Evacuation by Earliest Arrival Flows

Dipl.-Math. oec Nadine Baumann
aus Potsdam

Vom Fachbereich Mathematik
der Universität Dortmund
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss
Berichter: Prof. Dr. Martin Skutella
Prof. Dr. Ekkehard Köhler

Tag der Disputation: 18. Januar 2007

Contents

Introduction

Considering the last few years, the number of evacuations of areas endangered by tsunamis or hurricanes, of airplanes having engine turbine problems, of buildings on fire, as well as of evacuations because of bomb alarms has highly increased. The increasing number of emergencies naturally increases the interest in an optimal preparation of the evacuation before the emergency occurs. In particular, the heavy interest of the media in the evacuation test of the newly build airplane Airbus A380 (see for example BBC News Online [7] and Spiegel Online [76]) strengthens the impression that the role of evacuation planning and prediction becomes more important, also in public. While the importance of good evacuation prediction became obvious for the public during the last few years, practitioners and theoreticians were searching for methods to find good evacuation plans for buildings, airplanes, or whole areas for the case of an emergency for a long time.

Besides the large amount of simulation tools for evacuation prediction under certain assumptions (see for example [48, 64, 52]), evacuation problems can be modeled as network flow problems. In contrast to static flows, flows over time introduced by the seminal work of Ford and Fulkerson [24, 25] explicitly model the impact of time, which is an important factor in evacuation situations. In this extended model, *transit times* on arcs indicate the loss of time when traversing a site from one end to the other. Although network flows over time are not capable to map real world behavior exactly to mathematical models, they offer a good tool to predict the evacuation behavior in existing buildings, airplanes, or areas and to plan new buildings and airplanes with respect to good evacuations.

Given a seat map of an airplane or the locations of working stations in a building together with the building topology, we can model the network. If we can find an optimal evacuation plan in a *perfect* flow model where passengers are considered to act rational without panic, the evacuation time we determine is a lower bound on the time the evacuee would need. Moreover, it is possible to find a lower bound on the total evacuation time for example for certain seat maps already in the construction phase of an airplane. Therefore, it is possible to optimize the position of seats and emergency exits in an airplane with respect to evacuation situations.

In typical evacuation situations, the most important task is to get people

out of an airplane, an endangered building, or an area as fast as possible. Since it is usually not known how long a building can brave a fire before it collapses, how much time it needs before a smoking aircraft turbine engine really starts burning, or how long a dam can resist a flood before it breaks, it is advisable to organize an evacuation such that as much as possible is saved no matter when the inferno will actually happen. Therefore, it is not enough to only bound the total evacuation time. By sending as much flow as possible into the sinks at each point in time, the uncertainty of the planned evacuation time is taken into consideration. While solutions to the *quickest flow problem* minimize the total evacuation time, so called *earliest arrival flow problems* aim at optimizing the evacuation process for every point in time. Those and some related flow over time problems such as maximum flows over time are usually referred to as *Evacuation Problems* (see [35, 36, 38, 39]).

Speaking of earliest arrival flows, two main settings are taken into account. On the one hand, the *earliest arrival transshipment problem* is considered. This network flow problem models especially the airplane evacuation. We know exactly where people are located in a given fixed network topology. Also office buildings with a known amount of working stations fulfill this requirement. The task is to guide them out of the airplane or building as quickly as possible. On the other hand, the *earliest arrival s-t-flow problem* is analyzed. Earliest arrival s-t-flows consider the easier model with only one source and one sink but do not give a bound on the number of people to evacuate. In this thesis, we consider these two problems and focus on efficient algorithms for solving them. Further, we do not longer restrict ourselves to evacuation of people but consider the evacuation of data packages in an unstable network.

Earliest Arrival Transshipments. Given a network with multiple sources and multiple sinks with assigned supplies and demands, the quickest transshipment problem asks for the minimum time horizon up to which the supplies and demands can be satisfied over time. This problem can be solved in strongly polynomial time. The strongly related earliest arrival transshipment problem, which additionally maximizes the amount of flow sent into sinks simultaneously for each time $\theta \geq 0$, does not necessarily need to exist in this kind of networks. In order to solve the later problem, we need to restrict to multiple-sources-single-sink networks. Hoppe and Tardos [39] develop an FPTAS for the earliest arrival transshipment problem. Other approaches to solve the earliest arrival transshipment problem strictly restrict the transit times or the capacities of the considered networks. The first exact algorithm to solve the earliest arrival transshipment problem for the multiple-sources-

single-sink setting that runs in time polynomial in the input plus output size is derived in Chapter 3. Using the necessary and sufficient criterion for the feasibility of transshipment over time problems presented by Klinz [50], the earliest arrival pattern can be recursively constructed. In a second step, the earliest arrival pattern is turned into an earliest arrival transshipment by slightly extending the network and applying an algorithm of Hoppe and Tardos [40] that computes a quickest transshipment in strongly polynomial time. In particular, the algorithm to compute an earliest arrival transshipment as presented in Chapter 3 is the first algorithm for this problem which does not rely on time-expansion of the network into exponentially many time layers, even in the analysis.

Earliest Arrival s-t-Flows. Given a network with a single source node s, a single sink node t, and a time horizon $T \geq 0$, Ford and Fulkerson [24] (see also [25]) consider the problem of sending as much flow as possible from s to t by time T; the *maximum s-t-flow over time problem*. Gale [28], Wilkinson [77], and Minieka [65] analyzed the problem of maximizing the amount of flow that reached the sink by every time $0 \leq \theta < T$; the *earliest arrival s-t-flow problem*. The above approaches assume transit times on the arcs to be constant or time-dependent. Whoever has tried to leave a stadium after a soccer game knows that this assumption is far from reality for some real world applications like evacuation or traffic routing. A much more realistic setting is to view transit time as a value that depends on the flow rate, the congestion, or the amount of flow in an arc of the network. In particular, this means that the more flow units are present in an arc the higher is the transit time of this arc. More formally, the transit time of each arc in the network at each time θ depends on the flow on this particular arc at that time θ; so called *flow-dependent transit times*. Two special models of flow-dependent transit times are considered in this thesis; *inflow-dependent transit times* and *load-dependent transit times*. It has been shown by Gale that earliest arrival s-t-flows exist for any network with constant transit times. We give examples in Chapter 4, showing that this is no longer true for flow-dependent transit time functions. Here, we restrict ourselves to inflow-dependent and load-dependent transit times. For that reason, we define an optimization version of this problem where the objective is to find flows that are almost earliest arrival s-t-flows. In particular, we are interested in flows that, for each $\theta \in [0, T)$, need only α-times longer to send the maximum amount of flow to the sink. Both constant lower and upper bounds on α are given. Furthermore, we present a constant factor approximation algorithm for this problem. Finally, we give some computational results to show the practicability of the designed approximation algorithm.

Data Evacuation. Evacuation planning does not only play an important role in human evacuation but also in evacuating data sent through an unstable network via several processors. Data in the network can be seen as insecure in the sense that it gets lost immediately in case of a disruption. Data that is saved in a processor can be seen as secure, since it can be resent after the disruption. The property that data can be copied is used in order to save copies of data into the processors. The storage capacity of processors is bounded and therefore the storage of data needs to be examined. In Chapter 5, we define the problem and show that we have to concentrate only on the storage rules. Optimal storage rules are determined for special path topologies: *transit time excess paths* and *capacity excess paths*. Optimality means that the maximum number of different data packages is stored when considering all nodes of the network. Moreover, it is required that the data packages stored shall be renewed in the sense that there is a steady exchange in the stored data packages in nodes. There also exist an optimal storage rule for arbitrary paths. It can be shown that arbitrary paths are a compound of transit time excess paths and capacity excess paths.

Outline of the Thesis. We define basic notation, introduce network flow models known from literature and give known results in Chapter 1. Moreover, we introduce to two mathematical concepts needed in this thesis; submodular functions and parametric search. In Chapter 2, we formally introduce several evacuation problems considered in this thesis. In particular, we concentrate on known results for the quickest transshipment problem and earliest arrival flow problems. Further, we overview other methods of evacuation prediction. An exact algorithm for the earliest arrival transshipment problem is presented in Chapter 3. Before describing the algorithm in detail, we show how to transform networks with supplies and demands given as upper and/or lower bounds into equivalent networks having constant supplies and demands. In Chapter 4, we study the simpler earliest arrival s-t-flow problem for the case that the transit time of each arc in the network at each point in time θ depends on the flow on this particular arc at the considered time θ. In Chapter 5, we extend the concept of evacuation to non-physical commodities such as data. We consider only networks building a path and show that it is possible to store the maximum number of different data packages along the path, respecting the capacity of storage space. For each possible path topology, we determine storage rules that define which data needs to get stored in which node.

This thesis tries to be self-contained in the sense that models and algorithms used are introduced in detail. Nevertheless, the reader should be familiar with basic graph notation, graph algorithms, optimization problems, and complexity of algorithms. Moreover, a basic knowledge in classical network flow theory is advantageous. For a deeper introduction on network flow models, the reader is referred to the textbooks on combinatorial optimization in general and network flow theory in particular of Ahuja et al. [1], Korte and Vygen [56], and Schrijver [72]. A more detailed introduction in flow over time models is given in the Ph.D thesis of Hoppe [38] and in the survey articles of Aronson [3] and Powell et al. [68]. Several articles of Fleischer, Tardos, and Skutella ([19, 18, 17, 23, 22, 21, 20]) describe special network flow over time problems in detail. A very good introduction to inflow- and load-dependent transit times is given in the Ph.D thesis of Langkau [57] and articles by Köhler, Langkau, Hall, and Skutella ([55, 54, 33]). Substantial parts of Chapters 3 and 4 of this thesis are already published in [5, 4, 6].

Acknowledgment

Over all, I thank Martin Skutella for evoking interest in new topics of network flow theory, for his help to find new points of view to the considered problems, and for supervising me. I am grateful to the German Science Foundation for the position in the project "Algorithms in Large and Complex Networks" which granted my financial support (grants no. SK 58/4-1 and SK 58/5-3). Following Martin from Berlin to Saarbrücken and later to Dortmund, I had the privilege to work in Prof. Kurt Mehlhorn's working group at the Max-Planck-Institute for Computer Science and in his own working group at Dortmund University. Furthermore, I was always welcome to the working group of Prof. Rolf Möhring at Technical University Berlin and thank him for his hospitality. In particular, it was a pleasure to work together with Ekkehard Köhler and Sebastian Stiller.

Especially, I thank Ekkehard Köhler for his willingness to take the second assessment to this thesis although he is occupied with many other things.

I would like to thank Sebastian Stiller, Joachim Reichel, and Eric Berberich for their careful and helpful proof-reading and Ingo Schulz for his help producing some of the practical results. Further, I thank Nicole Megow for being in the same situation and having the same problems.

Finally, I thank all members of the research groups of Prof. Martin Skutella at Dortmund University, of Prof. Kurt Mehlhorn and NWG2 at MPII Saarbrücken, and of Prof. Rolf Möhring at TU-Berlin for all the coffee breaks and for being more than just colleges.

Chapter 1

Preliminaries

1.1 Introduction

Network flows are an important topic in combinatorial optimization. Not only evacuation settings can be modeled and optimized using network flows but also transportation problems, telecommunication flows, problems of creating timetables, financial flows, and so on.

We define a *network* $\mathcal{N} = (V, A)$ to be a directed graph consisting of a set of nodes V and a set of directed arcs $A \subseteq V \times V$. We assign a *capacity* $u(a)$ to each arc determining an upper bound on the amount of flow actually on that arc. Further, we assign arc a a *transit time* $\tau(a)$ representing the time that is needed to traverse the arc. Two concepts of transit times will be considered: constant transit times and transit time functions dependent on flow. Moreover, two specified node sets are considered; *sources* and *sinks*. The set of sources is denoted by S^+ and the set of sinks by S^-. The remaining nodes $v \in V \setminus (S^+ \cup S^-)$ are called *intermediate nodes*. Sometimes we are also given a *supply-demand function* $d : S^+ \cup S^- \to \mathbb{R}$. It assigns a positive value, supply $d(s) > 0$, to each source $s \in S^+$ and a negative value, demand $-d(t) > 0$, to each sink $t \in S^-$.

Considering evacuation settings, we can observe that each site to evacuate can be represented as such a network. Nodes model rooms in a building, seats in an airplane, crossings of floors, or positions in the aisle of an airplane and so on. Arcs connect locations represented by nodes. They represent the aisle of an airplane, halls, floors, and ways from one room to another in a building. Sources in such a network are sites at which people stay like the seats of an airplane, sleeping rooms of an apartment house, or working stations in an office building. Sinks model safe sites outside the place to evacuate.

In this chapter, we will introduce the notion of network flow models in general. Special network flow models related to evacuation problems are considered in detail. We start by the well studied static flow model in Section 1.2.1. In Sections 1.2.2 and 1.2.3, we consider several models for flows over time. In this flow model, each arc is given a transit time. Flow is sent through the network continuously and can change its value on an arc over

time. Notice that in earlier work, flows over time were also called dynamic flows. During the last years, the notion *dynamic* became important in problems where input changes over time or arrives over time. Algorithms solving those problems need to deal with the dynamic input and aim to readjust the solutions to the new information available. The data needed to solve a flow over time problem is available from the beginning. In order to avoid misunderstanding, we always speak of flows over time. This seems to be more intuitive to us and more clear in the parlance of network flows. In Section 1.2.4, we describe an even more realistic model of network flows over time. In this model, transit times change with the flow on an arc, i.e., they are flow-dependent. We will discuss two models of flow-dependent transit times; *inflow- and load-dependent transit times*.

The last two sections of this chapter consider elaborate mathematical concepts which are relevant in this thesis. In Section 1.3, we give a short introduction to submodular functions. In Section 1.4, we briefly introduce the parametric search. This search method finds the optimum value for a given parameter in an optimization problem in strongly polynomial time.

1.2 Network Flow Models

In the following sections, we describe basic properties of the presented flow models. Further, we present results from the literature for two flow problems in the corresponding flow models which form substantial building blocks of the results given in this thesis. In the *s-t-flow problem* we are given a single-source-single-sink network. The task is to find a feasible flow from source s to sink t. If we are given a multiple-sources-multiple-sinks network together with a supply-demand function $d : S^+ \cup S^- \to \mathbb{R}$, we will focus on the *transshipment problem*. A transshipment is a flow that fulfills the supplies and demands of sources and sinks, respectively. The task is to find such a flow, if it exists in the considered network flow model.

1.2.1 Static Flows

Given a network $\mathcal{N} = (V, A)$, a function $x : A \to \mathbb{R}^+$ is called a *static flow function*, if the *capacity constraints* are satisfied, i.e., it holds $x(a) \leq u(a)$ for all $a \in A$. A static flow x is said to observe *flow conservation* in node v if

$$\sum_{a \in \delta^-(v)} x(a) = \sum_{a \in \delta^+(v)} x(a)$$

holds. Here $\delta^+(v)$ and $\delta^-(v)$ denote the set of outgoing arcs of node v and incoming arcs into node v, respectively. A flow satisfying flow conservation in all nodes is called a *circulation*. Considering an s-t-flow problem, flow conservation only has to hold for nodes in $V \setminus \{s,t\}$. An s-t-flow fulfilling the required flow conservation equations for nodes in $V \setminus \{s,t\}$ and capacity constraints for all arcs is called *feasible*.

The *value* of an s-t-flow x is given by

$$\begin{aligned} \text{value}(x) &:= \sum_{a \in \delta^+(s)} x(a) - \sum_{a \in \delta^-(s)} x(a) \\ &= \sum_{a \in \delta^-(t)} x(a) - \sum_{a \in \delta^+(t)} x(a) \ , \end{aligned}$$

where the second equation follows from flow conservation in all nodes $v \in V \setminus \{s,t\}$. An s-t-flow with maximum value is called a *maximum s-t-flow*.

Let $X \subsetneq V$ be a subset of the nodes. Then we say that X is an *s-t-cut*, if $s \in X$ and $t \in V \setminus X$. The *capacity* u of an s-t-cut X is defined as follows:

$$u(X) := \sum_{a \in \delta^+(X)} u(a) \ .$$

Here, $\delta^+(X)$ denotes the set of directed arcs from nodes in X to nodes in $V \setminus X$. If the capacity $u(X)$ is minimal over all sets $X \subsetneq V$, then we call X a *minimum s-t-cut*.

The following theorem, called the *max-flow min-cut theorem*, states a fundamental relation between the value of a maximum s-t-flow and the capacity of a minimum s-t-cut.

Theorem 1.1 (Ford and Fulkerson [24]). For any network, the maximal flow value from s to t is equal to the minimum cut capacity of all cuts separating s and t.

Ford and Fulkerson [24] suggest an algorithm to compute a maximum s-t-flow. Before describing the algorithm, we give some useful definitions.

Given a network $\mathcal{N} = (V, A)$ and a feasible flow x, then the *residual network* $\mathcal{N}_x = (V, A_x)$ is defined as follows. Let $a = (v, w)$ be an arc in A. If $x(a) < u(a)$, then $a \in A_x$. We set the *residual capacity* of a to $u_x(a) := u(a) - x(a) > 0$. If $x(a) > 0$, we also insert the backward arc $\overleftarrow{a} := (w, v)$ to A_x having a residual capacity $u_x(\overleftarrow{a}) := x(a)$. If each arc a is given a certain cost $\tau(a) \in \mathbb{R}$ then the negative of this cost, namely $-\tau(a)$ is assigned to the backward arc. A path from s to t in the residual network with respect to a flow x is called an *augmenting s-t-path*.

The algorithm to compute the maximum s-t-flow x starts with the zero flow, i.e., $x(a) = 0$ for all $a \in A$. Then it searches for augmenting s-t-paths as long as they exist in the residual networks with respect to the actual flow x. Along such an s-t-path, flow amounting to the minimal residual capacity γ is augmented, i.e., we increase $x(a)$ by γ, if a is a forward arc on the path, and we decrease $x(a)$ by γ, if its backward arc $\overleftarrow{a}$ is part of the path. Since γ is chosen as the minimal residual capacity along this path, we guarantee capacity constraints and non-negativity constraints $x(a) \geq 0$ for all $a \in A$. Unfortunately, the total number of paths found can be exponentially large.

