefan Wolfgang Pickl, editors, *11th Cologne-Twente Workshop on Graphs and Combinatorial Optimization. Extended Abstracts*, pages 30–33. Munich, Germany, 2012.

[BKS+01] Carsten Burstedde, Kai Klauck, Andreas Schadschneider and Johannes Zittartz. **Simulation of Pedestrian Dynamics Using a 2-Dimensional Cellular Automaton**. *Physica A*: 507–525, 2001.

[Bol55] Alexander W. Boldyreff. **Determination of the Maximal Steady State Flow of Traffic Through a Railroad Network**. *Journal of the Operations Research Society of America*, 3(4): 443–465, 1955.

[Bre65] Jack E. Bresenham. **Algorithm for Computer Control of a Digital Plotter**. *IBM Systems Journal*, 4(1): 25–30, 1965.

[BS09] Nadine Baumann and Martin Skutella. **Earliest Arrival Flows with Multiple Sources**. *Mathematics of Operations Research*, 34(2): 499–512, 2009.

[CFS82] Luc G. Chalmet, Richard L. Francis and Patsy B. Saunders. **Network Models for Building Evacuation**. *Fire Technology*, 18: 90–113, 1982.

[CG97] Boris V. Cherkassky and Andrew V. Goldberg. **On Implementing the Push–Relabel Method for the Maximum Flow Problem**. *Algorithmica*, 19(4): 390–410, 1997.

[CHT88] Wonjoon Choi, Horst W. Hamacher and Süleyman Tüfekçi. **Modeling of Building Evacuation Problems by Network Flows with Side Constraints**. *European Journal of Operational Research*, 35(1): 98–110, 1988.

[Chv73] Vašek Chvátal. **Edmonds Polytopes and a Hierarchy of Combinatorial Problems**. *Discrete Mathematics*, 4(4): 305–337, 1973.

[CSM04] Richard L. Church, Maria P. Scaparra and Richard S. Middleton. **Identifying Critical Infrastructure: The Median and Covering Facility Interdiction Problems**. *Annals of the Association of American Geographers*, 94(3): 491–502, 2004.

[Cun76] William H. Cunningham. **A Network Simplex Method**. *Mathematical Programming*, 11(1): 105–116, 1976.

[Dan51a] George B. Dantzig. **Application of the Simplex Method to a Transportation Problem**. *Activity Analysis of Production and Allocation*, 13: 359–373, 1951.

[Dan51b] George B. Dantzig. **Maximization of a Linear Function of Variables Subject to Linear Inequalities**. In: Tjalling C. Koopmans, editor, *Activity Analysis of Production and Allocation Cowles Commission Monograph No. 13*, pages 339–347. Jon Wiley & Sons, Inc., New York, NY, USA, 1951.

[DF56] George B. Dantzig and Delbert R. Fulkerson. **On the Max-Flow Min-Cut Theorem of Networks**. In: *Linear Inequalities and Related Systems Annals of Mathematics Studies, no. 38*, pages 215–221. Princeton University Press, Princeton, NJ, 1956.

[DGK+10] Daniel Dressler, Martin Groß, Jan-Philipp Kappmeier, Timon Kelter, Joscha Kulbatzki, Daniel Plümpe, Gordon Schlechter, Melanie Schmidt, Martin Skutella and Sylvie Temme. **On the Use of Network Flow Techniques for Assigning Evacuees to Exits**. *Procedia Engineering*, 3: 205–215, 2010. 1st Conference on Evacuation Modeling and Management.

[Dha14] Tanka N. Dhamala. **A Survey on Models and Algorithms for Discrete Evacuation Planning Network Problems**. *Journal of Industrial and Management Optimization*, 11(1): 265–289, 2014.

[Dij59] Edsger W. Dijkstra. **A Note on Two Problems in Connexion with Graphs**. *Numerische Mathematik*, 1: 269–271, 1959.

[Din70] Efim A. Dinic. **Algorithms for Solution of a Problem of Maximum Flow in Networks with Power Estimation**. *Doklady Akademii Nauk SSSR*, 194(4), 1970. English translation: *Soviet Mathematics Doklady*, 11(5): 1277-1280, 1970.

[DS11] Daniel Dressler and Martin Skutella. **An FPTAS for Flows over Time with Aggregate Arc Capacities**. In: Klaus Jansen and Roberto Solis-Oba, editors, *Proceedings of the 8th Workshop on Approximation and Online Algorithms (WAOA 2010)*. Volume 6534 of *Lecture Notes in Computer Science*, pages 106–117. Springer, Berlin, Heidelberg, 2011.

[DS15] Yann Disser and Martin Skutella. **The Simplex Algorithm is $\mathcal{NP}$-mighty**. In: *Proceedings of the 26rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 858–872. Society for Industrial and Applied Mathematics, 2015.

[EFS56] Peter Elias, Amiel Feinstein and Claude E. Shannon. **A Note on the Maximum Flow Through a Network**. *Institute of Radio Engineers, Transactions on Information Theory*, IT-2(December): 117–119, 1956.