An s-t-flow $x : A \to \mathbb{R}^+$ defines a flow on arcs. Since the overall goal is to optimize an evacuation problem, we need to determine paths along which people can leave a site to evacuate. Therefore it is advantageous, if we can give a formulation of the flow on a set of paths $\mathcal{P}$ from s to t. It is well known that the flow function x defined on arcs can be decomposed into a flow on at most $|A|$ paths $\mathcal{P}$ and cycles $\mathcal{C}$ in $\mathcal{N}$ together with non-negative flow values $x(P)$ for $P \in \mathcal{P} \cup \mathcal{C}$. For those values on paths and cycles, the following property has to hold:

$$x(a) = \sum_{\substack{P \in \mathcal{P} \cup \mathcal{C}: \\ a \in P}} x(P) \qquad \text{for all } a \in A.$$

We call the set $\mathcal{P}$ together with the flow values $x(P)$ for $P \in \mathcal{P}$ a *path decomposition*, if $\mathcal{C} = \emptyset$. Algorithmically, the path decomposition can be found by using the algorithm for finding a maximum s-t-flow backwards. Therefore, we only consider the network consisting of arcs $a \in A$ having a positive value $x(a)$. In this restricted network $\mathcal{N}'$, we set the capacity $u'(a) := x(a)$ and apply the algorithm to find a maximum s-t-flow in $\mathcal{N}'$. Thereby, at least one arc is expunged from the relevant network $\mathcal{N}'$ in each iteration and therefore maximal $|A|$ many paths can be found. The resulting set of paths together with the corresponding minimal residual capacity builds the path decomposition. Flow that is still on cycles can be ignored since it does not increase the amount of flow sent from node s to t.

Another important static flow problem, which will be needed throughout this thesis, is the *min-cost circulation problem*. Assume we are additionally given a cost function $\tau : A \to \mathbb{R}$ on the arcs. The cost of a circulation x is defined as:

$$\text{cost}(x) := \sum_{a \in A} \tau(a)x(a) \ .$$

The min-cost circulation problem looks for a feasible circulation x having minimum cost $\text{cost}(x)$. In the following, we describe how to solve the min-cost circulation problem to optimality in network $\mathcal{N}$. Therefore, we use the optimality criterion of Klein which suggests an algorithm.

Theorem 1.2 (Klein [49]). A flow x is a min-cost circulation if and only if there is no directed cycle C in the residual network $\mathcal{N}_x$ such that the sum of the cost around C's arcs is negative. A directed cycle is a sequence of distinct directed arcs of the form $\{(v_0, v_1), (v_1, v_2), \ldots, (v_k, v_0)\}$ involving distinct nodes.

The naturally induced algorithm, of course, does the following. To compute the min-cost circulation, we start in network $\mathcal{N}$ with the zero flow. By successively searching for cycles having negative cost as long as they exist in the corresponding residual networks, we increase the flow on those cycles. Once we find such a cycle, we augment flow of value of the minimal residual capacity γ along the cycle.

If flow on cycles of zero length is added to the circulation, the cost does not increase but the amount of flow in the network increases. We will call a min-cost circulation that maximizes the amount of flow in the network *min-cost (maximum) circulation*.

Zadeh [78] gives bad examples for this algorithm for which the algorithm indicated by Klein has an exponential running time in worst case. The algorithm requires $2^n + 2^{n-2} - 2$ augmentations in a network having $2n + 2$ nodes for $n \in \mathbb{N}$. The minimum mean cycle-canceling algorithm presented by Goldberg and Tarjan [29] has a strongly polynomial running time. A survey on the complexity of min-cost circulation algorithms can be found in the book of Schrijver [72].

A related problem is the problem of finding a *min-cost s-t-flow*. There the goal is to compute a min-cost circulation in the network extended by an arc (t, s) with specified cost $\tau(t, s)$. There can be several circulations having minimum cost in such a network. Depending on the problem, we are sometimes seeking the one with highest flow value on arc (t, s); the *min-cost (maximum) s-t-flow*. Here, flow on zero length cycles needs to be added. Notice that the cost of such a circulation stays the same while the amount of flow in the network increases.

We define the *cost of the min-cost s-t-flow* x dependent on the cost of the additional arc (t, s):

$$\begin{aligned}\text{cost}^{\tau(t,s)}(x) &:= \sum_{a \in A \cup \{(t,s)\}} \tau(a)x(a) \\ &= \sum_{a \in A} \tau(a)x(a) + \tau(t,s)x(t,s) \ . \end{aligned} \tag{1.1}$$

Another problem related to evacuation is the *transshipment problem* where we are given multiple sources and multiple sinks in the network. The supply-demand-function $d : S^+ \cup S^- \to \mathbb{R}$ assigns a positive supply to each source in S^+ and a negative demand to sinks in S^-. Considering a transshipment problem, flow conservation has to hold only for nodes $w \in V \setminus S^+ \cup S^-$. For nodes $v \in S^+ \cup S^-$ the flow function x has to satisfy the supplies and demands, that is

$$\sum_{a \in \delta^+(v)} x(a) - \sum_{a \in \delta^-(v)} x(a) = d(v) \ .$$

Such a transshipment problem in a network with multiple sources and multiple sinks can be reduced to an s-t-flow problem by a slight network transformation. We add a *supersource* s with arcs (s, s') for all $s' \in S^+$ and assign a capacity of $d(s')$ to this arc. For each sink $t' \in S^-$ we insert arcs (t', t) to a new sink node t which will be called the *supersink*. We assign a capacity of $-d(t')$ to those arcs. A feasible s-t-flow with value $\sum_{s' \in S^+} d(s')$ in the modified network naturally induces a feasible flow in the original network satisfying all supplies and demands. Each maximum s-t-flow obviously has the demanded flow value. Thus, when considering static flow models, we can restrict ourselves to networks with a single source and a single sink.

1.2.2 Flows over Time with Constant Transit Times

In various well known static flow problems one seeks a function $x : A \to \mathbb{R}^+$ that assigns a flow value $x(a)$ to an arc a. In contrast to that, a *flow over time* f is a function on $A \times \mathbb{R}$ where the second parameter denotes the time component – $f(a, \theta)$ describes the flow on arc a at time θ. This flow can be interpreted as flow rate, i.e., the amount of flow entering the particular arc per time unit. The flow rate entering an arc is bounded by the capacity of that arc, i.e., $f(a, \theta) \leq u(a)$. Considering flows over time, we need to introduce a *time* component into the network itself. Therefore, we assign a constant transit time $\tau(a)$ to each arc $a = (v, w)$. This value determines the time that flow needs to traverse the arc from the start node v to the target node w. Moreover, sending one unit of flow into the arc at flow rate one at time θ means, that at time $\theta + \tau(a)$ flow of rate 1 enters the target node

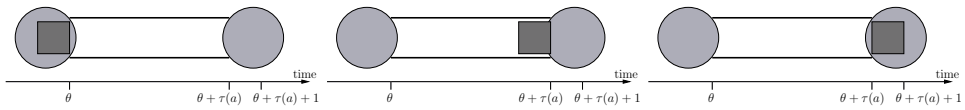


Figure 1.1: Arc having a capacity of one unit of flow per time unit. Each unit of flow entering that arc at time θ starts leaving the arc at time $\theta + \tau(a)$. It totally has left the arc at time $\theta + \tau(a) + 1$.

of the arc. The whole flow unit has left the arc another time unit later at time $\theta + \tau(a) + 1$ (see Figure 1.1 for clarification).

The flow function f has to satisfy flow conservation constraints not only in every node $w \in V \setminus (S^+ \cup S^-)$, but also at every time $\theta \in \mathbb{R}$. Notice that the flow rate $f(a, \theta)$ determines the amount of flow entering arc a at time θ. The flow rate entering node w at time θ from an arc $a \in \delta^-(w)$ is thus determined by the flow rate entering that arc at time $\theta - \tau(a)$. Therefore, the flow conservation constraint in node w is defined as follows:

$$\sum_{a \in \delta^+(w)} \int_{-\infty}^{\theta} f(a, \theta) d\theta \leq \sum_{a \in \delta^-(w)} \int_{-\infty}^{\theta} f(a, \theta - \tau(a)) d\theta \ .$$

Equality is not required in order to allow flow to be stored in nodes of the network. Note that some algorithms for flows over time explicitly forbid storage of flow in nodes whereas others only have a bounded capacity of node storage. We allow storage of flow at nodes in principle and without upper bounds on the amount. However, the algorithms used in this thesis do not make use of the storage opportunity; they send the flow through the network without storing flow at any of the non-source nodes.

A flow over time f, satisfying the flow conservation constraints in all required nodes, is said to be *feasible*, if for all arcs a and every time θ: $f(a, \theta) \leq u(a)$ holds. For a given *time horizon* T, we further require that there is only flow in the network during the time interval $[0, T)$. Since flow entering arc $a = (v, w)$ at time θ reaches node w at time $\theta + \tau(a)$, we especially require $f(a, \theta) = 0$ for all $\theta \notin [0, T - \tau(a))$. This guarantees that there is no flow in the network before time zero and that at time T all flow has left the network. The flow conservation constraint of each node w needs to be satisfied only within this time interval:

$$\sum_{a \in \delta^+(w)} \int_{0}^{T} f(a, \theta) d\theta \leq \sum_{a \in \delta^-(w)} \int_{0}^{T} f(a, \theta - \tau(a)) d\theta \ .$$

Notice that it would not be necessary to subtract the transit time of the considered arc in the set of all incoming arcs. By the assumption that $f(a, \theta) = 0$

for $\theta \notin [0, T - \tau(a))$ the compared values stay the same if we require:

$$\sum_{a \in \delta^+(w)} \int_0^T f(a, \theta) d\theta \leq \sum_{a \in \delta^-(w)} \int_0^T f(a, \theta) d\theta \ .$$

An s-t-flow over time naturally requires the flow conservation only in nodes in $V \setminus \{s, t\}$. The *value* of an s-t-flow over time f with time horizon T is defined by the net flow value that leaves the source over all time steps or enters the sink over all time steps $\theta \in [0, T)$.

$$\begin{aligned} \text{value}(f) &= \sum_{a \in \delta^+(s)} \int_0^T f(a, \theta) d\theta - \sum_{a \in \delta^-(s)} \int_0^T f(a, \theta) d\theta \\ &= \sum_{a \in \delta^-(t)} \int_0^T f(a, \theta) d\theta - \sum_{a \in \delta^+(t)} \int_0^T f(a, \theta) d\theta \end{aligned}$$

If the value is maximized, we speak of a maximum s-t-flow over time. In the following, we will denote the value of a maximum flow out of source s reaching sink t for a time horizon T also by $o^T(\{s\})$, if we are not considering a concrete flow f.

More formally, the function $o^\theta(X)$ determines the maximum value of flow over time that can be sent out of sources in X to sinks not in X by time $\theta \in \mathbb{R}^+$. Since s is the single source and t the single sink:

$$o^T(\{s\}) = \max\{\text{value}(f) | f \text{ feasible } s\text{-}t\text{-flow with time horizon } T\}.$$

This function is used throughout this thesis for the different settings.

Various results for this flow over time model with constant transit times are known. One of the most remarkable ones is a theorem by Ford and Fulkerson [24] together with an argument for continuity by Anderson and Philpott [2] on maximum s-t-flows over time for the time horizon T.[1] Ford and Fulkerson introduce a special class of flows over time, which resembles static flows. They define a *temporally repeated flow* to be a static flow x that is decomposed into flows on paths in network $\mathcal{N}$. Let $\mathcal{P}$ be the set of paths and $x(P)$ the non-negative flow values on paths $P \in \mathcal{P}$ such that the length of each path is bounded from above by T. A temporally repeated flow repeatedly sends flow over the paths of $\mathcal{P}$ as long as this flow can reach the sink before time T, i.e., it starts sending flow at time zero and stops at time $T - \tau(P)$. Obviously, such a flow is a feasible flow over time. It naturally obeys flow

[1] Note that Ford and Fulkerson considered a discrete flow model where time is being discretized into points of time $\{0, 1, \ldots, T - 1\}$.

conservation and capacity bounds by the nature of the path decomposition and the feasibility of the static flow x, namely $\sum_{P\in\mathcal{P}:a\in P} x(P) = x(a) \leq u(a)$. By definition, no flow is in the network before time zero and after time T.

The following lemma determines the value of the temporally repeated flow from s to t in terms of the static flow x.

Lemma 1.3. The value of a flow over time f that is computed as a temporally repeated flow of a static flow x equals:

$$\text{value}(f) = \sum_{P\in\mathcal{P}} (T - \tau(P))x(P) = T \cdot \text{value}(x) - \sum_{a\in A} \tau(a)x(a) \ . \tag{1.2}$$

Observe that the flow value is independent of the path decomposition $\mathcal{P}$. One natural goal is now to find a temporally repeated flow having maximum flow value. The existence of such a kind of maximum s-t-flow over time is given by Ford and Fulkerson [24] (see also [2]).

Theorem 1.4 (Ford and Fulkerson [24]). There always exists a temporally repeated flow that is a maximum s-t-flow over time. Such a temporally repeated flow can be computed using a static min-cost s-t-flow.

Notice that the value of the maximum s-t-flow over time for time horizon T, i.e., $o^T(\{s\})$, determined as in in (1.2), equals $-\text{cost}^T(x)$, i.e., the negative of the cost of a static min-cost circulation in network $\mathcal{N}$ extended by an uncapacitated arc (t, s) with cost $-T$ (compare equation 1.1). Here, x denotes the static min-cost s-t-flow computed in network $\mathcal{N}$ where the transit times are interpreted as costs. The choice of $-T$ as cost of the additional arc bounds the length of the used paths by T and therefore the time horizon is obeyed. Further, we know that the static flow x and the flow value value(x) are integral if all capacities are integral.

Analogously to the static case, we define a *transshipment over time* to be a flow in a multiple-source-multiple-sink network where given supplies and demands now have to be satisfied over time. Flow conservation only has to hold for nodes $w \in V \setminus (S^+ \cup S^-)$. For nodes $v \in S^+ \cup S^-$ the outgoing and incoming flow over time has to equal the supply and demand, respectively, i.e.,

$$\sum_{a\in\delta^+(v)} \int_0^T f(a,\theta)d\theta - \sum_{a\in\delta^-(v)} \int_0^T f(a,\theta)d\theta = d(v).$$

Discrete Flows over Time. The above described flow model is also called *continuous* flow over time model. When Ford and Fulkerson introduced the model of flows over time, they first considered a discrete model. Instead of

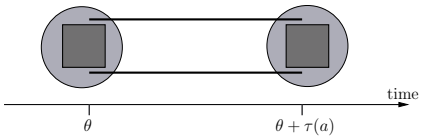


Figure 1.2: Arc with transit time $\tau(a)$ in a discrete flow over time model. Flow units leaving a node via arc a at time $\theta \in \{0, 1, \dots, T - 1 - \tau(a)\}$ totally have left arc a at time $\theta + \tau(a)$.

sending flow at (continuous) flow rates, they send packets of flow units at discrete points of time into the arcs. In this model, not the continuous time interval $[0, T)$, but all discrete time steps $\{0, 1, \dots, T - 1\}$ are considered. That means we are looking for a flow function $f : A \times \{0, 1, \dots, T - 1\} \to \mathbb{R}^+$. The flow rate $f(a, \theta)$ on arc $a = (v, w)$ determines the amount of flow that is sent into arc a at time step $\theta \in \{0, 1, \dots, T - 1\}$. Flow units sent into an arc a at a time θ totally reach the target node w of that arc at time $\theta + \tau(a)$. Here, $\tau(a)$ denotes the transit time of arc a (see Figure 1.2 for a better understanding of the difference to continuous flows). In such a model, obviously, all transit times need to be integral. Consequently, it suffices to consider integral time horizons.

Flow conservation can be adapted directly to the discrete case. Instead of integrating, it suffices to sum over the considered time steps $\{0, 1, \dots, T-1\}$. Thus, flow conservation in node v is obeyed in the discrete model, if

$$\sum_{a \in \delta^+(v)} \sum_{\theta=0}^{T} f(a, \theta) \leq \sum_{a \in \delta^-(v)} \sum_{\theta=0}^{T} f(a, \theta)$$

holds.

In the discrete model, the flow value of an s-t-flow over time can be described by summing up the flow leaving the source s in each time step or entering the sink t in each time step, i.e.,

$$\begin{aligned} \text{value}(f) &= \sum_{a \in \delta^-(t)} \sum_{\theta=0}^{T} f(a, \theta) - \sum_{a \in \delta^+(t)} \sum_{\theta=0}^{T} f(a, \theta) \\ &= \sum_{a \in \delta^+(s)} \sum_{\theta=0}^{T} f(a, \theta) - \sum_{a \in \delta^-(s)} \sum_{\theta=0}^{T} f(a, \theta) \ . \end{aligned}$$

If supplies and demands are given, they have to be fulfilled for all nodes $v \in$

$S^+ \cup S^-$, i.e.,

$$\sum_{a \in \delta^+(v)} \sum_{\theta=0}^{T} f(a,\theta) - \sum_{a \in \delta^-(v)} \sum_{\theta=0}^{T} f(a,\theta) = d(v)$$

must hold.

As indicated before, many results made for discrete flows over time can be generalized to continuous flows over time. A similar interrelation can be made about the two flows itself. Assume we are given a feasible discrete flow over time f for time horizon T with flow $f(a,\theta)$ entering arc $a \in A$ at time $\theta \in \{0,\dots,T-1-\tau(a)\}$. This discrete flow over time can be interpreted as a continuous flow over time f' by sending flow of rate $f(a,\theta)$ into arc a during the whole time interval $[\theta,\theta+1)$, i.e., $f'(a,\theta') := f(a,\theta)$ for $\theta' \in [\theta,\theta+1)$ and for all $a \in A$. The capacity constraints of the continuous flow over time are obviously obeyed, since $f(a,\theta) \leq u(a)$ induces $f'(a,\theta') \leq u(a)$ for all $\theta' \in [\theta,\theta+1)$. This interrelation is a bidirectional one, if the time horizon and all transit times of network $\mathcal{N}$ are integral. Suppose we are given a feasible continuous flow over time f' computed in network $\mathcal{N}$ with time horizon T. Then this flow f' can also be interpreted as a discrete flow over time f. We set the flow on arc a at discrete time θ to the total flow that has entered arc a during the time interval $[\theta,\theta+1)$ in f', i.e., $f(a,\theta) := \int_\theta^{\theta+1} f'(a,\theta')d\theta'$ for all $\theta \in \{0,\dots,T-1-\tau(a)\}$ for all $a \in A$. The capacity constraints are obeyed since

$$f(a,\theta) = \int_\theta^{\theta+1} f'(a,\theta')d\theta' \leq \int_\theta^{\theta+1} u(a)d\theta' \leq u(a)$$

holds. The first inequality follows from the feasibility of the continuous flow over time f'.

1.2.3 Time-Expanded Networks

Flows over time are more complex than static flows in the sense that the flow is specified for all times $\theta \in [0,T)$. Klinz and Woeginger [51] observed, for example, that there does not exist a deterministic polynomial time algorithm that solves the min-cost s-t-flow over time problem with given supply. In this problem, we search for a flow that sends the total supply over time into the sink within a given time horizon T. The corresponding static flow problem is easy to solve as described in Section 1.2.1. If all transit times are integral[2], we

[2]In case of rational transit times scaling helps to fulfill the integrality assumption. Speaking of time-expanded network, we explicitly forbid irrational transit times.

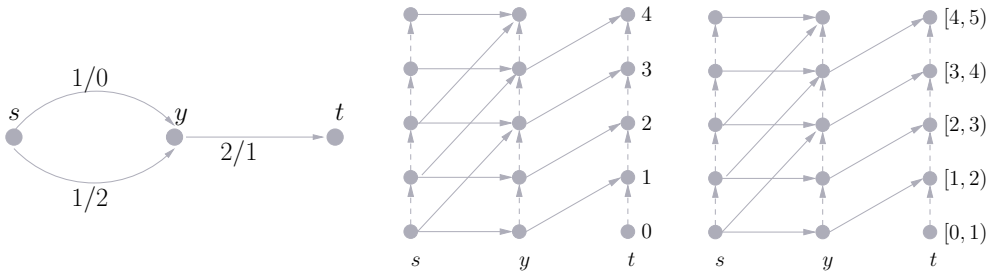


Figure 1.3: Network and its time-expanded version (discrete and continuous) for a time horizon $T = 5$, including holdover arcs. The arc notation denotes capacity/transit time.

can directly apply the algorithm for the static flow problem in an extended network. This special network acts as a static representation of the flow over time model where only integral points in time are considered. Ford and Fulkerson [24] introduced the so called time-expanded network in which the flow over time problem can be solved in pseudo-polynomial running time using static flow algorithms.

A time-expanded network with time horizon T consists of T copies of the node set V, one for each time unit. We call such a copy a *time layer*. For each arc $a = (v, w)$ of the network $\mathcal{N}$ having transit time $\tau(a)$, an arc is inserted between the copy of node v in time layer $\theta \in \{0, \dots, T - 1 - \tau(a)\}$ and the copy of node w in time layer $\theta + \tau(a)$ into the time-expanded network $\mathcal{N}^T$ (see Figure 1.3). We denote the copy of node v in time layer θ as $v(\theta)$ and the copy of node w in time layer $\theta + \tau(a)$ as $w(\theta + \tau(a))$. If we allow storage of flow in nodes, *holdover arcs* are needed that connect the copy of node v in time layer θ to the copy of node v in the next time layer. The set of holdover arcs consists of all arcs $(v(\theta), v(\theta + 1)$, for all $\theta \in \{0, \dots, T - 2\}$. One other application for the use of holdover arcs is the existence of supplies and demands at sources and sinks in the original network. In order to satisfy all supplies and demands over time, sometimes flow units have to wait before leaving the source. In such a case, holdover arcs are inserted into the time-expanded network for all sources and all sinks. The supply $d(s')$ for each source $s' \in S^+$ is assigned to node $s'(0)$ in the time-expanded network. The demand $d(t')$ for each sink $t' \in S^-$ is assigned to node $t'(T - 1)$ in the time-expanded network. Network $\mathcal{N}^T$ is obviously a static network as defined in Section 1.2.1 and the use of known algorithms for static flow problems is possible. In this network we can solve the static transshipment problem again by inserting a supersource and a supersink. The insertion of supersource and supersink works analogously to the case of static network flow models.

A flow in a time-expanded network can directly be transformed into a discrete flow over time and vice versa. Assume we are given a static flow $x(a)$ on arcs in $\mathcal{N}^T$. Each arc a in $\mathcal{N}^T$ corresponds to an arc a' in network $\mathcal{N}$. Let the start node of arc a be in time layer θ. We define a flow over time on arc a' at time θ to have flow rate $f(a', \theta) := x(a)$. This flow is feasible by construction. The interpretation of a discrete flow over time as a static flow in the time-expanded network works exactly the other way round. By the interrelation between discrete and continuous flows over time as described in the previous section, we can also interpret the flow computed in the time-expanded network as a continuous flow over time. Then, each time layer represents a time interval instead of a point in time. Static flow on an arc with target node in time layer θ is now interpreted as flow arriving at this node during the time interval $[\theta, \theta + 1)$. Analogously, static flow on an arc with start node in time layer θ is now interpreted as sent out of this node during the time interval $[\theta, \theta + 1)$. See on the right hand side in Figure 1.3 for the interpretation in the time-expanded network.

A drawback of the time-expanded network is that the size of the network depends on T. Even polynomial time algorithms for static flow problems become pseudo-polynomial, if they are applied in the time-expanded network. This is indicated by the following theorem.

Lemma 1.5. Given a network $\mathcal{N}$ with n nodes and m arcs. The time-expanded network $\mathcal{N}^T$ consists of Tn nodes and $(n + m)T - n - \sum_{a \in A} \tau(a)$ arcs where T is the considered time horizon.

Nevertheless, time-expanded networks are not only useful for problems for which no flow over time algorithm is known but they are also useful proving the correctness of a given algorithm. Ford and Fulkerson, for example, proved the correctness of their maximum s-t-flow over time algorithm by determining a cut in the corresponding time-expanded network.

1.2.4 Flows over Time with Flow-Dependent Transit Times

While flows over time are much more appropriate to model real-world situations than static flows, often the constant transit times do not model reality in a sufficiently precise way. For example, it can be easily observed that there is a high correlation between the congestion of a hallway, staircases, or an aisle in an airplane and the time needed to traverse it. A more accurate method for describing this correlation is provided by the use of *flow-dependent transit times* instead of constant transit times. For this purpose, we will assume in

the following that a transit time function $\tau : A \times \mathbb{R}^+ \to \mathbb{R}^+$ is given as a left-continuous and non-decreasing function of the flow on arcs. Here, in contrast to the model with constant transit times, we are given a second parameter that refers to the flow.

Flows over time with flow-dependent transit times have been studied before. Merchant and Nemhauser [63] suggest a model where for every arc there is both a flow-dependent cost function and a so called exit function that determines the amount of traffic that can leave the arc in dependence of the amount of flow on that particular arc. Although, their model was both nonlinear and non-convex and thus difficult to handle for efficient algorithms, it was influential for many other results in this area. Carey and Subrahmanian [11] introduce a model that uses an approach similar to time-expanded networks for the case of flow-dependent transit times. In this model, not only the arcs are expanded over time but also by several states of the transit time function.

In literature on flow dependent transit times, different options of modeling them are described. We will briefly describe two known models, that we will make use of in the sequel.

On the one hand, we consider *inflow-dependent transit times* as described in Köhler, Langkau, Skutella [54]. In this model, the transit time is a function of the rate at which the flow is entering the arc a at time θ. In particular, flow entering arc a at the flow rate $f(a, \theta)$ needs a transit time of $\tau(a, f(a, \theta))$ to traverse the arc. Hence, flow units entering an arc at the same time have the same speed (defined by the transit time of the arc) and, while traversing the arc, their speed stays constant. This transit time model allows a comparably easy description of the dependency between flow and transit times. However, it has the disadvantage that there can be situations where flow entering at a small flow rate can pass flow units that entered the same arc at a higher flow rate before—the so called *first-in-first-out (FIFO)* property is not fulfilled. What makes this model somewhat difficult to handle is, that we have to guarantee that the network is empty after time T. More precisely, when $f(a, \theta) > 0$, it must hold that $\theta + \tau(a, f(a, \theta)) < T$. The conditions on flow conservation constraints and supply/demand requirements have to be adapted to take this property into account.

On the other hand, a more precise account of transit times that depend on the flow in some real applications is to assume that the transit times depend not only on the amount of flow entering an arc but also on the amount of flow currently being in the whole arc. Köhler and Skutella [55] describe this kind of transit time function called *load dependent transit times*. In this model, the total amount of flow on an arc a at a time θ is used as the input to the transit time function τ; this amount of flow is called the *load* of the arc.

Since the flow on an arc changes continuously, also the transit time of the arc changes with each unit of flow entering or leaving the arc. Note that at each moment all units of flow on an arc have the same speed.

Although both of the above models cannot describe evacuation flows in their whole complexity, they are however capable of modeling at least some important aspects of evacuees behavior.

1.3 Submodular Functions

In Chapter 3 of this work, we consider submodular functions. Since submodular functions are of general interest, we give a definition and some properties of submodular functions. After that, we briefly survey existing algorithms for submodular function minimization.

Definition 1.6. Let $\mathcal{E}$ be a finite set. Any function $\rho : 2^{\mathcal{E}} \to \mathbb{Z}$ satisfying

$$\forall X, Y \subseteq \mathcal{E} : \rho(X) + \rho(Y) \geq \rho(X \cup Y) + \rho(X \cap Y) \tag{1.3}$$

is called *submodular function*. A function for which strict equality holds is called *modular* function.

A well known example of submodular functions is the capacity function $u : 2^V \to \mathbb{R}^+$ of an s-t-cut. This can easily be proven by checking equation (1.3). The following lemma gives some easy to check properties of submodular functions which we will need.

Lemma 1.7. Let $\rho, \phi : 2^{\mathcal{E}} \to \mathbb{Z}$ be submodular functions and $\psi : 2^{\mathcal{E}} \to \mathbb{Z}$ a modular function. Then the following holds:

1. The function $a : 2^{\mathcal{E}} \to \mathbb{Z}$ defined as $a(X) := \rho(X) + \phi(X)$ for $X \subseteq \mathcal{E}$ is submodular.
2. The function $b : 2^{\mathcal{E}} \to \mathbb{Z}$ defined as $b(X) := \rho(X) - \psi(X)$ for $X \subseteq \mathcal{E}$ is submodular.

Minimizing Submodular Functions. The first combinatorial but pseudo-polynomial time algorithm was already given in 1985 by Cunningham [16]. Here the submodular function can be minimized in time polynomial in $|\mathcal{E}|$ and $\max |\rho(X)|$.

The first polynomial time algorithm to minimize submodular functions without any side constraint was given by Grötschel, Lovasz, and Schrijver [30] in 1988. They use the Ellipsoid Method by Khachiyan [47] which solves linear programs in strongly polynomial time. This algorithm is not combinatorial

and for a long time there has not been found a combinatorial, strongly polynomial time algorithms to minimize submodular functions.

Nearly at the same time, Iwata, Fleischer, and Fujishige [43] and Schrijver [71] found combinatorial, strongly polynomial time algorithms. Both algorithms are based on the fundamental work by Cunningham [15, 16]. Iwata improved in [42] the algorithm of Iwata, Fleischer, and Fujishige to be even fully combinatorial.

A wide survey of the different minimization algorithms and several improvements is given by McCormick [59]. More details and background information on submodular functions in general are given in Fujishige [26].

1.4 Parametric Search

In his seminal paper, Megiddo [62] describes an algorithm for a search method called parametric search. We are given a parameterized instance of a certain problem, i. e., an optimization problem or a feasibility problem. Such a parameterized instance is allowed to contain one linear parameter. A solution to a parameterized algorithm now consists of an assignment of values to the variables of the instance and, additionally, a value for the given parameter such that the problem is optimal or feasible, respectively. Megiddo shows that the value for the parameter can be found in strongly polynomial time, if there exists an "easy" algorithm for the non-parameterized problem that runs in strongly polynomial time. In the following, we show how this parametric search works and enumerate the requirements for such an algorithm to be called "easy".

We are given a parameterized instance of a certain problem where a linear parameter λ is part of the instance. The parameter has to be known to lie in an interval $I \subseteq (-\infty, \infty)$. Further, we assume that we are given a strongly polynomial algorithm $\mathcal{A}$ for the considered problem. We require that the steps of $\mathcal{A}$ are only of one of the following three types: additions, scalar multiplications, and comparisons.

Using the parameterized instance, the algorithm $\mathcal{A}$ has to be adapted to be a parameterized version. In the adapted version not only values a_i and a_j have to be added, multiplied by a scalar c, or compared, but linear functions $a_i + \lambda b_i$ and $a_j + \lambda b_j$.

Addition of linear functions $a_i + \lambda b_i + a_j + \lambda b_j$ reduces to additions of known values $(a_i + a_j) + \lambda(b_i + b_j)$, as well as scalar multiplication $c(a_i + \lambda b_i) = (ca_i) + \lambda(cb_i)$ reduces to scalar multiplication of known values and conserves the linearity. Therefore, the knowledge of the value of λ is not necessary. The comparison of linear parameterized functions is more difficult without

knowing λ. By simple computation of the intersection point of two linear functions $a_i + \lambda b_i$ and $a_j + \lambda b_j$, the critical value λ^* for which $a_i + \lambda^* b_i = a_j + \lambda^* b_j$ can be found. If there is no critical value or the value of λ^* does not lie in the feasible interval I, the result of the comparison is independent of λ. The result of the comparison thus can be determined immediately. If the value λ^* lies in I, we have to check whether it is too high or too low for a feasible value of λ. This check can be done using the strongly polynomial algorithm $\mathcal{A}$ on the non-parameterized problem where the parameter is fixed to λ^*. Then the outcome is analyzed, for example by comparing the objective values for the actual choice of λ^* and the one from the last iteration or checking the existence of a feasible solution, respectively. If λ^* was too low, we substitute the lower bound of interval I by λ^*, if it was too high, we substitute the upper bound.

Megiddo [62] shows that the number of tests for λ^* can be bounded by the number of comparisons done by the algorithm $\mathcal{A}$ for a non-parameterized problem. Since $\mathcal{A}$ is used as a subroutine whenever a comparison is done, the total running time of $\mathcal{A}$ for a parameterized problem lies in the running time of algorithm $\mathcal{A}$ for the non-parameterized problem times the number of the comparisons of $\mathcal{A}$. Thus, if the running time of algorithm $\mathcal{A}$ for the non-parameterized problem is strongly polynomial, the running time of the algorithm for the parameterized version will be strongly polynomial, too.

Throughout this thesis, all algorithms used together with parametric search fulfill the requirements that only additions, scalar multiplications, and comparisons are necessary. Further, only linear parameters are used.

Chapter 2

A Survey on Evacuation Problems

2.1 Introduction

The task of evacuation problems is to send the maximum number of people to evacuate from a dangerous site to safe sites. Obviously, the parameter time plays an important role in evacuation problems. Therefore, only flow over time models are considered to model the evacuation setting in a reasonably realistic way. *Quickest transshipment problems* are analyzed in order to get a reasonable lower bound for the minimum time horizon of a real evacuation situation and is therefore of high relevance. Given an amount of supplied and demanded flow in source and sink nodes the quickest transshipment problem asks for a flow that sends this amount of flow over time from sources to the sinks in the minimum possible time horizon. In the context of emergency evacuation from buildings, Berlin [8] and Chalmet et al. [12] study the quickest transshipment problem in networks with multiple sources and a single sink. Jarvis and Ratliff [44][1] showed that three different objectives of this optimization problem can be achieved simultaneously: (1) Minimizing the total time needed to send the supplies of all sources to the sink, (2) maximizing the amount of flow leaving the network at all times $\theta \geq 0$, and (3) minimizing the average time for all flow needed to reach the sink. Each transshipment optimizing objective (1) is a quickest transshipment. Objective (2) says that it is additionally possible to simultaneously maximize the amount of flow sent into the sink up to all times $\theta \geq 0$ in a transshipment problem. A transshipment optimizing objective (2) is called *earliest arrival transshipment*. Each earliest arrival transshipment obviously additionally optimizes objective (1) and is therefore a quickest transshipment. The reverse does not hold, i.e., not every quickest transshipment optimizes (2), too. Notice that earliest arrival transshipments do not necessarily exist in networks with multiple sinks, but quickest transshipments do. Besides, each transshipment maximizing objective (2) naturally minimizes objective (3). We will not consider objective (3) in the following, since we concentrate on objective (2), maximizing the

[1] Strictly speaking, Jarvis and Ratliff [44] only consider the single-source case but their observation also applies to the more general case with multiple sources.

amount of flow leaving the network at all times $\theta \geq 0$, in this thesis.

Not only transshipment problems but also maximum s-t-flows over time problems can be considered in order to model different evacuation settings. The task is to send as many people out of an endangered site. Again, it is preferable, if the computed maximum s-t-flow has the property, that it also maximizes the amount of flow sent into the sink up to each time $\theta \geq 0$. Therefore, *earliest arrival s-t-flows* are considered. An evacuation fulfilling the earliest arrival property is known to be optimal, even if the catastrophe occurs before all people have left the site to evacuate, since the number of evacuated people by then is as large as possible.

In the following, we will give a survey on the above mentioned network flow problems and state known results as well as algorithms. In Section 2.4 we overview further methods of solving evacuation problems.