[EG77] Jack Edmonds and Rick Giles. **A Min-max Relation for Submodular Functions on Graphs**. *Annals of Discrete Mathematics*. Studies in Integer Programming, 1: 185–204, 1977. Peter L. Hammer, Ellis L. Johnson, Bernhard H. Korte and George L. Nemhauser, editors.

[EGI98] David Eppstein, Zvi Galil and Giuseppe F. Italiano. **Dynamic Graph Algorithms**. In: Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, pages 8-1–8-25. CRC Press, 1998.

[EHH+] Harri Ehtamo, Simo Heliövaara, Simo Hostikka and Timo Korhonen. **Modeling Evacuees' Exit Selection with Best Response Dynamics**. In: Wolfram W. F. Klingsch, Christian Rogsch, Andreas Schadschneider and Michael Schreckenberg, editors, *Pedestrian and Evacuation Dynamics 2008*, pages 309–319. Springer, Berlin, Heidelberg, 2010.

[EK72] Jack Edmonds and Richard M. Karp. **Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems**. *Journal of the Association for Computing Machinery*, 19(2): 248–264, 1972.

[FD55] Delbert R. Fulkerson and George B. Dantzig. **Computation of Maximal Flows in Networks**. *Naval Research Logistics Quarterly*, 2(4): 277–283, 1955.

[FF56] Lester R. Ford Jr. and Delbert R. Fulkerson. **Maximal Flow Through a Network**. *Canadian Journal of Mathematics*, 8: 399–404, 1956.

[FF57] Lester R. Ford Jr. and Delbert R. Fulkerson. **A Simple Algorithm for Finding Maximal Network Flows and an Application to the Hitchcock Problem**. *Canadian Journal of Mathematics*, 9: 210–218, 1957.

[FF58] Lester R. Ford Jr. and Delbert R. Fulkerson. **Constructing Maximal Dynamic Flows from Static Flows**. *Operations Research*, 6(3): 419–433, 1958.

[FF62] Lester R. Ford Jr. and Delbert R. Fulkerson. **Flows in Networks**. Princeton University Press, Princeton, NJ, USA, 1962.

[Fle01] Lisa K. Fleischer. **Faster Algorithms for the Quickest Transshipment Problem**. *SIAM Journal on Optimization*, 12(1): 18–35, 2001.

[For56] Lester R. Ford. **Network Flow Theory**, The RAND Corporation Paper P-923, 1956.

[Fra81] Richard L. Francis. **A 'Uniformity Principle' for Evacuation Route Allocation**. *Journal of Research of the National Bureau of Standards*, 86(5): 509–513, 1981.

[FS07] Lisa K. Fleischer and Martin Skutella. **Quickest Flows over Time**. *SIAM Journal on Computing*, 36(6): 1600–1630, 2007.

[FT91] Drew Fudenberg and Jean Tirole. **Game Theory**. MIT Press, 1991.

[FT98] Lisa K. Fleischer and Éva Tardos. **Efficient Continuous-Time Dynamic Network Flow Algorithms**. *Operations Research Letters*, 23(3-5): 71–80, 1998.

[Fuj03] Satoru Fujishige. **A Maximum Flow Algorithm Using MA Ordering**. *Operations Research Letters*, 31(3): 176–178, 2003.

[Ful61] Delbert R. Fulkerson. **A Network Flow Computation for Project Cost Curves**. *Management Science*, 7(2): 167–178, 1961.

[Gai97] Arlette Gaillard. **Switchdec Polyhedra**. *Discrete Applied Mathematics*. Second International Colloquium on Graphs and Optimization, 76(1): 141–163, 1997.

[Gal58] Tibor Gallai. **Maximum-Minimum Sätze über Graphen**. *Acta Mathematica Academiae Scientiarum Hungarica*, 9(3-4): 395–434, September 1958.

[Gal59] David Gale. **Transient Flows in Networks**. *The Michigan Mathematical Journal*, 6(1): 59–63, 1959.

[Gar70] Martin Gardner. **Mathematical Games: The Fantastic Combinations of John Conway's New Solitaire Game "life"**. *Scientific American*, 223(4): 120–123, 1970.

[GGT89] Giorgio Gallo, Michael D. Grigoriadis and Robert E. Tarjan. **A Fast Parametric Maximum Flow Algorithm and Applications**. *SIAM Journal on Computing*, 18(1): 30–55, 1989.

[GH82] Heinz Gröflin and Alan J. Hoffman. **Lattice Polyhedra II: Generalization, Constructions and Examples**. *North-Holland Mathematics Studies*, 65: 189–203, 1982.

[GJ79] Michael R. Garey and David S. Johnson. **Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness**. W. H. Freeman & Co., New York, NY, USA, 1979.

[GK98] Naveen Garg and Jochen Könemann. **Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems**. In: *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS 1998)*, pages 300–309. IEEE, 1998.

[GKS+12] Martin Groß, Jan-Philipp W. Kappmeier, Daniel Schmidt and Melanie Schmidt. **Approximating Earliest Arrival Flows in Arbitrary Networks**. In: Leah Epstein and Paolo Ferragina, editors, *Proceedings of the 20th European Symposium on Algorithms (ESA 2012)*. Volume 7501 of *Lecture Notes in Computer Science*, pages 551–562. Springer, Berlin, Heidelberg, 2012.

[GKT51] David Gale, Harold W. Kuhn and Albert W. Tucker. **Linear Programming and the Theory of Games**. *Activity Analysis of Production and Allocation*, 13: 317–335, 1951.

[GLP14] GLPK Community. **GNU Linear Programming Kit**. 2014.

[GLS81] Martin Grötschel, László Lovász and Alexander Schrijver. **The Ellipsoid Method and its Consequences in Combinatorial Optimization**. *Combinatorica*, 1(2): 169–197, 1981.

[GLS88] Martin Grötschel, László Lovász and Alexander Schrijver. **Geometric Algorithms and Combinatorial Optimization**, volume 2 of *Algorithms and Combinatorics*. Springer, Berlin, Heidelberg, 2nd edition, 1988.

[Gol08] Andrew V. Goldberg. **The Partial Augment–Relabel Algorithm for the Maximum Flow Problem**. In: Dan Halperin and Kurt Mehlhorn, editors, *Proceedings of the 16th European Symposium on Algorithms (ESA 2008)*. Volume 5193 of *Lecture Notes in Computer Science*, pages 466–477. Springer, Berlin, Heidelberg, 2008.

[Gol97] Andrew V. Goldberg. **An Efficient Implementation of a Scaling Mini-mum-Cost Flow Algorithm**. *Journal of Algorithms*, 22(1): 1–29, 1997.

[Gom58] Ralph E. Gomory. **Outline of an Algorithm for Integer Solutions to Linear Programs**. *Bulletin of the American Mathematical Society*, 64(5): 275–278, 1958.

[Gom60] Ralph E. Gomory. **Solving Linear Programming Problems in Integers**. *Combinatorial Analysis*, 10: 211–215, 1960.

[GR98] Andrew V. Goldberg and Satish Rao. **Beyond the Flow Decomposition Barrier**. *Journal of the ACM*, 45(5): 783–797, 1998.

[GRK+12] Ashish Goel, Kajamalai G. Ramakrishnan, Deepak Kataria and Dimitris Logothetis. **Efficient Computation of Delay-Sensitive Routes from One Source to all Destinations**. In: *IEEE INFOCOM 2001. 20th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, volume 2, pages 854–858. 2001.

[Gro09] Martin Groß. **Flows over Time with Commodity-dependent Transit Times**. Master's thesis. Technische Universität Dortmund, 2009.

[GS12a] Martin Groß and Martin Skutella. **Generalized Maximum Flows over Time**. In: *Proceedings of the 9th Workshop on Approximation and Online Algorithms (WAOA 2011)*. Volume 7164 of *Lecture Notes in Computer Science*, pages 247–260. Springer, Berlin, Heidelberg, 2012.

[GS12b] Martin Groß and Martin Skutella. **Maximum Multicommodity Flows over Time without Intermediate Storage**. In: Leah Epstein and Paolo Ferragina, editors, *Proceedings of the 20th European Symposium on Algorithms (ESA 2012)*. Volume 7501 of *Lecture Notes in Computer Science*, pages 539–550. Springer, Berlin, Heidelberg, 2012.

[GS15] Martin Groß and Martin Skutella. **A Tight Bound on the Speed-up Through Storage for Quickest Multi-commodity Flows**. *Operations Research Letters*, 43(1): 93–95, January 2015.

[GT88] Andrew V. Goldberg and Robert E. Tarjan. **A new Approach to the Maximum-Flow Problem**. *Journal of the Association for Computing Machinery*, 35(4): 921–940, 1988.

[GT89] Andrew V. Goldberg and Robert E. Tarjan. **Finding Minimum-cost Circulations by Canceling Negative Cycles**. *Journal of the ACM*, 36(4): 873–886, 1989.

[Gur14] Gurobi Optimization, Inc. **Gurobi Optimizer Reference Manual**. 2014.

[Hač79] Leonid G. Hačijan. **Polinomialʹnyi algoritm v lineinom programmirovanii**. *Doklady Akademii Nauk SSSR*, 244(5): 1093–1096, 1979. English tranlation: Leonid G. Hačijan. **A Polynomial Algorithm in Linear Programming**. *Soviet Mathematics Doklady*, 20(1): 191–194, 1979.

[Has92] Refael Hassin. **Approximation Schemes for the Restricted Shortest Path Problem**. *Mathematics of Operations Research*, 17(1): 36–42, 1992.