2.2 Quickest Transshipment Problems

In the following, we consider *quickest transshipment problems* which are studied very well, i.e., problems where objective (1) is optimized. An instance of the quickest transshipment problem consists of a network $\mathcal{N}$ with capacities and transit times on the arcs, multiple source nodes S^+ and multiple sink nodes S^- with a supply-demand function $d : S^+ \cup S^- \to \mathbb{R}$. The task is to determine the minimum time horizon θ^* for which there exists a feasible flow satisfying all supplies and demands as well as to find such a feasible flow. In the following, we will give a short overview over results for the quickest transshipment problem for several network flow models.

2.2.1 Quickest Transshipments with Constant Transit Times

For an instance of the quickest transshipment problem with constant transit times, we further require that capacities and transit times on arcs are constant and integral. If capacities and transit times are not integral, scaling helps to fulfill this property.

There exists an easy to see algorithm in the time-expanded network to determine a quickest transshipment. For such an algorithm, we first guess an (integral) time horizon θ and construct the time-expansion of network $\mathcal{N}$. We add a supersource s which is connected to each copy $s'(0)$ of sources $s' \in S^+$ at time layer zero by a zero transit time arc having capacity $d(s')$. Further, we add a supersink t to the network with uncapacitated, zero transit time arcs $(t'(\theta-1), t)$ for all copies of $t' \in S^-$ at time layer $\theta-1$. Using a maximum static s-t-flow computation in this network, we compare the resulting flow

value with the total supply of the problem. If the flow value is smaller than the total supply, we need to increase the guessed time horizon. Otherwise, we have found an upper bound on the optimal time horizon. A natural upper bound for the time horizon is obviously $|V| \cdot \max\{\tau(a)|a \in A\} + \sum_{s \in S^+} d(s)$ since we assumed integral capacities. The minimum (integral) value for θ such that all supplies and demands are fulfilled can then be found using binary search. This algorithm has the disadvantage that it needs the time-expansion which makes it pseudo-polynomial. If we do not restrict ourselves to integral time horizons and transit times there can occur more problems. First, the construction of time-expanded networks needs to be redefined, if the time horizon or transit times of arcs are not integral. Moreover, the binary search could need exponentially many steps until the optimal time horizon is determined.

In the following we present the strongly polynomial time algorithm by Hoppe and Tardos [40] to solve the quickest transshipment problem. In particular, we show how to determine the optimal time horizon avoiding binary search.

A Strongly Polynomial Time Algorithm

In order to compute a quickest transshipment in strongly polynomial time, we especially need to look at the closely related *transshipment over time problem*. An instance of the transshipment over time problem consists of an instance of the quickest transshipment problem where we are further given a time horizon $\theta > 0$. The task is to find a feasible transshipment over time within time horizon θ or to determine that there does not exist a feasible transshipment over time. The following definition determines the notion of feasibility.

Definition 2.1. An instance of the transshipment over time problem is said to be *feasible*, if there exists a feasible flow over time that fulfills supplies and demands within the given time horizon θ.

In order to compute a feasible transshipment with minimum time horizon, the following has to be done. In a first step, the minimum time horizon θ^* need to be computed. For a fixed value θ, the quickest transshipment problem reduces to the transshipment over time problem. In this instance, we have only to check feasibility. Doing this for several values of θ, we are able to find the minimum value of θ such that the transshipment over time problem is feasible. If we found the minimum time horizon θ^*, we are given an instance of the transshipment over time problem. By modifying the network, we reduce

this instance in network $\mathcal{N}$ to an instance of the *lexicographically maximum flow over time problem*.

In the following, we first show how to determine the minimum possible time horizon θ^* for the quickest transshipment problem in strongly polynomial time. After that, the assumptions on the network and an algorithm for solving the lexicographically maximum flow over time problem will be stated before we describe the network modification to equivalently transfer an instance of the transshipment over time problem to an instance of the lexicographically maximum flow over time problem.

Minimizing the time horizon θ^*. In order to determine the minimum time horizon for which an instance of the quickest transshipment problem stays feasible we need the following powerful criterion by Klinz [50]. It determines whether the instance is feasible for a fixed value θ and is an important building block of this thesis. We use here the continuous version which was extended from the discrete version of Klinz by Fleischer and Tardos [23].

Lemma 2.2 (Klinz [50] and Fleischer and Tardos [23]). For $\theta \geq 0$ and $X \subseteq S^+ \cup S^-$ let $d(X) := \sum_{v \in X} d(v)$ and let $o^\theta(X)$ be the maximal amount of flow that can be sent from the sources in $S^+ \cap X$ to the sinks in $S^- \setminus X$ within time θ (ignoring supplies and demands). There exists a continuous flow over time with time horizon θ that satisfies all supplies and demands if and only if

$$o^\theta(X) \;\geq\; d(X) \qquad \text{for all } X \subseteq S^+ \cup S^-. \tag{2.1}$$

Since Lemma 2.2 is of importance, we will prove it in the following. First, we need to state a well known result from Gale [27].

Lemma 2.3 (Gale [27]). For each static network $\mathcal{N} = (V, A)$, $S^+ \cup S^- \subseteq V$, let u be the capacity function and d the supply-demand function. The cut condition, i.e.,

$$u(X) \geq d(X) \qquad \text{for each } X \subseteq S^+ \cup S^-, \tag{2.2}$$

is fulfilled if and only if there exists a feasible flow obeying supplies and demands.

Proof of Lemma 2.2. If there exists a flow satisfying all supplies and demands, then obviously the feasibility criterion (2.1) holds. Thus, it only remains to show sufficiency. We assume that the feasibility criterion holds. The sufficiency of the feasibility criterion (2.1) follows directly from Lemma 2.3: Let $\mathcal{N}^\theta = (V^\theta, A^\theta)$ be the time-expanded network of $\mathcal{N}$ with time horizon θ.

In this network we define the set of sources to be all copies of $s \in S^+$ at time layer zero and the set of sinks to be all copies of $t \in S^-$ at time layer $\theta - 1$. Since a cut in $\mathcal{N}^\theta$ corresponds to a feasible cut over time in $\mathcal{N}$, the feasibility criterion (2.1) holds in the network $\mathcal{N}$ if and only if the cut condition (2.2) holds in the static network $\mathcal{N}^\theta$. Applying Lemma 2.3 and by the correspondence of a static flow in the time-expanded network and a flow over time in the original network, we have proven the statement of Lemma 2.2 for all networks for which there exists a time-expansion.

A time-expansion obviously exists for integral values of θ. We will now consider non-integral values and show how to find adequate time-expansions. Using these time-expansions and applying Lemma 2.3 we have proven the statement for all possible values θ.

Observe that the function $o^\theta(X)$ is a continuous function in θ for each set $X \subseteq S^+ \cup S^-$. Therefore, it exists a time θ^* at which $o^{\theta^*}(X) = d(X)$ holds. Consider the extended network $\mathcal{N}'$ defined as follows. Starting from $\mathcal{N}$, we insert a supersource s that is connected to all sources in $S^+ \cap X$ by an uncapacitated arc with transit time zero. Furthermore, we insert a supersink t that can be reached from all sinks in $S^- \setminus X$ by an uncapacitated, zero transit time arc. By construction of $\mathcal{N}'$, the value $o^{\theta^*}(X)$ is equal to the value of a maximum s-t-flow over time in $\mathcal{N}'$ with time horizon θ^* which can be computed via one static min-cost s-t-flow computation. Let x_X be such a static flow. By formula (1.2) (page 15) for the value of a maximum s-t-flow over time we know that

$$d(X) = \theta^* \cdot \text{value}(x_X) - \sum_{a \in A(\mathcal{N})} \tau(a) x_X(a)$$

and therefore

$$\theta^* = \frac{d(X) + \sum_{a \in A(\mathcal{N})} \tau(a) x_X(a)}{\text{value}(x_X)} \ .$$

Because of the integrality assumption for the capacities, the flow x_X is integral. Then it follows that time horizon θ^* for which $o^{\theta^*}(X) = d(X)$ is a rational where the denominator is bounded by the value of a maximum static s-t-flow x^* in network $\mathcal{N}$.

Let us now consider arbitrary values of θ. If θ equals θ^* for which $o^{\theta^*}(X) = d(X)$ for some $X \subseteq V$, we can build a time-expanded network where the transit times, especially of the holdover arcs, and the time horizon become integral by scaling. That is, all transit times and θ have to be multiplied by a constant $z \leq \text{value}(x^*)$ and all capacities need to get divided by z.

For all other values of θ, observe that the feasibility of the problem does not change when we decrease θ to the nearest number θ' with denominator bounded by value(x^*). Since function o is continuous in θ we will maintain the property that $o^{\theta'}(X) \geq d(X)$ if and only if $o^{\theta}(X) \geq d(X)$.

Thus, it follows that for all times $\theta \geq 0$, it is possible to construct a time-expanded network with time horizon at most θ in which the cut condition (2.2) holds. □

As described in the proof of Lemma 2.2, the value $o^{\theta}(X)$, for $\theta \geq 0$ and $X \subseteq S^+ \cup S^-$, can be obtained by a static min-cost s-t-flow computation. It follows from the work of Ford and Fulkerson [24] for flows over time (compare also Section 1.2.2) that

$$o^{\theta}(X) \;=\; -\min\left\{\text{cost}^{\theta}(x) \mid x \text{ static min-cost } s\text{-}t\text{-flow in } \mathcal{N}'\right\} \;. \tag{2.3}$$

Here, $\text{cost}^{\theta}(x)$ denotes the cost of the static min-cost s-t-flow x where transit times on arcs are interpreted as cost coefficients and the cost coefficient of dummy arc (t, s) is set to $-\theta$. As a consequence of (2.3), the function $\theta \mapsto o^{\theta}(X)$ is the negative of the cost function of a parametric static min-cost s-t-flow problem. As such, it is piecewise linear and convex.

Hoppe and Tardos [40] observe that the function $o^{\theta} : S^+ \cup S^- \to \mathbb{R}$ is submodular, that is,

$$o^{\theta}(X) + o^{\theta}(Y) \;\geq\; o^{\theta}(X \cup Y) + o^{\theta}(X \cap Y) \qquad \text{for all } X, Y \subseteq S^+ \cup S^-.$$

By the modularity of the supply-demand function d, we get that the function $b : 2^V \to \mathbb{R}$ defined as

$$b(X) := o^{\theta}(X) - d(X)$$

is submodular. Thus, the feasibility criterion (2.1) reduces to $b(X) \geq 0$ for all $X \subseteq S^+ \cup S^-$. Using the complexity results of minimization of submodular functions (see Section 1.3), it is possible to check for a given value θ in strongly polynomial time whether the problem is feasible. Since we are interested in the minimum time horizon for which the problem stays feasible, we look for the minimum value of θ. For finding this value, we can use the strongly polynomial parametric search as described in Section 1.4 where θ is the linear parameter. This yields the following Theorem.

Theorem 2.4 (Hoppe and Tardos [40]). The minimum time horizon θ^* such that the feasibility criterion (2.1) is fulfilled, i.e., there exists a flow satisfying all supplies and demands, can be found in strongly polynomial time.

Lexicographically Maximum Flow over Time Problem. Before we consider the flow over time version, we consider the *lexicographically maximum static flow problem*. An instance of the lexicographically maximum static flow problem consists of a network $\mathcal{N} = (V, A)$ with capacities on the arcs, a set of sources $S^+ \subsetneq V$ and a set of sinks $S^- \subseteq V \setminus S^+$. On the set of sources and sinks we require an ordering, say $s_0, s_1, \dots, s_\ell$, for $S^+ \cup S^- = \{s_0, s_1, \dots, s_\ell\}$. A lexicographically maximum static flow is a feasible static flow that maximizes the amount of flow leaving each source/sink in the given order. Flow leaving a sink s_i is meant to leave the network by entering the sink. Minieka [65] and Megiddo [61] observe that a solution to the lexicographically maximum static flow problem can be described as a flow that simultaneously maximizes the amount of flow leaving subsets $S_i = (s_0, \dots, s_i)$ for $i = 0, \dots, \ell$. For the lexicographically maximum static flow problem the following existence result holds.

Lemma 2.5 (Minieka [65]). In networks with a given set of ordered sources and sinks $\{s_0, \dots, s_l\}$ there exists a lexicographically maximum static flow, i. e., a flow that simultaneously maximizes the amount of flow leaving subsets $S_i = (s_0, \dots, s_i)$.

In an instance of the *lexicographically maximum flow over time problem*, we are further given transit times on the arcs and a time horizon θ^*. By the correspondence of a static flow in the time-expanded network and a flow over time in the original network, the above result also holds for discrete flows over time. Fleischer and Tardos [23] extended this correspondence to continuous flows over time. The following paragraph shortly describes an algorithm by Hoppe and Tardos [40] on how to find such a flow over time that maximizes the flow sent out of sources and reaching the sinks, respectively, up to time θ^* in the given order.

Starting from the original network $\mathcal{N}$, we construct network $\mathcal{N}_{\ell+1}$ as follows. We add a node ψ that is connected by uncapacitated, zero transit time arcs (ψ, s_i) to all sources $s_i \in S^+$. Let $f^{\ell+1}$ denote the zero flow. We denote the residual network of some network $\mathcal{N}_i$ and a suitable flow f^i by $\mathcal{N}_i^{f^i}$. By the above notation, obviously $\mathcal{N}_{\ell+1}^{f^{\ell+1}} = \mathcal{N}_{\ell+1}$ holds. During the next $\ell + 1$ iterations $i = \ell, \dots, 0$, the algorithm considers the sources and sinks $s_\ell, s_{\ell-1}, \dots, s_0$ in descending order. If s_i is a source, arc (ψ, s_i) becomes deleted from network $\mathcal{N}_{i+1}^{f^{i+1}}$ to obtain network $\mathcal{N}_i^{f^{i+1}}$. Then we compute a maximum static ψ-s_i-flow g^i having minimum cost in $\mathcal{N}_i^{f^{i+1}}$ where transit times are interpreted as costs of the arcs. If s_i is a sink, we first add an uncapacitated arc (s_i, ψ) with transit time $-\theta^*$ to network $\mathcal{N}_{i+1}^{f^{i+1}}$. Then

we compute a static min-cost circulation g_i in the resulting residual network $\mathcal{N}_i^{f^{i+1}}$, again interpreting transit times as costs. At the end of each iteration, we construct flow f^i by adding the new flow g^i to f^{i+1}.

To construct the lexicographically maximum flow over time, we temporally repeat the flows over the paths computed in the several iterations (confer [40]). The resulting flow is a feasible lexicographically maximum flow over time.

Theorem 2.6 (Hoppe and Tardos [40]). A lexicographically maximum flow over time can be found in strongly polynomial time using the above described algorithm.

Transshipment Over Time Problem. The *transshipment over time problem* can be solved by modifying the network and defining an order on the sources and sinks such that we can easily apply the algorithm to compute a lexicographically maximum flow over time. We will shortly explain the network modification and how to define the order of sources and sinks in the following. For more details we refer to [40].

In a network, we are given supplies and demands. If the transshipment over time problem is feasible, then the maximum flow value that can be sent out of a source over time is larger than or equal to the supply of this source. Analogously, the maximum amount that can enter a sink within the time interval $[0, \theta^*)$ is larger than or equal to the negative demand of this sink. The goal is to decrease, if necessary, the maximum flow that can be sent out of a source or reached by a sink over time to the value of the supply and the negative of the demand, respectively. This can be done by reducing the capacity of the outgoing and incoming arcs of sources and sinks in a clever way. Therefore, we define tightness of sets of sources and sinks.

Definition 2.7. We say a subset $X \subseteq S^+ \cup S^-$ of sources and sinks is *tight* if $o^{\theta^*}(X) = d(X)$.

In a first step, we insert for all sources $s \in S^+$ a node s' which is connected to s via an uncapacitated, zero transit time arc. The supply of s is moved to s' and these new nodes now build the set of sources S'^+ in this new network. Here s acts as an intermediate node. We proceed with the sinks analogously. The new set is then called S'^-. Notice that the problem itself stays the same since all transformations are redundant. Let $\mathcal{C}$ denote a set of subsets of sources and sinks. We further require $\mathcal{C}$ to be a *chain*, i.e., the elements in $\mathcal{C}$ can be ordered in such a way that they are *nested*. This means that for each element $S_i \in \mathcal{C}$ it holds that $S_i \subsetneq S_j$ for $i < j$. We initialize the chain $\mathcal{C}$ by $\mathcal{C} := \{\emptyset, S'^+ \cup S'^-\}$. Both sets in $\mathcal{C}$ are trivially tight and $\mathcal{C}$ is

nested. We will successively insert tight subsets of sources and sinks into $\mathcal{C}$. The goal is to find a chain of nested, tight subsets of sources and sinks where the size of subsequent elements in the chain differs by one. This will define an ordering of the sources and sinks such that each element S_i of the chain $\mathcal{C}$ contains sources and sinks $\{s_0, s_1, \ldots, s_i\}$.

We call two subsets Q and Q' *adjacent in* $\mathcal{C}$ if there is no set R in $\mathcal{C}$ such that $Q \subsetneq R \subsetneq Q'$. As long as there are two adjacent subsets Q and Q' in the chain which differ by more than one element, we choose one of these elements, say s'. Assume s' is a source. If $Q \cup \{s'\}$ is tight, we are done. Otherwise, we add two sources s_1 and s_2 to S'^+. We insert arc (s_1, s), which has zero transit time and a non-negative capacity to be determined, and arc (s_2, s), which has a non-negative transit time to be determined and infinite capacity. Here s is the original source $s \in S^+$ to which s' is connected. Then, we choose first the capacity of (s_1, s) and the supply of s_1 and secondly the transit time of (s_2, s) and the supply of s_2 such that $Q \cup \{s_1\}$ and $Q \cup \{s_1\} \cup \{s_2\}$ are tight, the total sum of new supplies of s', s_1, s_2 equals the former supply of s', and the problem is still feasible. Nodes s_1 and s_2 have to be added to all subsets in $\mathcal{C}$ where s' is in. The procedure works analogously, if s' is a sink. We denote the resulting transformed network by $\tilde{\mathcal{N}}$.