[HFM+02] Dirk Helbing, Illes J. Farkas, Péter Molnár and Tamás Vicsek. **Simulation of Pedestrian Crowds in Normal and Evacuation Situations**. *Pedestrian and Evacuation Dynamics*, 21: 21–58, 2002.

[HGS11] Yunjun Han, Xiaohong Guan and Leyuan Shi. **Optimization Based Method for Supply Location Selection and Routing in Large-Scale Emergency Material Delivery**. *IEEE Transactions on Automation Science and Engineering*, 8(4): 683–693, 2011.

[HHS07] Alex Hall, Steffen Hippler and Martin Skutella. **Multicommodity Flows over Time: Efficient Algorithms and Complexity**. *Theoretical Computer Science*, 379(3): 387–404, 2007.

[HKM+14] Tobias Harks, Felix König, Jannik Matuschke, Alexander Richter and Jens Schulz. **An Integrated Approach to Tactical Transportion Planning in Logistics Networks**. *Transportation Science*, 2014. To appear.

[HL07] Laura Heinrich-Litan and Marco E. Lübbecke. **Rectangle Covers Revisited Computationally**. *Journal of Experimental Algorithmics*, 11: 2–6, 2007.

[HO84] Bruce Hajek and Richard G. Ogier. **Optimal Dynamic Routing in Communication Networks with Continuous Traffic**. *Networks*, 14(3): 457–487, 1984.

[Hoc97] Dorit S. Hochbaum. **Approximation Algorithms for NP-hard Problems**. PWS Publishing Company, 1997.

[Hof74] Alan J. Hoffman. **A Generalization of Max Flow – Min Cut**. *Mathematical Programming*, 6(1): 352–359, 1974.

[Hof78] Alan J. Hoffman. **On Lattice Polyhedra III: Blockers and Anti-blockers of Lattice Clutters**. In: *Polyhedral Combinatorics*, pages 197–207. Springer, Berlin, Heidelberg, 1978.

[Hop95] Bruce Hoppe. **Efficient Dynamic Network Flow Algorithms**. PhD thesis. 1995.

[HR55] Ted E. Harris and Frank S. Ross. **Fundamentals of a Method for Evaluating Rail Net Capacities**, The RAND Corporation Research Memorandum RM-1573, 1955.

[HS78] Alan J. Hoffman and D. E. Schwartz. **On Lattice Polyhedra**. In: *Proceedings 5th Hungarian Colloquium on Combinatorics, North Holland*, pages 593–598. 1978.

[HT00] Bruce Hoppe and Éva Tardos. **The Quickest Transshipment Problem**. *Mathematics of Operations Research*, 25(1): 36–62, 2000.

[HT01] Horst W. Hamacher and Stevanus A. Tjandra. **Mathematical Modelling of Evacuation Problems: A State of Art**. Fraunhofer-Institut für Techno-und Wirtschaftsmathematik, Fraunhofer (ITWM), 2001.

[HT87] Horst W. Hamacher and Süleyman Tüfekçi. **On the Use of Lexicographic Min Cost Flows in Evacuation Modeling**. *Naval Research Logistics*, 34(4): 487–503, 1987.

[HT94] Bruce Hoppe and Éva Tardos. **Polynomial Time Algorithms for some Evacuation Problems**. In: *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1994)*, pages 433–441. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1994.

[HZ80] Gabriel Y. Handler and Israel Zang. **A Dual Algorithm for the Constrained Shortest Path Problem**. *Networks*, 10(4): 293–309, 1980.

[IBM14] IBM Corporation. **IBM ILOG CPLEX Optimizer**. 2014.

[Jok66] Hans C. Joksch. **The Shortest Route Problem with Constraints**. *Journal of Mathematical Analysis and Applications*, 14(2): 191–197, 1966.

[Jol02] Ian T. Jolliffe. **Principal Component Analysis**. Springer, Berlin, Heidelberg, 2002.

[JR82] John J. Jarvis and H. Donald Ratliff. **Some Equivalent Objectives for Dynamic Network Flow Problems**. *Management Science*, 28(1): 106–109, 1982.

[Kap09] Jan-Philipp W. Kappmeier. **Berechnung maximaler Flüsse unter Vermeidung von Residualnetzwerken**. Master's thesis. Technische Universität Dortmund, 2009.

[Kar72] Richard M. Karp. **Reducibility among Combinatorial Problems**. In: Raymond E. Miller, James W. Thatcher and Jean D. Bohlinger, editors, *Complexity of Computer Computations The IBM Research Symposia Series*, pages 85–103. Springer US, 1972.

[Kar74] Alexander V. Karzanov. **Nakhozhdenie maksimal'nogo potoka v seti metodom predpotokov**. *Doklady Akademii Nauk SSSR*, 215(1): 49–52, 1974. English translation: Alexander V. Karzanov. **Determining a Maximal Flow in a Network by the Method of Preflows**. *Soviet Mathematics Doklady*, 15(2): 434–437, 1974.

[Kar84] Narendra B. Karmarkar. **A New Polynomial-Time Algorithm for Linear Programming**. *Combinatorica. An International Journal of the János Bolyai Mathematical Society*, 4(4): 373–395, 1984.

[Kle67] Morton Klein. **A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems**. *Management Science*, 14(3): 205–220, 1967.

[Kli94] Bettina Klinz. Cited as personal communication in [Hop95]. 1994.

[KLS02] Ekkehard Köhler, Katharina Langkau and Martin Skutella. **Time-expanded Graphs for Flow-Dependent Transit Times**. In: Rolf Möhring and Rajeev Raman, editors, *Proceedings of the 10th European Symposium on Algorithms (ESA 2002)*. Volume 2461 of *Lecture Notes in Computer Science*, pages 599–611. Springer, Berlin, Heidelberg, 2002.

[Klü03] Hubert L. Klüpfel. **A Cellular Automaton Model for Crowd Movement and Egress Simulation**. PhD thesis. Universität Duisburg-Essen, 2003.

[KMP14] Jan-Philipp W. Kappmeier, Jannik Matuschke and Britta Peis. **Abstract Flows over Time: A First Step Towards Solving Dynamic Packing Problems**. *Theoretical Computer Science*. Algorithms and Computation, 544: 74–83, 2014.

[KMS09] Ekkehard Köhler, Rolf H. Möhring and Martin Skutella. **Traffic Networks and Flows over Time**. In: Jürgen Lerner, Dorothea Wagner and Katharina A. Zweig, editors, *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*. Volume 5515 of *Lecture Notes in Computer Science*, pages 166–196. Springer, Berlin, Heidelberg, 2009.

[KNS11] Ronald Koch, Ebrahim Nasrabadi and Martin Skutella. **Continuous and Discrete Flows over Time**. *Mathematical Methods of Operations Research*, 73(3): 301–337, 2011.

[Koc12] Ronald Koch. **Routing Games over Time**. PhD thesis. Technische Universität Berlin, 2012.

[KP82] Richard M. Karp and Christos H. Papadimitriou. **On Linear Characterizations of Combinatorial Optimization Problems**. *SIAM Journal on Computing*, 11(4): 620–632, 1982.

[KRT94] Valerie King, Satish Rao and Robert E. Tarjan. **A Faster Deterministic Maximum Flow Algorithm**. *Journal of Algorithms*, 17(3): 447–474, 1994.

[KS10] Ekkehard Köhler and Martin Strehler. **Traffic Signal Optimization Using Cyclically Expanded Networks**. In: *OASIcs – OpenAccess Series in Informatics*, volume 14. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2010.

[KS11] Ronald Koch and Martin Skutella. **Nash Equilibria and the Price of Anarchy for Flows over Time**. *Theory of Computing Systems*, 49(1): 71–97, 2011.

[KS12] Ekkehard Köhler and Martin Strehler. **Combining Static and Dynamic Models for Traffic Signal Optimization. Inherent Load-dependent Travel Times in a Cyclically Time-expanded Network Model**. *Procedia – Social and Behavioral Sciences*, 54: 1125–1134, 2012.

[KV12] Bernard Korte and Jens Vygen. **Combinatorial Optimization**, volume 21 of *Algorithms and Combinatorics*. Springer, Berlin, Heidelberg, 5th edition, 2012.

[KW04] Bettina Klinz and Gerhard J. Woeginger. **Minimum-Cost Dynamic Flows: The Series-Parallel Case**. *Networks*, 43(3): 153–162, 2004.

[LR01] Dean H. Lorenz and Danny Raz. **A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem**. *Operations Research Letters*, 28(5): 213–219, 2001.

[Mar07] Maren Martens. **Path-constrained Network Flows**. PhD thesis. Technische Universität Berlin, 2007.

[Mat13] Jannik Matuschke. **Network Flows and Network Design in Theory and Practice**. PhD thesis. Technische Universität Berlin, 2013.

[MAT14] **MATSim : Multi-Agent Transport Simulation Toolkit**. 2014.

[McC96] S. Thomas McCormick. **A Polynomial Algorithm for Abstract Maximum Flow**. In: *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1996)*, pages 490–497. Society for Industrial and Applied Mathematics, 1996.

[Meg74] Nimrod Megiddo. **Optimal Flows in Networks with Multiple Sources and Sinks**. *Mathematical Programming*, 7(1): 97–107, 1974.

[Meg79] Nimrod Megiddo. **Combinatorial Optimization with Rational Objective Functions**. *Mathematics of Operations Research*, 4(4): 414–424, 1979.

[Mel07] Vardges Melkonian. **Flows in Dynamic Networks with Aggregate arc Capacities**. *Information Processing Letters*, 101(1): 30–35, 2007.

[Min73] Edward Minieka. **Maximal, Lexicographic, and Dynamic Network Flows**. *Operations Research*, 21(2): 517–527, 1973.