The transformation from network $\mathcal{N}$ to network $\tilde{\mathcal{N}}$ has the property that the amount of flow that can be sent out of sources or reached by sinks over time is reduced. Thereby, it is guaranteed that the original supplies and demands still can be fulfilled. Then the following can be concluded.

Theorem 2.8 (Hoppe and Tardos [40]). The network modification as described above can be done in strongly polynomial time and a feasible transshipment in network $\tilde{\mathcal{N}}$ does not need more time to fulfill all supplies and demands than a feasible transshipment in $\mathcal{N}$.

To apply the lexicographically maximum flow over time algorithm we order the sources and sinks of the new network $\tilde{\mathcal{N}}$ according to their occurrence in the sets in chain $\mathcal{C}$. This yields sets $S_i := \{s_0, \ldots, s_i\}$ for all nested sets $S_i \in \mathcal{C}$. In particular, in $\tilde{\mathcal{N}}$ it holds for each source s_i that the supply equals the value of a maximum flow over time out of this source subject to the fact that all supplies and demands of sources and sinks s_j, $j < i$, are fulfilled. Equivalently, for each sink s_i it holds that the negative of the demand equals the maximum amount of flow that can be sent into the sink up to time θ^* subject to the fact that all supplies and demands of sources and sinks s_j, $j < i$, are fulfilled. This network is a feasible instance of the lexicographically maximum flow over time problem which now can be solved

in this network. The resulting flow is a feasible transshipment in $\tilde{\mathcal{N}}$ and induces a feasible transshipment in the original network $\mathcal{N}$.

2.2.2 Quickest Transshipments with Inflow-Dependent Transit Times

In the following, we will consider the quickest transshipment problem in a single-source-single-sink network having inflow-dependent transit times on arcs. For the case of constant transit times, we can either use the algorithm from the previous section or apply a strongly polynomial time algorithm by Burkard, Dlaska, and Klinz (see [9] for details) for this special case of quickest transshipments.

For the case of inflow-dependent transit times, some more elaborate techniques need to be used. Building up on the approach of Carey and Subrahmanian [11], Köhler, Langkau, and Skutella [54] introduce a model that is both time-expanded and expands the transit time function for the case of inflow-dependent transit times that depend on the amount of flow entering an arc. While this expanded model only allows pseudo-polynomial algorithms, they also give a reduced version of this network – the *bowgraph*. Using this, they design an efficient 2-approximation algorithm for the quickest transshipment problem with inflow-dependent transit times. In the following, we present this model and the algorithm.

The Quickest Transshipment Algorithm in the Bowgraph. The main idea of the approximation algorithm from [54] is the usage of a certain relaxed network that is constructed from the original network: Instead of working with arcs with inflow-dependent transit times, the arcs of the original network are expanded in such a way that all the transit times are constant. More precisely, a set of parallel arcs with different transit times and capacities is created that represents the different possible states of the particular arc. This expanded network is called the *bowgraph* (see Figure 2.1 for a simple example).

The approximation algorithm first computes a quickest transshipment in this bowgraph, providing a lower bound on the time horizon of the quickest transshipment for the inflow-dependent problem. This quickest transshipment can be determined in polynomial time since the bowgraph has only arcs with constant transit times. Although the flow computed this way is not feasible in the original network (there can be flow on different copies of the same arc), it can be shown that it can be made feasible by a simple push-up operation that puts all the flow of the different copies of an arc onto only a single copy of it (see Figure 2.2). While this push-up operation in-

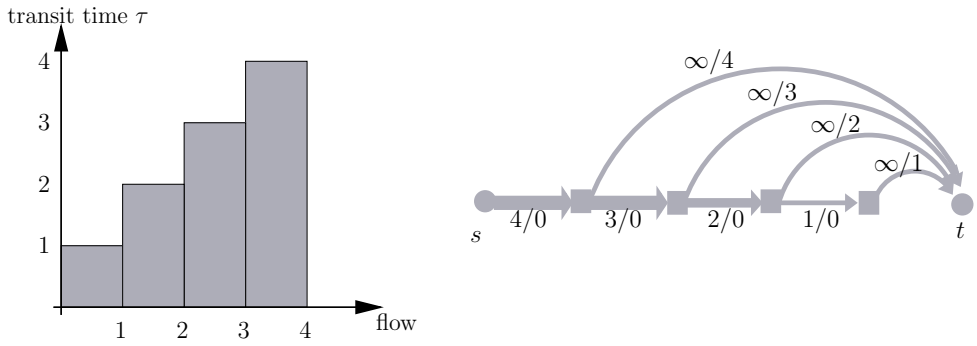


Figure 2.1: Consider a network consisting of only one arc (s, t). The transit time function is given as a left-continuous step function as depicted on the left hand side. On the right hand side, the bowgraph for the single arc network is shown. The different bows represent different possible transit times for this arc. A suitable set of capacities on the horizontal arcs guarantees that for each bow no more flow can enter than the transit time function of the original network would allow. The bows itself have infinite capacity. Notation on the arcs of the network: *capacity/transit time*.

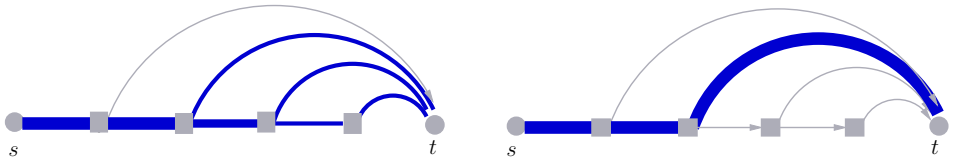


Figure 2.2: The left versus the right version of the picture shows the bowgraph before and after the push-up operation, respectively. The width of the flow represents the amount of flow sent over the corresponding arc.

creases the time horizon of the original computed quickest transshipment, it can be shown that this will be not more than twice as large as before and thus gives a 2-approximation of the quickest transshipment problem for the inflow-dependent model. For more details, we refer to [54].

Hall, Langkau, and Skutella [33] extend the above described function to an approximation scheme for this problem even for the multi-commodity case.

Remarks. Köhler and Skutella [55] consider load-dependent transit times, give a model for this setting and give a 2-approximation algorithm for the quickest transshipment problem in the single-source-single-sink setting. They also show that there cannot be a polynomial-time approximation scheme unless P=NP.

2.3 EARLIEST ARRIVAL FLOW PROBLEMS

In the following we will consider two flow problems taking into account the earliest arrival property. Flows having this property are not only optimal at a given time horizon but have sent the maximum amount possible into the sink by each time $\theta \geq 0$. Therefore, solutions to earliest arrival flow problems are much more applicable to evacuation settings, where the time of the catastrophe is unknown, as solutions to the quickest transshipment problems are.

In the following, we will give a short overview over results on the earliest arrival s-t-flow problem which is a special case of the maximum s-t-flow over time problem. After that, we consider the earliest arrival transshipment problem — a special case of the quickest transshipment problem.

2.3.1 Earliest Arrival s-t-Flows

Assume we are given a single-source-single-sink network with constant transit times on the arcs. In Section 1.2.2, we shortly described how to find a maximum s-t-flow over time from source s to sink t. There we are given a time horizon T and the task is to send as much flow from s to t within that time horizon. A feasible maximum s-t-flow over time could send the total flow value shortly before the given time horizon T into the sink. In an evacuation setting, we are also interested in routing as many people as possible into the safe site up to the time where the catastrophe occurs. Since the exact times of catastrophes are not predictable, we are interested in solutions which avoid sending flow into the sink only shortly before time T. Earliest arrival s-t-flows prevent such a flow behavior, if possible.

Shortly after Ford and Fulkerson[24] introduced an algorithm for the maximum s-t-flow over time problem, the more elaborate *earliest arrival s-t-flow problem* was studied by Gale [28]. He gave a proof of the existence of earliest arrival s-t-flows in the discrete flow over time model. The existence in the continuous flow over time model has been first observed by Philpott [67]. Recall from Section 2.1 that the goal is to find an s-t-flow over time that simultaneously maximizes the amount of flow reaching the sink t up to any time $\theta \geq 0$. By the definition, it can easily be seen that each earliest arrival s-t-flow up to a time $\theta \geq 0$ is also a maximum s-t-flow over time with time horizon θ. The converse does not hold in general.

Though Gale [28] showed the existence of earliest arrival s-t-flows, it is not obvious how to find them algorithmically. Minieka [65] and Wilkinson [77] were the first who tackled this problem and designed algorithms for finding

earliest arrival s-t-flows. Both algorithms rely on the *successive shortest path algorithm* [45, 41, 10]. There, we successively search for shortest paths from s to t in the residual network as long as there exist augmenting s-t-paths. An amount of the minimal residual capacity on such a path is augmented. The found path P and the minimal residual capacity $x(P)$ are stored.

In order to obtain the earliest arrival s-t-flow, which is a flow over time, flow of value $x(P)$ is sent temporally repeated over paths $P \in \mathcal{P}$. Notice that the set $\mathcal{P}$ together with the flow values $x(P)$ for all $P \in \mathcal{P}$ forms a kind of path decomposition. To maintain the earliest arrival property of that s-t-flow, it is necessary to send flow also over backward arcs. By sending flow over an backward arc, we send this flow back to an earlier point in time. Path decompositions also using backward arcs are called *generalized path decompositions*. Unfortunately, the number of paths in a generalized path decomposition cannot longer be bounded by the number of arcs of the network as it can be done for (non-generalized) path decompositions as described in Section 1.2.1. For more details on generalized path decomposition we refer to [38].

As shown by Zadeh [78], there are families of networks that require the successive shortest path algorithm to have a pseudo-polynomial number of iterations, that means the algorithm does not run in polynomial time. In fact, the networks given by Zadeh can be interpreted as instances of the earliest arrival s-t-flow problem, requiring the successive shortest paths algorithm to need $\Omega(T)$ iterations. Here, T denotes the given time horizon. It is still an open question whether the earliest arrival s-t-flow problem can be solved in polynomial time.

Hoppe and Tardos [39] were the first to present an approximation algorithm for finding earliest arrival s-t-flows where their objective was to find for a given $\varepsilon > 0$ a flow f such that for each $\theta \geq 0$ the amount of flow reaching t by θ is at most an ε-fraction less than the optimum. They achieved this by using a capacity scaling approach and the generalized path decomposition achieved by using the successive shortest path algorithm. Fleischer and Tardos [23] extend the correctness of the above algorithm to the continuous flow over time model. Fleischer and Skutella [22] use the algorithm of Hoppe and Tardos [39] to obtain an approximate min-cost (maximum) s-t-flow over time where the time horizon is enlarged by a factor $(1 + \varepsilon)$.

A closely related problem is the earliest arrival s-t-flow problem for time-dependent transit times (i.e., every arc can have for different times different (constant) transit times) or time-dependent capacities. In this setting, the transit time of an arc is not constant but varies over time. The existence result by Gale also holds for this special case of transit times in discrete flows over time. Other parameters that could change over time are capacities

of arcs and, if given, for nodes (recall that node capacities allow that flow that reached a node can be stored in this node up to a given capacity and is sent into an outgoing arc at a later time). Tjandra [75] shows how to compute discrete earliest arrival s-t-flows in networks with time-dependent transit times and capacities (arc capacities as well as node capacities) in time $\mathcal{O}(|V|(|V|+|A|)T^2)$. Ogier [66] studied the case of zero transit times where arc capacities and node capacities are piecewise-constant functions of time. For this special case he gives a polynomial algorithm. Fleischer [19] improved the running time of this algorithm.

2.3.2 Earliest Arrival Transshipments

In the following, we consider the earliest arrival transshipment problem which is a special case of the quickest transshipment problem. That is, every earliest arrival transshipment is also a quickest transshipment, but the reverse does not hold in general. Earliest arrival transshipments are motivated by applications strongly related to evacuation of sites where the amount of people at specified places is known in advance. This can be the evacuation of office buildings, apartment buildings, or airplanes. In those applications, we may assume to know exactly how many people are in an office because, e.g., the number of working stations is known. In an airplane, exactly one person can sit on a seat. The seats are modeled as the sources. We assume that there will be no restriction on the amount of people safe sites can house as it is the case in places outside a building or outside an airplane. All these settings can be modeled as a network with a given set of sources and a set of sinks and a supply-demand function $d : S^+ \cup S^- \to \mathbb{R}$. We require that $\sum_{s\in S^+} d(s) + \sum_{t\in S^-} d(t) = 0$. The task is to evacuate as many people as early as possible from the site to be evacuated.

Fleischer [18] showed that transshipments having the earliest arrival property do not necessarily exist in multiple-sources-multiple-sinks networks. We give a simple counterexample with one source and two sinks in Figure 2.3. Richardson and Tardos [70] observed, that for the case of several sources with given supplies and a single sink, earliest arrival transshipments do always exist. This follows, for example, from the existence of lexicographically maximum static flows in time-expanded networks; see, e.g., Minieka [65]. Therefore, we will only consider multiple-sources-single-sink networks in the following.

An easy to see algorithm to compute an earliest arrival transshipment is the use of the time-expanded network as for the quickest transshipment problem (confer Section 2.2.1). In order to build up the time-expanded net-

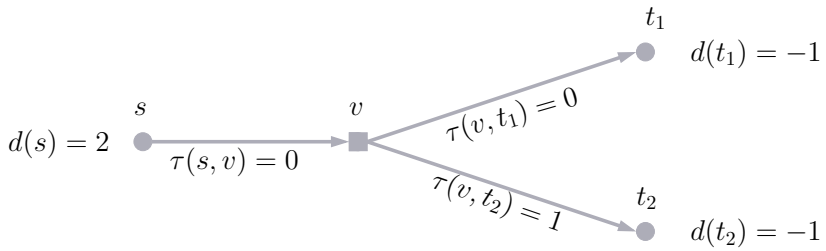


Figure 2.3: A network with one source and two sinks with unit demands for which an earliest arrival flow does not exist. All arcs have unit capacity and the transit times are given in the drawing. Notice that one unit of flow can reach sink t_1 by time 1; in this case, the second unit of flow reaches sink t_2 only by time 3. Alternatively we can send one unit of flow into sinks t_1 and t_2 simultaneously by time 2. It is impossible to fulfill the requirement, that both flow units have reached their sink by time 2 and one of them has already arrived at time 1.

work, we first need to guess a (integral) time horizon T. Then there are several static flow algorithms that yield an earliest arrival transshipment for the original network. One algorithm is indicated by the above stated existence result of Richardson and Tardos [70]. To apply a lexicographically maximum static flow algorithm in the time-expanded network we extend it by the supersource s connected to all sources $s' \in S^+$ at time layer zero. The capacity of each arc (s, s') is set to the supply of the connected original source $s' \in S^+$. The sinks $t(0), t(1), \ldots, t(T-1)$ in the time-expanded network are sorted in this order corresponding to the time layer they are in. The application of the lexicographically maximum static flow algorithm on the ordered set $\{s, t(0), \ldots, t(T-1)\}$ yields a static flow which can be interpreted as a feasible earliest arrival transshipment in the original network.

A second possibility of computing an earliest arrival transshipment using the time-expanded network is the use of a static min-cost s-t-flow algorithm. To apply this algorithm, we add a supersource s to the time-expanded network as before and assign zero costs to those arcs. Further, a supersink t is inserted and connected from all sinks $t'(0), \ldots, t'(T-1)$ of the time-expanded network with zero costs. The cost of a holdover arc is set to one. For all remaining arcs we set the cost to the transit time of the corresponding arc in the original network. A feasible static min-cost (maximum) s-t-flow in this network can be interpreted as a feasible earliest arrival transshipment in the original network.

Since the use of time-expanded networks increases the running time to be at least pseudo-polynomial, we are interested in algorithms that do not need a time-expanded network.

Hajek and Ogier [32] give the first polynomial time algorithm for the earliest arrival transshipment problem in networks with zero transit times

on all arcs. Fleischer [18] gives an algorithm with improved running time. Fleischer and Skutella [21] use geometrically condensed time-expanded networks to approximate the earliest arrival transshipment problem for the case of arbitrary transit times. They give an FPTAS that approximates the time delay as follows: For every time $\theta \geq 0$ the amount of flow that should have reached the sink in an earliest arrival transshipment by time θ, reaches the sink at latest at time $(1+\varepsilon)\theta$.

2.4 Further Methods of Modeling and Optimizing Evacuation

There are many more results for evacuation models and its optimization. In the following, we mention further literature modeling certain evacuation problems by network flows. Thereafter, we shortly overview the literature on simulation approach concerning evacuation problems.

A huge variety of extensions to existing network flow models exist in order to apply them to the evacuation problem. Most of them only consider the quickest transshipment problem and side constraints. Hamacher and Tufecki [37] study the quickest transshipment problem and propose solutions which further prevents unnecessary movement within a building. Moreover, they consider the problem where several areas of certain priorities for evacuation exist. Both these models are solved via time-expansion of the network and are therefore undesirable. Another problem solved in the time-expanded network is a network flow problem with flow-dependent capacities but constant transit times by Choi et al. [13] and Choi, Hamacher, and Tufecki [14]. By introducing turnstile costs which penalize flow leaving the network late, they obtain earliest arrival flows. In his dissertation, Tjandra [75] considers several variants of the discrete quickest transshipment problem. For the case of time-dependent supplies and capacities (i.e., every node/arc can have for different times different (constant) supplies/capacities), he states a pseudo-polynomial time algorithm. Moreover, he solves the discrete earliest arrival s-t-flow problem with time-dependent supplies and capacities (see also [35]). A more detailed overview over network flow algorithms for certain settings of the evacuation problem is given in the survey article by Hamacher and Tjandra [36]. Also integer programming methods are used to improve evacuation planning (see for example [46, 79]). The work of Kalafatas and Peeta [46] deals with the problem of allowing flow to use lanes of a highway into the backward direction and thus increasing street capacity. Heuristic approaches that account to capacity constrained routing are, for example, given by Lu, Huang, and Shekhar [58]. They consider constant travel times and constant capacities and determine the order of people to get evacuated from a single

source via pre-computed shortest paths and the order of people out of several sources by iteratively re-calculating shortest travel times.