[MKM78] Vishv M. Malhotra, M. Pramodh Kumar and Shachindra N. Maheshwari. **An $\mathcal{O}(V^3)$ Algorithm for Finding Maximum Flows in Networks**. *Information Processing Letters*, 7(6): 277–278, 1978.

[MM08] Maren Martens and S. Thomas McCormick. **A Polynomial Algorithm for Weighted Abstract Flow**. In: *Proceedings of the 13th Conference on Integer Programming and Combinatorial Optimization*. Volume 5035 of *Lecture Notes in Computer Science*, pages 97–111. Springer, Berlin, Heidelberg, 2008.

[MM70] Alan W. McMasters and Thomas M. Mustin. **Optimal interdiction of a supply network**. *Naval Research Logistics Quarterly*, 17(3): 261–268, 1970.

[Mon81] Gaspard Monge. **Mémoire sur la théorie des déblais et de remblais.** Pages 666–704. Histoire de l'Académie Royale des Sciences, Année M. DCCLXXXI, avec les Mémoires de Mathématique et de Physique, pour la même année, Paris, 1781.

[Moo59] Edward F. Moore. **The Shortest Path Through a Maze**. In: *Proceedings of an International Symposium on the Theory of Switching 1957, Part II*, pages 285–292. Harvard University Press, Cambridge, MA, USA, 1959.

[MP03] Ernesto Q. V. Martins and Marta M. B. Pascoal. **A New Implementation of Yen's Ranking Loopless Paths Algorithm**. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2): 121–133, 2003.

[Neu66] John von Neumann. **Theory of Self-Reproducing Automata**. Arthur W. Burks, editor. University of Illinois Press, Champaign, IL, USA, 1966.

[NRT+07] Noam Nisan, Tim Roughgarden, Éva Tardos and Vijay V. Vazirani. **Algorithmic Game Theory**. Cambridge University Press, New York, NY, USA, 2007.

[NS92] Kai Nagel and Michael Schreckenberg. **A Cellular Automaton Model for Freeway Traffic**. *Journal de Physique I*, 2(12): 2221–2229, 1992.

[Orl13] James B. Orlin. **Max Flows in $\mathcal{O}(nm)$ Time, or Better**. In: *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 765–774. ACM, New York, NY, USA, 2013.

[Orl93] James B. Orlin. **A Faster Strongly Polynomial Minimum Cost Flow Algorithm**. *Operations research*, 41(2): 338–350, 1993.

[Osb04] Martin J. Osborne. **An Introduction to Game Theory**. Oxford University Press, New York, NY, 2004.

[Pap94] Christos. H. Papadimitriou. **Computational Complexity**. Addison-Wesley, 1994.

[Phi90] Andrew B. Philpott. **Continuous-time Flows in Networks**. *Mathematics of Operations Research*, 15(4): 640–661, 1990.

[Pin12] Michael L. Pinedo. **Scheduling: Theory, Algorithms, and Systems**. Springer Berlin Heidelberg, 3rd edition, 2012.

[PR81] Manfred W. Padberg and M. Rammohan Rao. **The Russian Method for Linear Inequalities III: Bounded Integer Programming**, 1981.

[PW11] Britta Peis and Andreas Wiese. **Universal Packet Routing with Arbitrary Bandwidths and Transit Times**. In: Oktay Günlük and Gerhard J. Woeginger, editors, *Proceedings of the 15th Conference on Integer Programming and Combinatorial Optimization (IPCO 2011)*. Volume 6655 of *Lecture Notes in Computer Science*, pages 362–375. Springer, Berlin, Heidelberg, 2011.

[RSS11] Stefan Ruzika, Heike Sperber and Mechthild Steiner. **Earliest Arrival Flows on Series-Parallel Graphs**. *Networks*, 57(2): 169–173, 2011.

[RT02] D. Richardson and Éva Tardos. Cited as personal communication in [FS07]. 2002.

[RT07] Kaushik Roy and Claire J. Tomlin. **Solving the Aircraft Routing Problem Using Network Flow Algorithms**. In: *American Control Conference, 2007*, pages 3330–3335. IEEE, 2007.

[Sch02] Alexander Schrijver. **On the History of the Transportation and Maximum Flow Problems**. *Mathematical Programming*, 91(3): 437–445, 2002.

[Sch03] Alexander Schrijver. **Combinatorial Optimization: Polyhedra and Efficiency**, volume 24 of *Algorithms and Combinatorics*. Springer, Berlin, Heidelberg, 2003.

[Sch11] Marlen Schwengfelder. **Modellierung von Gebäudestrukturen durch Graphen in der Evakuierungsplanung**. Master's thesis. Technische Universität Berlin, 2011.

[Sch84] Alexander Schrijver. **Total Dual Integrality From Directed Graphs, Crossing Families, and Sub-and Supermodular Functions**. *Progress in Combinatorial Optimization*: 315–361, 1984.

[Sch98] Alexander Schrijver. **Theory of Linear and Integer Programming**. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Chichester, 1998.

[SCI14] Zuse Institute Berlin. **SCIP – Solving Constraint Integer Programs**. 2014.

[Sku09] Martin Skutella. **An Introduction to Network Flows over Time**. In: William J. Cook, László Lovász and Jens Vygen, editors, *Research Trends in Combinatorial Optimization*, pages 451–482. Springer, Berlin, Heidelberg, 2009.

[SS14] Melanie Schmidt and Martin Skutella. **Earliest arrival flows in networks with multiple sinks**. *Discrete Applied Mathematics*. Combinatorial Optimization, 164, Part 1: 320–327, 2014.

[SW10] Sebastian Stiller and Andreas Wiese. **Increasing Speed Scheduling and Flow Scheduling**. In: Otfried Cheong, Kyung-Yong Chwa and Kunsoo Park, editors, *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC 2010)*. Volume 6507 of *Lecture Notes in Computer Science*, pages 279–290. Springer, Berlin, Heidelberg, 2010.

[SZ99] Avi Shoshan and Uri Zwick. **All Pairs Shortest Paths in Undirected Graphs with Integer Weights**. In: *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS 1999)*, pages 605–614. 1999.

[SZJ09] Pablo Soldati, Haibo Zhang and Mikael Johansson. **Deadline-constrained Transmission Scheduling and Data Evacuation in Wireless HART Networks**. In: *Proceedings of the 10th European Control Conference (ECC 2009)*. IEEE, 2009.

[Tja03] Stevanus A. Tjandra. **Dynamic Network Optimization With Application to the Evacuation Problem**. PhD thesis. Technische Universität Kaiserslautern, 2003.

[Tol30] Aleksei N. Tolstoĭ. **Metody nakhozhdeniya naimen'shego summovogo kilometrazha priplanirovanii perevozok v prostranstve**. *Planirovanie Perevozok, Sbornik pervyĭ, Transpechat'NKPS*: 23–55, 1930. In Russian: **Methods of Finding the Minimal Total Kilometrage in Cargo-transportation Planning in Space**. *Transportation Planning, Volume I*, TransPress of the National Commissariat of Transportation.

[Tom71] Nobuaki Tomizawa. **On some Techniques Useful for Solution of Transportation Network Problems**. *Networks*, 1(2): 173–194, 1971.

[Vaz01] Vijay V. Vazirani. **Approximation Algorithms**. Springer, Berlin, Heidelberg, 2001.

[vN47] John von Neumann. **Discussion of a Maximum Problem**. In: John von Neumann and Abraham H. Taub, editors, *John von Neumann Collected Works: Theory of Games, Astrophysics, Hydrodynamics and Meteorology*. Volume 6, pages 89–95. Pergamon Press, 1963. Manuscript from 1947.

[Vyg04] Jens Vygen. **Near-Optimum Global Routing with Coupling, Delay Bounds, and Power Consumption**. In: Daniel Bienstock and George Nemhauser, editors, *Proceedings of the 10th Conference on Integer Programming and Combinatorial Optimization (IPCO 2004)*. Volume 3064 of *Lecture Notes in Computer Science*, pages 308–324. Springer, Berlin, Heidelberg, 2004.

[War87] Arthur Warburton. **Approximation of Pareto Optima in Multiple-Objective, Shortest-Path Problems**. *Operations Research*, 35(1): 70–79, 1987.

[Weg05] Ingo Wegener. **Complexity Theory**. Springer, Berlin, Heidelberg, 2005.

[Wei93] Ulrich Weidmann. **Transporttechnik der Fussgänger – Transporttechnische Eigenschaften des Fussgängerverkehrs (Literaturstudie)**. Technical report (90). In German. Institut für Verkehrsplanung, Transporttechnik, Strassen- und Eisenbahnbau IVT an der ETH Zürich, 1993.

[WG65] Christoph Witzgall and Alan J. Goldman. **Most Profitable Routing before Maintenance**. In: *Bulletin of the Operations Research Society of America*, B82. 1965.

[Wil71] William L. Wilkinson. **An Algorithm for Universal Maximal Dynamic Flows in a Network**. *Operations Research*, 19(7): 1602–1612, 1971.

[WS11] David P. Williamson and David B. Shmoys. **The Design of Approximation Algorithms**. Cambridge University Press, 1st edition, 2011.

[WSL+12] Yiwen Wang, Sen Su, Alex X. Liu and Zhongbao Zhang. **Multiple Bulk Data Transfers Scheduling Among Datacenters**. *Computer Networks*. Communications and Networking in the Cloud, 68: 123–137, 2014.