Another area of huge interest is the development of simulation tools to predict the global evacuation behavior by defining behavior rules for individuals either by probabilistic methods or by state definitions for cellular automatas. Well known simulation tools are EVACNET4 [48] for building evacuation and BYPASS [64] for the evacuation of passenger ships. A simulation for the evacuation of a football stadium using cellular automatas is presented in [52]. An analysis of pedestrian behavior needed for the evacuation simulation tools is given in [53]. Since this is a wide field with a huge amount of literature and projects, we refer to the survey articles of Hamacher and Tjandra [36] and Gwynne et al. [31] to obtain a deeper insight.

Chapter 3

Earliest Arrival Transshipments

3.1 Introduction

In Section 2.3 we already considered the earliest transshipment problem. Former approaches for earliest arrival transshipments only gave approximation results or algorithms for very special networks. Further, they assume that all supplies and demands are fixed. Various real time problems, as the problem of evacuating an apartment building, do not allow an exact knowledge of the number of people currently in the building. We can only estimate the number of people in each room and bound the supplies and demands from below or above, or both. We show in Section 3.2 that we can assume constant supplies and demands without loss of generality. By network modifications, each network having variable supplies and demands can be transformed into an equivalent network featuring constant supplies and demands. Moreover, we present the first exact algorithm for the earliest arrival transshipment problem for fixed supplies and demands whose running time is strongly polynomial in the input plus output size of the problem. As a by-product, we present a new proof for the existence of earliest arrival transshipments that does not rely on time-expansion. In Sections 3.3 and 3.4, we give an in-depth analysis of the structure of the earliest arrival pattern and present a recursive algorithm to compute it. The earliest arrival pattern is the pointwise maximal function of flow arriving at the single sink. Given the earliest arrival pattern, we show in Section 3.5 how to compute the actual earliest arrival transshipment. The running time of our algorithm can be determined as polynomial in the input size plus the number of breakpoints of the earliest arrival pattern which is part of the output. An alternative algorithm by Rauf [69] for a special case of the earliest arrival transshipment problem is described in Section 3.6. There we restrict to tight problems. Given the minimum time horizon for which an instance is feasible, it has to hold that the maximum amount that can be sent out of all sources together equals the sum of the supplies of the sources. Although, we can solve this problem with our approach, the direct reduction to the transshipment over time problem seems worth mentioning. In Section 3.7, we present some practical results that are

achieved by implementing the earliest arrival transshipment algorithm that computes a static min-cost s-t-flow in the time-expanded network. We especially concentrate on airplane evacuation and determine special network properties. Earliest arrival patterns will be computed for some airplane instances.

A substantial part of this chapter is based on joint work with Martin Skutella. An extended abstract will appear in [6]. The algorithm described in Section 3.6 is developed in the master thesis of Rauf [69].

3.2 Bounded Supplies and Demands

In real world applications, the amount of people in a site to evacuate and the capacity of safe sites can only be estimated. In some settings, the amount of people in a site to be evacuated is bounded from below, for example in office buildings publicly accessible. Upper bounded supplies can occur, for example, in dormitories, where the number of residents is known but not all are at home in an emergency. Thus, there are instances where we are only given lower and/or upper bounds for the number of people in an office and for the capacity available in safe sites. All known algorithms can only handle constant supplies and demands. Therefore we will describe how to transform networks with variable supplies and demands to equivalent networks having constant supplies and demands at sources and sinks, respectively. In those networks all known algorithms for networks with constant demands and supplies can be applied. This technique will also be adapted in the earliest arrival transshipment algorithm presented later in this chapter.

A real world setting where people need to get evacuated from unsafe sites to safe sites is modeled as a network having supplies and demands at sources and sinks, respectively, of one of the following types.

1. If the amount of people in sites to evacuate and the capacity of safe sites are exactly known, the supplies and demands of sources and sinks are fixed. We denote a fixed supply of such a source s by $d(s) > 0$ and the fixed demand of a sink t analogously by $d(t) < 0$. Let S_F^+ and S_F^- denote the set of sources with fixed supplies and the set of sinks with fixed demand, respectively, in case there are several types of supplies and demands in the network.

2. Assume the supply of a source is bounded from below by $d_L(s) > 0$, i.e., at least $d_L(s)$ flow units have to be sent out of source s over time. We assign a lower bounded demand $d_L(t) < 0$ to sink t. At least $-d_L(t)$ units of flow need to be sent into sink t. We denote these sets by S_L^+

and S_L^-, respectively.

3. If the amount of people in a site can only be estimated from above, the supply of such a source s determines an upper bound on the amount of flow that has to be sent out of s. We denote an upper bounded supply by $d_U(s) > 0$. Analogously, $d_U(t) < 0$ is called an upper bound of the demand of sink t if no more than $-d_U(t)$ units of flow are allowed to enter sink t over time. Let S_U^+ and S_U^- denote these sets of sources and sinks, respectively.

4. Lower and upper bounds of supplies and demands can also be given sources and sinks at the same time. The flow value sent over time out of such a source s needs to be within the interval $[d_L(s), d_U(s)]$. The total flow sent over time into sink t with given lower and upper bounds for the demand needs to have its value in $[-d_L(t), -d_U(t)]$. We denote these sets of sources and sinks by S_{LU}^+ and S_{LU}^-, respectively.

In the following, we assume that we are additionally given a fixed flow value $D > 0$, which needs to be sent over time through the network from sources into sinks. Using this value D, we can modify the network such that the lower and upper bounded supplies and demands can be used as fixed supplies and demands. In most optimization problems, the value of D is unknown. Then the modified network is a parameterized one with parameter D. The accurate value for D can then be found using parametric search or any other search method like binary search.

For the case of fixed values of the supply and demand, we know how to solve several problems. There it obviously holds that $D := \sum_{s \in S^+} d(s) = \sum_{t \in S^-} -d(t)$. For cases 2 to 4 of the enumeration above, we will determine network modifications where the resulting network only has fixed supplies and demands. Each flow in such a modified network can easily be transformed into a flow in the original network obeying the given lower and/or upper bounds of supplies and demands.

Let us first consider sources $s \in S_L^+$ whose supply is bounded from below by $d_L(s_i)$. Feasibility requires that $D \geq \sum_{s \in S^+} d(s)$. The goal is to send additionally needed flow into the network via all sources $s \in S_L^+$ when fixing the supplies to their lower bounds. Therefore, we set the fix supply of sources $s \in S_L^+$ to $d_L(s)$ and insert one additional node s_0 into the network which will function as an additional source. We assign a supply of $D - \sum_{s \in S_L^+} d_L(s) \geq 0$ to source s_0 which equals the difference of requested flow value D and the lower bound of the flow value that can be sent out of sources $s \in S_L^+$ over time. Further, we add uncapacitated, zero transit

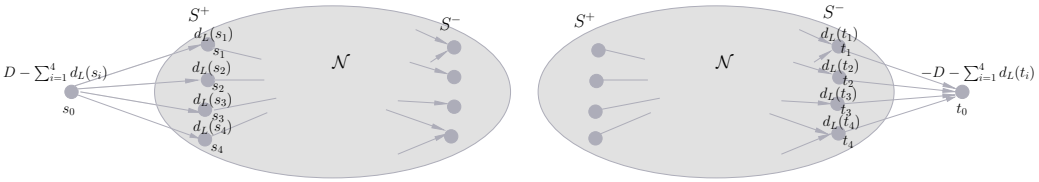


Figure 3.1: Modified networks where the given supplies at the sources (left hand side) and the demand at the sinks (right hand side) are fixed whereas in the original network only lower bounds are given.

time arcs from node s_0 to the sources $s \in S^+$. By the choice of the supply of s_0, any flow fulfilling the supplies in this new network sends at least D units of flow into the network out of and via sources $s \in S^+$. Deleting this additional node after the flow computation yields a flow that sends at least $d_L(s)$, $s \in S_L^+$, units of flow over time out of source s. A network with lower bounded supplies which is transformed to a network only having fixed supplies is depicted on the left hand side in Figure 3.1.

The modifications for sinks can be done analogously as depicted on the right hand side in Figure 3.1. Here, the additional flow units sent into sinks $t \in S_L^-$ having only a lower bounded demand are forwarded into the additional sink t_0 with demand $-D - \sum_{t \in S^-} d_L(t) \leq 0$ via uncapacitated, zero transit time arcs.

Consider the case, where supplies and demands are given as upper bounds of the real flow value that needs to be sent over time out of sources and reach the sinks over time, respectively. We will again fix the supplies and demands of sources and sinks to the value $d(s)$, $s \in S^+ \cup S^-$. Then the amount of flow sent out of sources and the amount of flow that reaches the sink over time can exceed the requested flow value, i.e., $D \leq \sum_{s \in S^+} d(s)$ and $D \leq \sum_{t \in S^-} -d(t)$, respectively. Therefore, in case of upper bounded supplies at sources, we need an additional sink t_0 directly connected from the sources by uncapacitated, zero transit time arcs (s, t_0), $s \in S_U^+$. We reroute the units of flow that are not needed to sent into this sink. The demand of sink t_0 is set to $D - \sum_{s \in S^+} d_U(s) \leq 0$ which equals the amount of flow not needed to send to original sinks. Such a modified network is given on the left hand side in Figure 3.2. The modifications for sinks can be done analogously, see on the right hand side in Figure 3.2. If a feasible flow can be found in this new network, then the solution can be adapted to a solution to the original problem with given upper bounds on the supply and demand.

For the last case, where we are given lower and upper bounds for a supply/demand of a source/sink, the modifications are a little bit more elaborate.

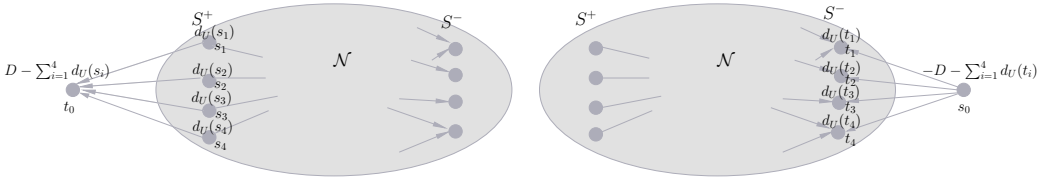


Figure 3.2: Modified networks where the given supplies at the sources (left hand side) and the demand at the sinks (right hand side) are fixed whereas in the original network only upper bounds are given.

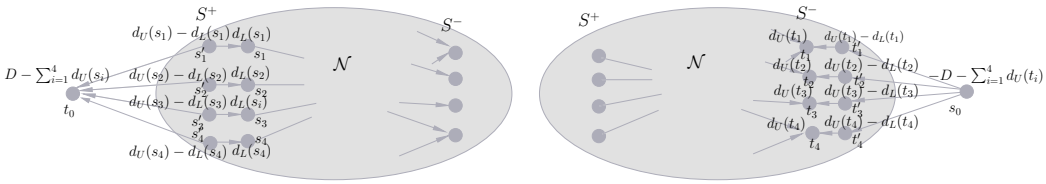


Figure 3.3: Modified networks where the supplies at the sources (left hand side) and the demand at the sinks (right hand side) are fixed whereas in the original network supplies and demands are given as intervals $[d_L(s_i), d_U(s_i)]$ for all sources s_i and $[-d_L(t_i), -d_U(t_i)]$ for sinks t_i, $i = 1, 2, 3, 4$.

Consider a source $s \in S^+_{LU}$ with a supply given as an interval $[d_L(s), d_U(s)]$. On the one hand, we need to ensure that at least $d_L(s)$ units of flow are sent out of s over time. On the other hand, we must guarantee that we do not send more than $d_U(s)$ units of flow over time out of the source. For each source $s \in S^+_{LU}$ with lower and upper bounded supply we add a copy s' of this source and connect it with the original source s by an uncapacitated, zero transit time arc (s', s). Node s' will function as an additional source which we assign a fixed supply $d_U(s) - d_L(s)$. For source s we fix the supply to $d_L(s)$. Further, we add one additional supersink t_0 with fixed demand $D - \sum_{s \in S^+_{LU}} d_U(s)$ and uncapacitated, zero transit time arcs (s', t_0) for all nodes s' added to the network. A flow over time of value exactly $d_L(s)$ has to be sent out of source s. Additional flow can be sent via s into the network out of the additional source s'. This additional amount is bounded by $d_U(s) - d_L(s)$ which is the remaining possible amount of flow to send. If the total remaining amount is not needed to yield a flow over time with value D, the surplus amount of flow can be transfered into sink t_0. Again the situation at sinks with lower and upper bounded demands is equivalent. Figure 3.3 shows the modified networks with fixed supplies and demands.

Since all the above transformations are independent in the sense that one source or sink does not interfere with another one during the modification, the sources and sinks of the network can have different types of given supplies or

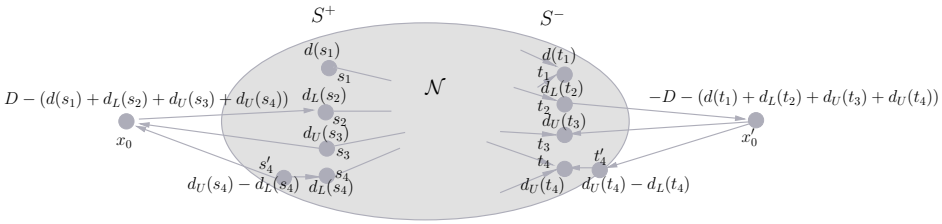


Figure 3.4: Modified network where each source and each sink is originally of different type of supplies and demands, respectively. The network was constructed using the modifications as described independently for each type.

demands, respectively. For each type of supply and demand, the appropriate modifications as described above are done. Notice, that we can no longer speak of supersources and supersinks which are additionally inserted. The supernode connected to all sources in $S_L^+ \cup S_U^+ \cup S_L^+U$ has supply or demand:

$$d(x_0) := D - \left(\sum_{s \in S_U^+} d_U(s) + \sum_{s \in S_L^+} d_L(s) + \sum_{s \in S_{LU}^+} d_U(s) + \sum_{s \in S_F^+} d(s) \right) .$$

The amount of required flow of this supernode can either be negative, positive, or zero, depending on the sum of individual given supplies of the nodes. Thus, it cannot be said in general whether this node is a source or a sink. The situation at the sinks is again equivalent. Figure 3.4 gives an impression of how such a network looks like.

In such a network, a flow can be transformed into a flow on the original network by deleting the additionally inserted nodes and arcs. For each source and sink the corresponding requirements on the supplies and demands are obviously fulfilled.

Theorem 3.1. Each network with supplies and demands which are bounded from below or above or both can be transformed into an equivalent network having fixed supplies and demands having additional sources or sinks. A flow computed in the modified network naturally induces a flow for the original network.

A similar construction is already described in the book of Ford and Fulkerson [25] for the *Caterer Problem*. Here, the caterer needs a certain amount of laundered napkins each day. Laundering causes costs and time, whereas purchasing only causes costs. The caterer only owns a certain amount of napkins and needs to buy some, if the laundering does not work quickly enough. This problem can be formulated as a network flow problem. The sources

model the beginning of the laundering and the sinks the day, when napkins are needed. Moreover, there exists an extra source directly connected to the sinks with infinite supply. This source models the purchase of napkins. It sends additionally needed flow into the sinks.

We showed in this section that the assumption of fixed supplies and demands is made without loss of generality. The general network with supplies and demands bounded from below or above can be transformed into a network having constant supplies and demands. In the following we will describe an algorithm that computes an exact solution for the earliest arrival transshipment problem. As already stated in Section 2.3 earliest arrival transshipments only exist in networks with a single sink. Therefore, networks having an upper bounded supply cannot be considered for the computation of earliest arrival transshipments since an additional sink is inserted. In the following, we restrict ourselves to networks where each source has a fixed supply and the single sink t has a fixed demand that equals the negative of the sum of the supplies of all sources. We start by analyzing earliest arrival patterns.

3.3 Earliest Arrival Pattern

The *earliest arrival pattern* $p : \mathbb{R}^+ \to \mathbb{R}^+$ is defined by setting $p(\theta)$ to the maximal amount of flow that can be sent into the sink by time θ without violating supplies at the sources, fulfilling capacity constraints and flow-conservation. Using the notion of the earliest arrival pattern we can define an *earliest arrival transshipment* to be a flow over time such that $p(\theta)$ units of flow have arrived at the sink by time θ for all $\theta \geq 0$ simultaneously.

For the case of a single source $S^+ = \{s\}$ with unbounded supply, the *s-t-earliest arrival pattern* is $p(\theta) = o^\theta(\{s\})$ and thus piecewise linear and convex. For the case of several sources, the earliest arrival pattern p is still piecewise linear (see Corollary 3.3 below) but not necessarily convex. A simple example with two sources is given in Figure 3.5. Notice that in this example the rate of flow arriving at the sink (i. e., the derivative of p) suddenly decreases since the entire supply of source s_1 has arrived at node t and this source has therefore run empty. In Section 3.4 we will observe this effect in a more general context.

The following lemma is essentially a reformulation of Lemma 2.2 for the setting of earliest arrival transshipments and will later turn out to be useful. Recall that $o^\theta(A)$, $A \subseteq S^+ \cup \{t\}$, denotes the maximum amount of flow that can be sent out of sources in A into sink t, if $t \notin A$, and zero otherwise, and d

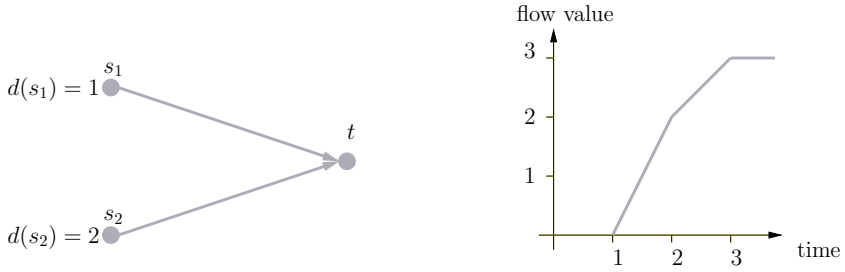


Figure 3.5: A simple example of a graph with two sources, unit capacities, and unit transit times where the optimal arrival pattern of a feasible earliest arrival transshipment is piecewise linear and non-convex.

the supply-demand-function.

Lemma 3.2. Let $\theta, q \geq 0$. Then $p(\theta) \geq q$ if and only if

$$o^\theta(S') \;\geq\; q - d(S^+ \setminus S') \qquad \text{for all } S' \subseteq S^+. \tag{3.1}$$

Proof. Consider an extended network $\mathcal{N}'$ with an additional sink t' that can be reached from any source by an uncapacitated, zero transit time arc. The demand of the new sink t' is defined to be $\bar{d}(t') := q - d(S^+)$ and the demand of the original sink t is set to $\bar{d}(t) := -q$. The supplies of sources remain unchanged, i.e., $\bar{d}(s) := d(s)$ for all $s \in S^+$. For $B \subseteq S^+ \cup \{t, t'\}$ let $\bar{o}^\theta(B)$ denote the maximum amount of flow that can be sent from the sources $S^+ \cap B$ to the sinks $\{t, t'\} \setminus B$. Notice that $p(\theta) \geq q$ if and only if the transshipment problem in the extended network $\mathcal{N}'$ with time horizon θ and supplies and demands $\bar{d}$ is feasible. By Lemma 2.2 this is the case if and only if

$$\bar{o}^\theta(B) \;\geq\; \bar{d}(B) \qquad \text{for all } B \subseteq S^+ \cup \{t, t'\}. \tag{3.2}$$

It remains to show that (3.1) holds if and only if (3.2) holds.
(3.1) $\Leftarrow$ (3.2): By definition of $\mathcal{N}'$ we get for $S' \subseteq S^+$

$$\begin{aligned} o^\theta(S') \;&=\; \bar{o}^\theta(S' \cup \{t'\}) \overset{(3.2)}{\geq} \bar{d}(S' \cup \{t'\}) \;=\; d(S') + q - d(S^+) \\ &=\; q - d(S^+ \setminus S') \;. \end{aligned}$$

(3.1) $\Rightarrow$ (3.2): Let $B \subseteq S^+ \cup \{t, t'\}$. We distinguish several cases. If $B \cap S^+ = \emptyset$, then $\bar{d}(B) \leq 0 \leq \bar{o}^\theta(B)$. We therefore assume from now on that $B \cap S^+ \neq \emptyset$. Since t' can be reached from every source node in S^+ by an uncapacitated arc with transit time zero, we get $o^\theta(B) = \infty > \bar{d}(B)$ if $t' \notin B$. We therefore assume from now on that $t' \in B$. If also $t \in B$, then $\bar{d}(B) \leq 0 \leq \bar{o}^\theta(B)$. It

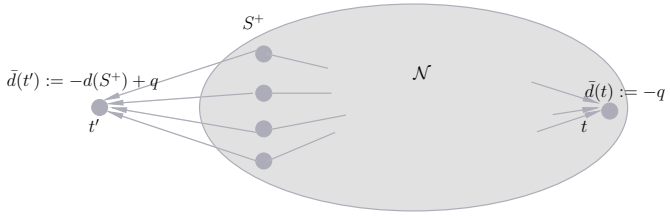


Figure 3.6: Network $\mathcal{N}'$ used in proof of Lemma 3.2. Obviously q needs to be smaller than or equal to $o^\theta(S^+)$ and for higher values of θ even smaller than or equal to $o^\theta(S')+d(S^+\setminus S')$ for some $S' \subset S^+$ to guarantee the feasibility of the earliest arrival transshipment problem in network $\mathcal{N}'$.

remains to consider the case that $B = S' \cup \{t'\}$ for some $S' \subseteq S^+$. In this setting we get

$$\bar{o}^\theta(B) \;=\; o^\theta(S') \;\overset{(3.1)}{\geq}\; q - d(S^+ \setminus S') \;=\; d(S') + q - d(S^+) \;=\; \bar{d}(B) \;.$$

This concludes the proof. □

In network $\mathcal{N}'$ used in the proof of Lemma 3.2 we use $d(t) := -q$ as an upper bound on the demanded flow that can enter the sink over time. The remaining flow, namely $d(S^+) - q$ flow units, enters the additional sink that is connected from all sources via uncapacitated, zero transit time arcs. This transformation is already shown in Section 3.2 where the flow value D is defined as $-q$. See also Figure 3.6 for a comparison.

As a consequence of Lemma 3.2, we can show that the earliest arrival pattern is a piecewise linear function.

Corollary 3.3. The earliest arrival pattern p is piecewise linear.

Proof. As a result of Lemma 3.2 we get

$$p(\theta) \;=\; \min\{o^\theta(S') + d(S^+ \setminus S') \mid S' \subseteq S^+\} \;.$$

Since $\theta \mapsto o^\theta(S')$ is a piecewise linear (and convex) function for all $S' \subseteq S^+$, the result follows. □

In the next section we show how we can determine the earliest arrival pattern of the earliest arrival transshipment problem. The earliest arrival transshipment itself can be obtained from the given earliest arrival pattern as shown in Section 3.5.

3.4 Constructing the Earliest Arrival Pattern

Throughout this section we use the following example instance to illustrate the presented ideas and techniques.

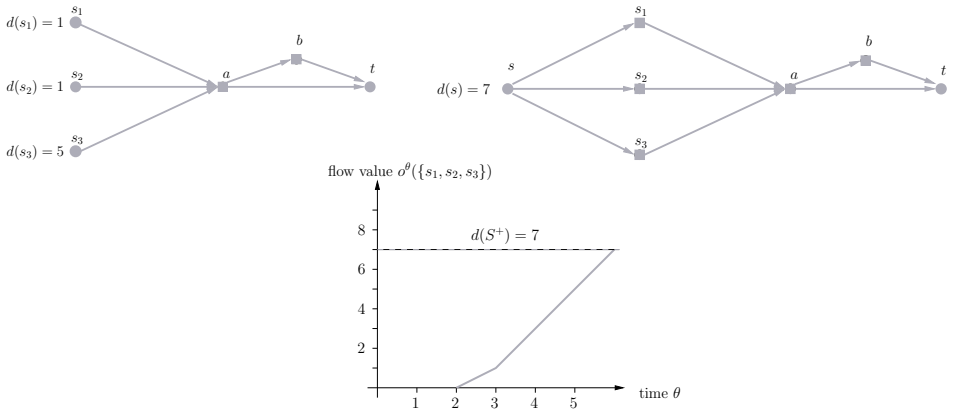


Figure 3.7: Example of a network $\mathcal{N} = (V, A)$, the network expanded by a supersource, and the corresponding s-t-earliest arrival pattern where individual supplies are ignored.

Example. *Assume we are given a network as depicted in the upper left corner of Figure 3.7 with unit transit times and unit capacities. The supplies of the sources are as given in the picture.*

3.4.1 The Structure of the Earliest Arrival Pattern

We show that the earliest arrival pattern p is composed of several s-t-earliest arrival patterns in extended networks with an additional supersource s that is connected to certain subsets of sources in S^+. We start by considering the extended network $\mathcal{N}_0$ that arises from connecting supersource s to all nodes in S^+ by an uncapacitated, zero transit time arc. The nodes in S^+ are no longer sources but take the role of intermediate nodes in $\mathcal{N}_0$ and their entire supply $d(S^+)$ is shifted to the supersource s. Thus, a feasible s-t-flow over time in the extended network $\mathcal{N}_0$ induces a flow over time in $\mathcal{N}$ where $d(S^+)$ units of flow are being sent from the sources in S^+ to sink t. Notice, however, that the induced flow over time in $\mathcal{N}$ might violate individual supplies at the source nodes.

The s-t-earliest arrival pattern in $\mathcal{N}_0$ is the function $\theta \mapsto o^\theta(S^+)$. For every $\theta \geq 0$ it holds that $p(\theta) \leq o^\theta(S^+)$. If $p(\theta) = o^\theta(S^+)$ for all $\theta \geq 0$, we are done since we know how to obtain the s-t-earliest arrival pattern $\theta \mapsto o^\theta(S^+)$. Otherwise, let $\theta_1 := \sup\{\theta \mid p(\theta) = o^\theta(S^+)\}$.[1] We use the following lemma to prove that $p(\theta) = o^\theta(S^+)$ for all $0 \leq \theta \leq \theta_1$.

[1] The supremum here is indeed a maximum since $p(\theta)$ and $o^\theta(S^+)$ are both continuous functions of θ.

Lemma 3.4. Let $S'' \subseteq S' \subseteq S^+$ and $0 \leq \theta' \leq \theta$. Then,

$$o^{\theta'}(S') - o^{\theta'}(S'') \;\leq\; o^{\theta}(S') - o^{\theta}(S'') \;.$$

Proof. Consider an extended network $\mathcal{N}'$ with an additional sink t' that can be reached from t through an uncapacitated arc (t, t') with transit time $\theta - \theta'$. The underlying intuition is that all flow arriving at t before time θ' can be forwarded to the new sink t' where it arrives before time θ. For $\bar{S} \subseteq S^+ \cup \{t, t'\}$ let $\bar{o}^{\theta}(\bar{S})$ denote the maximum amount of flow that can be sent from the sources in $\bar{S}$ to the sinks in $(S^+ \cup \{t, t'\}) \setminus \bar{S}$ by time θ. By construction of $\mathcal{N}'$ we get for $\bar{S} \subseteq S^+$ the following equalities:

$$\bar{o}^{\theta}(\bar{S}) \;=\; o^{\theta}(\bar{S}) \qquad \text{and} \qquad \bar{o}^{\theta}(\bar{S} \cup \{t\}) \;=\; o^{\theta'}(\bar{S}) \;. \tag{3.3}$$

We can now prove the statement of the lemma. By (3.3) and submodularity of $\bar{o}^{\theta}(\cdot)$ we get

$$\begin{aligned} o^{\theta'}(S') - o^{\theta'}(S'') \;&=\; \bar{o}^{\theta}(S' \cup \{t\}) - \bar{o}^{\theta}(S'' \cup \{t\}) \\ &\leq\; \bar{o}^{\theta}(S') - \bar{o}^{\theta}(S'') \\ &=\; o^{\theta}(S') - o^{\theta}(S'') \;. \end{aligned}$$

This concludes the proof. □

Corollary 3.5. Let $\theta_1 = \max\{\theta \mid p(\theta) = o^{\theta}(S^+)\}$. Then $p(\theta) = o^{\theta}(S^+)$ for all $0 \leq \theta \leq \theta_1$.

Proof. Assume by contradiction that $p(\theta) < o^{\theta}(S^+)$ for some $0 \leq \theta < \theta_1$. By Lemma 3.2 there exists $S' \subseteq S^+$ with

$$o^{\theta}(S') \;<\; o^{\theta}(S^+) - d(S^+ \setminus S') \;.$$

It follows from Lemma 3.4 that

$$o^{\theta_1}(S') \;<\; o^{\theta_1}(S^+) - d(S^+ \setminus S')$$

such that $p(\theta_1) < o^{\theta_1}(S^+)$ by Lemma 3.2. This contradicts the choice of θ_1. □

Example. *In order to compute the s-t-earliest arrival pattern for the network given in the left part of Figure 3.7 we insert a supersource s as depicted in the upper right corner of Figure 3.7. Applying the successive shortest path algorithm to this network yields, for example, the two paths* $P_1 = (s, s_1, a, t)$

and $P_2 = (s, s_3, a, b, t)$*, both with flow rate* 1*. The resulting arrival pattern up to time* 6 *is given in the lower part of Figure 3.7.*

Notice that the flow arriving at sink node t *after time* 3 *violates the supply of node* s_1 *since more than one unit of flow has been sent through path* P_1*. On the other hand, it can easily be seen that we can reroute the flow gaining a path decomposition with* $P_1' = (s, s_3, a, t)$*,* $P_2' = (s, s_1, a, b, t)$*, and* $P_3' = (s, s_2, a, b, t)$ *where the flow rate on path* P_1' *is* 1 *and the flow rates on paths* P_2' *and* P_3' *are only* 1/2*. Notice that the flow arriving over these paths at the sink does not violate supplies up to time* 5 *and has still the same arrival pattern. Further, there is no other way of sending flow obeying the supplies of sources* s_1, s_2, s_3 *for longer than* 5 *time units. After time* 5 *the slope of the earliest arrival pattern* p *decreases since no more flow out of sources* s_1 *and* s_2 *can reach the sink. In particular, the value of* θ_1 *equals* 5*.*

In our example, any flow over time in $\mathcal{N}$ that sends $p(\theta_1)$ units into the sink t by time θ_1 must use up the supplies of sources s_1 and s_2. In other words, the bounded flow values over time determined by the supplies of these sources are the reason why $p(\theta) < o^{\theta}(S^+)$ for $\theta > \theta_1$. The next lemma illuminates this effect for general instances.

Lemma 3.6. There exists a subset of sources $S_1 \subsetneq S^+$ such that

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

Before we prove the lemma, we first give an intuitive interpretation of its statement. In an earliest arrival transshipment, $p(\theta_1) = o^{\theta_1}(S^+)$ units of flow reach the sink by time θ_1. The lemma states that at most $o^{\theta_1}(S^+) - d(S^+ \setminus S_1)$ of these units can originate from sources in S_1. The remaining $d(S^+ \setminus S_1)$ units must originate from sources in $S^+ \setminus S_1$. These sources therefore run empty and cannot contribute to flow arriving after time θ_1 at the sink.

Proof. By contradiction assume that

$$o^{\theta_1}(S') \;>\; o^{\theta_1}(S^+) - d(S^+ \setminus S') \qquad \text{for all } S' \subsetneq S^+.$$

Since $o^{\theta}(S')$ and $o^{\theta}(S^+)$ are continuous functions of θ, there exists $\epsilon > 0$ such that

$$o^{\theta_1+\epsilon}(S') \;\geq\; o^{\theta_1+\epsilon}(S^+) - d(S^+ \setminus S') \qquad \text{for all } S' \subseteq S^+.$$

By Lemma 3.2 this implies $p(\theta_1 + \epsilon) \geq o^{\theta_1+\epsilon}(S^+)$. This contradicts the choice of θ_1. □

We consider the reduced instance of the earliest arrival transshipment problem that is obtained by setting the supplies of all sources in $S^+ \setminus S_1$ to zero. The earliest arrival pattern of the modified instance is denoted by p'. The following theorem is the main result of this section.

Theorem 3.7. Let $\theta_1 = \max\{\theta \mid p(\theta) = o^\theta(S^+)\}$ and $S_1 \subsetneq S^+$ such that $o^{\theta_1}(S_1) = o^{\theta_1}(S^+) - d(S^+ \setminus S_1)$ (see Lemma 3.6). Let p' denote the earliest arrival pattern of the modified instance with source set S_1. Then,

$$p(\theta) = \begin{cases} o^\theta(S^+) & \text{if } \theta < \theta_1, \\ p'(\theta) + d(S^+ \setminus S_1) & \text{if } \theta \geq \theta_1. \end{cases}$$

Proof. It follows from Corollary 3.5 that $p(\theta) = o^\theta(S^+)$ for $\theta \leq \theta_1$. It remains to show that

$$p(\theta) = p'(\theta) + d(S^+ \setminus S_1) \qquad \text{for all } \theta \geq \theta_1.$$

It is clear that "$\leq$" holds since by time θ at most $p'(\theta)$ and $d(S^+ \setminus S_1)$ units of flow can reach the sink originating from sources in S_1 and $S^+ \setminus S_1$, respectively.

It remains to show that "$\geq$" holds, that is, $p'(\theta) + d(S^+ \setminus S_1)$ units of flow can be sent into the sink t by time $\theta \geq \theta_1$ without exceeding supplies at the sources. We check the condition given in Lemma 3.2. For $S' \subseteq S^+$ and $\theta \geq \theta_1$ we get by submodularity of $o^\theta(\cdot)$:

$$o^\theta(S') \geq o^\theta(S' \cap S_1) + o^\theta(S' \cup S_1) - o^\theta(S_1)$$

by Lemma 3.4:

$$\geq o^\theta(S' \cap S_1) + o^{\theta_1}(S' \cup S_1) - o^{\theta_1}(S_1)$$

by Lemma 3.2 and Lemma 3.6:

$$\begin{aligned} &\geq \Big(p'(\theta) - d(S_1 \setminus S')\Big) + \Big(o^{\theta_1}(S^+) - d\big(S^+ \setminus (S' \cup S_1)\big)\Big) \\ &\quad - \Big(o^{\theta_1}(S^+) - d(S^+ \setminus S_1)\Big) \\ &= p'(\theta) - d(S_1 \setminus S') - d\big(S^+ \setminus (S' \cup S_1)\big) + d(S^+ \setminus S_1) \\ &= p'(\theta) - d(S^+ \setminus S') + d(S^+ \setminus S_1) \ . \end{aligned}$$

The result now follows from Lemma 3.2. □

As a result of Theorem 3.7, we have reduced the problem of constructing the earliest arrival pattern p to the problem of computing an s-t-earliest arrival pattern and computing an earliest arrival pattern for a smaller number of sources S_1.

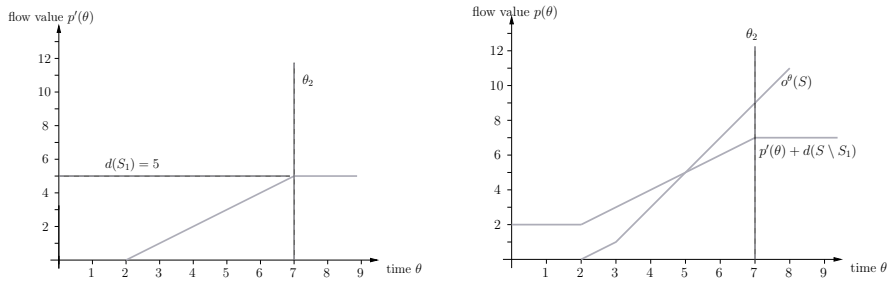


Figure 3.8: Optimal pattern p' for the problem with the reduced set of sources $S_1 = \{s_3\}$ (left) and the combined pattern p as the lower bound of the line segments (right).