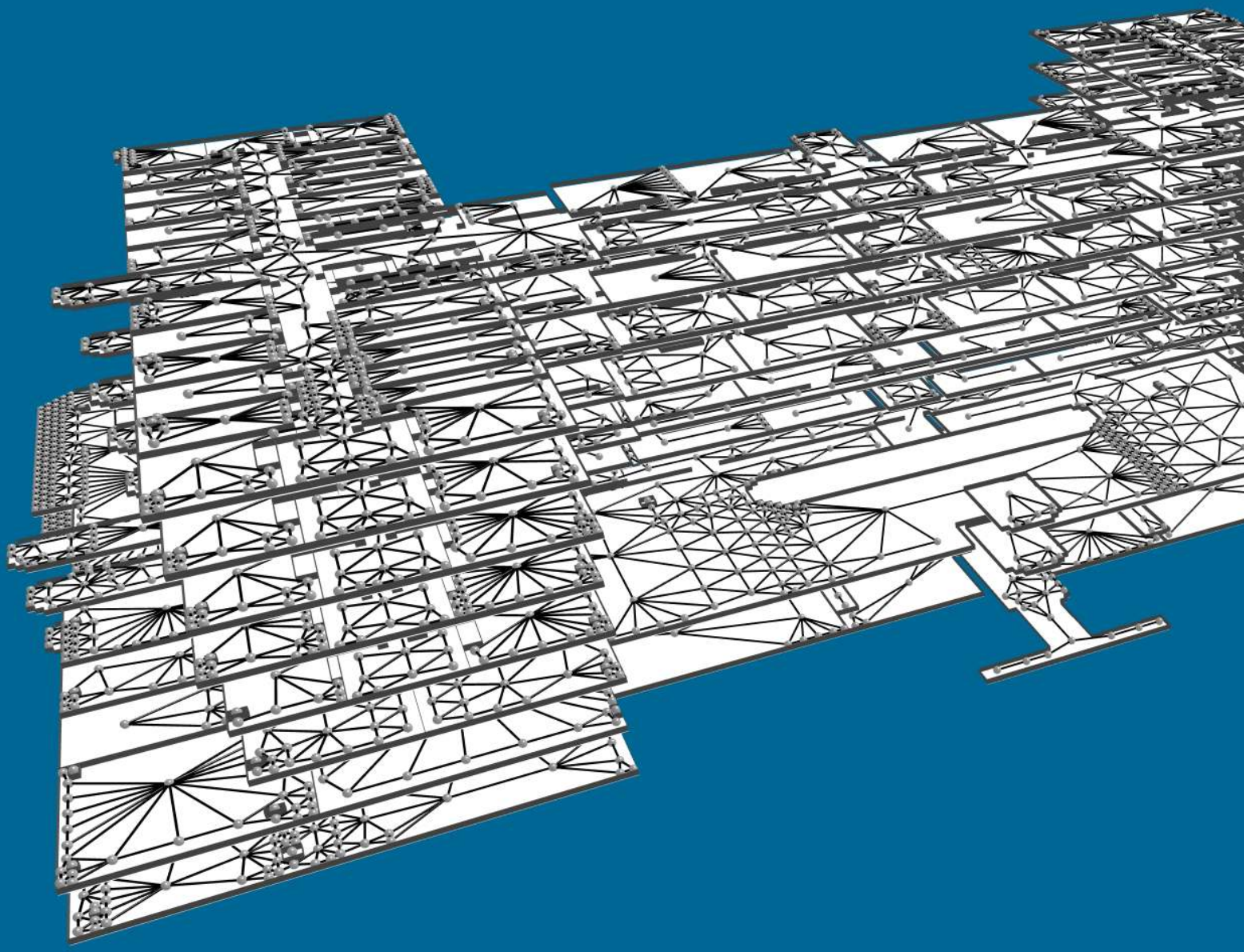
[Zad73] Norman Zadeh. **A Bad Network Problem for the Simplex Method and Other Minimum Cost Flow Algorithms**. *Mathematical Programming*, 5(1): 255–266, 1973.

[zet14] zet development team. **ZET – zet evacuation tool**. 2014. `http://www.zet-evakuierung.de/`.

[Zwi95] Uri Zwick. **The Smallest Networks on which the Ford-Fulkerson Maximum Flow Procedure may Fail to Terminate**. *Theoretical Computer Science*, 148, 1995.

Curriculum Vitae

Juni 1983:	Geboren in Herten, Westf.
1993 – 2002:	Besuch des Städtischen Gymnasiums in Herten
Juli 2002:	Abitur
2002 – 2003:	Zivildienst
2003 – 2009:	Studium der Informatik und der Mathematik an der Technischen Universität Dortmund
Oktober 2005:	Vordiplom in Informatik („gut“)
2006 – 2007:	Auslandsaufenthalt an der Università degli Studi di Verona, Italien
März 2008:	Vordiplom in Mathematik („sehr gut“)
Juli 2009:	Diplom in Informatik („mit Auszeichnung“)
2009 – 2014:	Wissenschaftlicher Mitarbeiter an der Technischen Universität Berlin
Dezember 2014:	Promotion zum Dr. rer. nat. („sehr gut“) Betreuer: Prof. Dr. Martin Skutella



ISBN 978-3-7375-3238-9