Example. *Concerning our example given in Figure 3.7 we have already seen that up to time $\theta_1 = 5$ flow of value 5 including the total supply of the sources s_1 and s_2 can be sent into the sink. In particular, it holds that $o^\theta(S') \geq o^\theta(S^+) - d(S^+ \setminus S')$ for all $S' \subseteq S^+$ and $\theta \leq \theta_1$. For the set $S_1 := \{s_3\} \subseteq S^+$ and $\theta = \theta_1$ this inequality is tight. The function $\theta \mapsto o^\theta(S^+)$ is already known (see the lower part of Figure 3.7).*

For the restricted earliest arrival problem with sources $S_1 = \{s_3\}$, the earliest arrival pattern p' is given in the left part of Figure 3.8. By Theorem 3.7, the resulting earliest arrival pattern p of the original instance is the lower envelop of the two functions depicted in the right part of Figure 3.8.

3.4.2 Computing the Earliest Arrival Pattern

Theorem 3.7 reduces the problem of computing the earliest arrival pattern to an earliest arrival s-t-flow problem and an earliest arrival transshipment problem on a reduced instance with a strictly smaller set of sources. Applying this result recursively to the reduced instance finally yields Algorithm 1 which computes the earliest arrival pattern p.

For the understanding of the algorithm it is helpful to observe that $\theta_i < \theta_{i+1}$ for all $i \geq 0$. The statement is clear for $i = 0$ since the sources in $S^+ \setminus S_1$ have positive supply and therefore cannot run empty at time $\theta_0 = 0$. For $i \geq 1$ assume by contradiction that $\theta_{i+1} \leq \theta_i$. This yields

$$\begin{aligned}
o^{\theta_i}(S_{i+1}) &\leq o^{\theta_i}(S_i) + o^{\theta_{i+1}}(S_{i+1}) - o^{\theta_{i+1}}(S_i) && \text{by Lemma 3.4,} \\
&= o^{\theta_i}(S_i) - d(S_i \setminus S_{i+1}) && \text{by (3.4),} \\
&= o^{\theta_i}(S_{i-1}) - d(S_{i-1} \setminus S_i) - d(S_i \setminus S_{i+1}) && \text{by (3.4) with } i := i-1, \\
&= o^{\theta_i}(S_{i-1}) - d(S_{i-1} \setminus S_{i+1})
\end{aligned}$$

which contradicts the minimal choice of $S_i \supsetneq S_{i+1}$ in step 4 of the algorithm.

Algorithm 1: Computing the earliest arrival pattern.

Input: (G, S^+, t)
Output: Earliest arrival pattern p.

1 set $i := 0$, $S_i := S^+$, and $\theta_i := 0$;
2 **while** $S_i \neq \emptyset$ **do**
3 compute the maximal value $\theta_{i+1} \geq 0$ such that

$$o^{\theta_{i+1}}(S') \;\geq\; o^{\theta_{i+1}}(S_i) - d(S_i \setminus S') \qquad \text{for all } S' \subseteq S_i;$$

4 compute an inclusion-wise minimal $S_{i+1} \subsetneq S_i$ with

$$o^{\theta_{i+1}}(S_{i+1}) \;=\; o^{\theta_{i+1}}(S_i) - d(S_i \setminus S_{i+1}) \;; \tag{3.4}$$

5 compute the function $\theta \mapsto o^{\theta}(S_i)$ on the interval $[\theta_i, \theta_{i+1})$ and set

$$p(\theta) \;:=\; o^{\theta}(S_i) + d(S^+ \setminus S_i) \qquad \text{for } \theta \in [\theta_i, \theta_{i+1});$$

6 $i := i + 1$;
7 set $p(\theta) := d(S^+)$ for all $\theta \geq \theta_i$;

Theorem 3.8. Algorithm 1 computes the earliest arrival pattern and can be implemented to run in strongly polynomial time in the input plus output size.

In order to prove this theorem, we need the following technical lemma which gives a bound on the computational complexity of step 5.

Lemma 3.9. For $0 \leq \theta_i \leq \theta_{i+1}$ and $S' \subseteq S^+$, the piecewise linear function $g : [\theta_i, \theta_{i+1}) \to \mathbb{R}$ with $g(\theta) := o^{\theta}(S')$ can be computed in time polynomial in the input size plus the number of breakpoints.

Proof. In order to compute $g(\theta) = o^{\theta}(S')$, we consider the extended network $\mathcal{N}'$ that is obtained as follows. Add a supersource s that is connected to all sources in S' by an uncapacitated arc with transit time zero and that can be reached from t by an uncapacitated dummy arc (t, s). As already stated in (2.3), $g(\theta)$ is equal to the negative of the cost of a static min-cost circulation in $\mathcal{N}'$ where the cost coefficient of the dummy arc (t, s) is set to $\tau(t, s) = -\theta$. We denote the cost of an arbitrary circulation x in this network by $\text{cost}^{\theta}(x)$.

We start by computing a static min-cost circulation x in $\mathcal{N}'$ for $\theta = \theta_i$. Let $\mathcal{N}'_x$ denote the residual network of x and let θ' be the length of a shortest

s-t-path in $\mathcal{N}'_x$. Since there is the uncapacitated dummy arc (t, s) of cost $-\theta_i$ in $\mathcal{N}'_x$, optimality of x implies $\theta' \geq \theta_i$. Moreover, for all $\theta \in [\theta_i, \theta']$, the circulation x is still a static min-cost circulation and $g(\theta) = -\text{cost}^\theta(x)$. Since the cost of x depends linearly on θ, the function g is thus linear on the interval $[\theta_i, \theta']$. If $\theta' \geq \theta_{i+1}$, then we are done. Otherwise we have discovered a breakpoint of g at θ'. Notice that x is no longer optimal for $\theta > \theta'$ since the cost can be reduced by augmenting flow on a negative cycle formed by a shortest s-t-path of length θ' in $\mathcal{N}'_x$ and the dummy arc (t, s) of length $-\theta$.

We obtain the next linear piece of g starting at θ' as follows. Compute the subnetwork $\mathcal{N}''_x$ of the residual network $\mathcal{N}'_x$ that is formed by all arcs that lie on some shortest s-t-path. Compute a maximum static s-t-flow in $\mathcal{N}''_x$ and turn it into a circulation y in $\mathcal{N}'_x$ by sending all flow from t back to s on the dummy arc (t, s). Augmenting x according to y yields a new circulation x. The new circulation is optimal for all $\theta \in [\theta', \theta'']$ where $\theta'' > \theta'$ is the length of a shortest s-t-path in the new residual network $\mathcal{N}'_x$, x the actual circulation, and determines the next breakpoint of g.

The described process is iterated until the length of a shortest s-t-path in the residual network is at least θ_{i+1}. Notice that the overall running time is dominated by the initial static min-cost s-t-flow computation plus number of breakpoints many maximum static s-t-flow computations. □

Example. *In our example depicted in Figure 3.7 we can find the function $g : [\theta_i, \theta_{i+1}) \to \mathbb{R}$ as described above. For the interval $[\theta_0, \theta_1)$ we get the networks $\mathcal{N}', \mathcal{N}'_x$, and $\mathcal{N}''_x$ as follows. Network $\mathcal{N}'$ is constructed by adding a supersource s connected to all sources by uncapacitated, zero transit time arcs and an uncapacitated arc (t, s) with transit time $\tau(t, s) = -2$. This is depicted in Figure 3.9.1. In this network we compute a static min-cost (maximum) circulation by sending one unit of flow for example over cycle s, s_2, a, t, s. This yields the residual network $\mathcal{N}'_x$ which is depicted in Figure 3.9.2. There the shortest s-t-path, for example path s, s_3, a, b, t, has length $\theta' = 3$. In the subnetwork $\mathcal{N}''_x$ consisting of all arcs being part of some shortest s-t-path we now compute a maximum static s-t-flow. Such a path flow s, s_1, a, b, t is depicted in Figure 3.9.3. Reconsidering network $\mathcal{N}'_x$ together with a new circulation of one unit of flow along cycle s, s_1, a, b, t, s results in the new residual network $\mathcal{N}'_x$(new) which is depicted in Figure 3.9.4. There no (shortest) s-t-path remains and therefore the transit time of arc (t, s) is set to infinity which is strictly greater than θ_1. Thus we have found function g on the interval $[\theta_0, \theta_1)$ which is of the form shown in Figure 3.9.5.*

Proof of Theorem 3.8. The correctness of the algorithm follows from Section 3.4.1 and in particular from Theorem 3.7. It thus remains to prove the

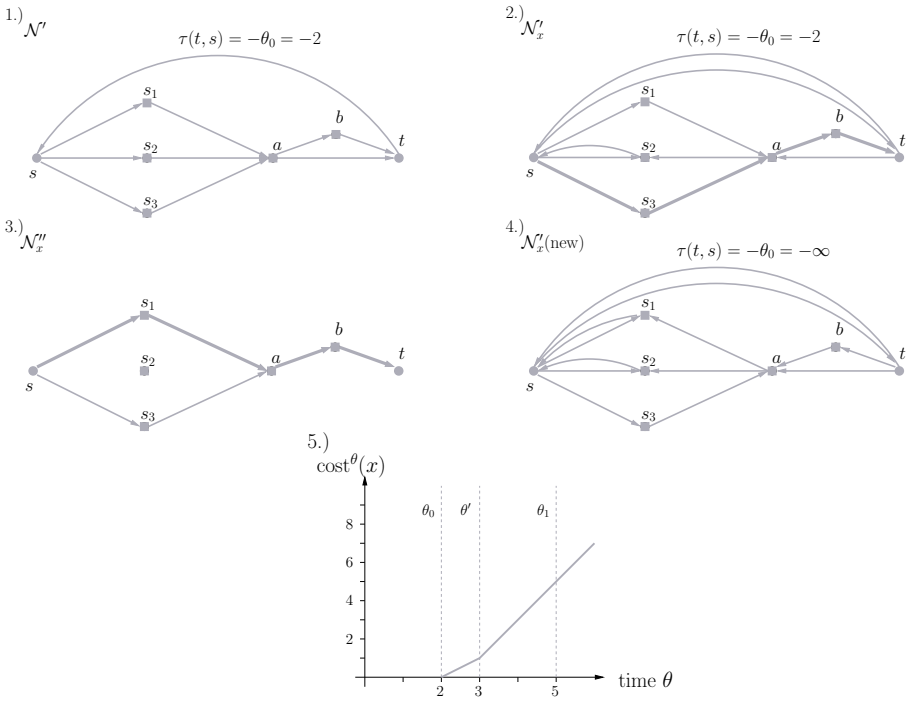


Figure 3.9: Networks used to compute the function $g : [\theta_0, \theta_1) \to \mathbb{R}$ as described in the proof of Lemma 3.9 for the network given in Figure 3.7.

stated bound on the running time of Algorithm 1.

First notice that the number of iterations of the while-loop in step 2 is bounded by the number of sources since at least one source is eliminated from S_i in every iteration. Since step 5 can be done in strongly polynomial time, it remains to show that steps 3 and 4 can also be done in strongly polynomial time.

We start with the computation of θ_{i+1} in step 3. For $\theta \geq 0$ we define the function $f^\theta : 2^{S_i} \to \mathbb{R}$ by $f^\theta(S') := o^\theta(S') - o^\theta(S_i) + d(S_i \setminus S')$ for $S' \subseteq S_i$. Computing θ_{i+1} thus amounts to finding the maximal value $\theta \geq 0$ such that

$$f^\theta(S') \;\geq\; 0 \qquad\qquad \text{for all } S' \subseteq S_i. \tag{3.5}$$

Since o^θ is submodular and the function $S' \mapsto d(S_i \setminus S') - o^\theta(S_i)$ is modular, f^θ is submodular. According to equation (2.3), computing $f^\theta(S')$ for some $S' \subseteq S_i$ requires two static min-cost flow computations where the cost coefficients depend linearly on the parameter θ. It can be tested in strongly polynomial time whether (3.5) is fulfilled for a fixed value θ. Embedding this algorithm into Megiddo's parametric search framework gives a procedure for step 3 whose running time is strongly polynomial in the input size of our problem.

We finally discuss how to compute S_{i+1} in step 4 in strongly polynomial time. Notice that (3.4) translates to $f^{\theta_{i+1}}(S_{i+1}) = 0$, that is, S_{i+1} minimizes the submodular function $f^{\theta_{i+1}}$. By submodularity of $f^{\theta_{i+1}}$, there exists a unique inclusion-wise minimal subset S_{i+1} which can be obtained as follows[2] (for details see, e.g., [72, Chapter 45]). Initialize $S_{i+1} := S_i$. For each $s \in S_i$, check whether the minimum value of $f^{\theta_{i+1}}$ over all subsets of $S_{i+1} \setminus \{s\}$ is zero. If so, reset $S_{i+1} := S_{i+1} \setminus \{s\}$. Doing this for all elements of S_i finally yields the unique inclusion-wise minimal subset S_{i+1} with $f^{\theta_{i+1}}(S_{i+1}) = 0$. A faster algorithm for computing the inclusion-wise minimal subset S_{i+1} was recently given by McCormick and Queyranne [60]. □

3.4.3 Reformulation of the Algorithm

In order to get rid of at least some submodular function minimization which is not practical at all, we reformulate the algorithm described above avoiding submodular functions. Notice, that we are not able to get rid of checking the feasibility criterion (2.1) of Klinz using the following method.

As described in the proof of Theorem 3.8, step 3 of Algorithm 1 can be seen as minimizing a submodular function within a parametric search. The submodular function f has to evaluate two static min-cost s-t-flow computations for each considered set S' in network $\mathcal{N}$ extended by a supersource. Step 4 directly uses the minimization of a submodular function and determines the set at which this minimum occurs. Both steps can be rewritten as computing static min-cost s-t-flows in a slightly modified network with parameterized supplies and demands. In the following we will describe the network modification.

We know by Corollary 3.5 that the value for the parameter q in Lemma 3.2 cannot exceed $o^\theta(S_i)$. As in the proof of this lemma we can determine $\theta_{i+1} := \max\{\theta \mid p(\theta) = o^\theta(S_i) + d(S^+ \setminus S_i)\}$ by considering network $\mathcal{N}'$ (see left part of Figure 3.10). An additional sink t' is inserted into $\mathcal{N}$ to which flow is transfered that cannot reach sink t by time θ_{i+1}. The value θ_{i+1} functions as a parameter. The demand $-q$ of sink t is set to $\bar{d}(t) := -o^{\theta_{i+1}}(S_i)$ and the demand of the additional sink t' is set to $\bar{d}(t) := -d(S_i) - \bar{d}(t)$. In this parameterized network $\mathcal{N}'$ we can parametrically search for the maximum θ_{i+1}. We only need to check the feasibility criterion (2.1), i.e., $o^\theta(X) - d(X) \geq 0$, for a fixed θ in each step of the parametric search. For a given set X, the value of $o^\theta(X)$ can be determined by one static min-cost s-t-flow computation as described in the proof of Lemma 2.2.

[2] For the purpose of our algorithm it is, of course, advantageous to choose the minimal subset S_{i+1} in order to reduce the number of sources as far as possible.

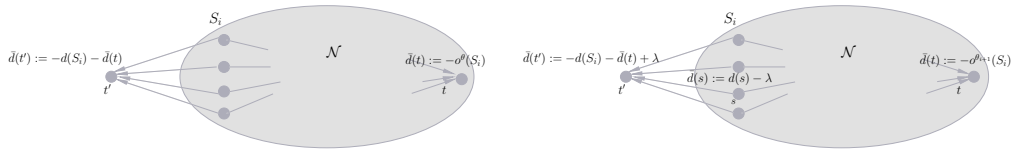


Figure 3.10: On the left hand side, we draw the network $\mathcal{N}$ extended by an additional sink and parameterized demands of the sources and sinks, respectively. It is parameterized by the time horizon θ. The supply of the sources stays the same. Used to compute θ_{i+1}. On the right hand side, we draw network $\mathcal{N}$ extended by an additional sink and parameterized demands of the sources and sinks, respectively. It is parameterized by the decrease of the supply of one source $s \in S_i$. The supply of the remaining sources in $S_i \setminus \{s\}$ stays the same. This network is used to compute the inclusionwise minimal set S_{i+1}.

Hence, extending the network reduces the problem of repeatedly minimizing the submodular function f^θ to repeatedly checking the feasibility criterion (2.1). That is, we need one static min-cost s-t-flow computation per considered set. This procedure is obviously strongly polynomial since only the parametric search and the check of the feasibility criterion (2.1) are used.

For a reformulation of step 4 of Algorithm 1 consider again network $\mathcal{N}'$. For one source $s \in S_i$ we change the supply to $\bar{d}(s) := d(s) - \lambda$. We set the demand of sink t to $\bar{d}(t) := -o^{\theta_{i+1}}(S_i)$ and the demand of sink t' to $\bar{d}(t') := -d(S_i) - \bar{d}(t) + \lambda$ (see on the right hand side in Figure 3.10).

Now we parametrically search the maximal λ for which the problem is still feasible, i.e., we check the feasibility criterion (2.1) for each fixed λ within the parametric search.

If the value for λ is strictly greater than zero, we know that there is a flow in the network $\mathcal{N}'$ sending the maximum possible amount $o^{\theta_{i+1}}(S_i)$ into the sink up to time θ_{i+1} by sending less flow than $d(s)$, namely $d(s) - \lambda$, out of source s within the time interval $[\theta_i, \theta_{i+1})$. If the determined value of λ for source s equals zero, then flow of value exactly $d(s)$ out of this source is necessary to gain the maximum flow value $o^{\theta_{i+1}}(S_i)$ in a feasible way. So this source has to send its total supply into the sink up to time θ_{i+1}. There exists no other flow in the network sending the same amount of flow into the sink without emptying source s.

Since we use the strongly polynomial parametric search only once to find the maximal λ-value for each source, we can check whether a source in S_i has to be emptied in strongly polynomial time.

Applying the method as stated above for all sources one after another we obtain the set of sources $S_{i+1} := \{s \in S_i | \lambda > 0 \text{ for } \bar{d}(s) := d(s) - \lambda\}$.

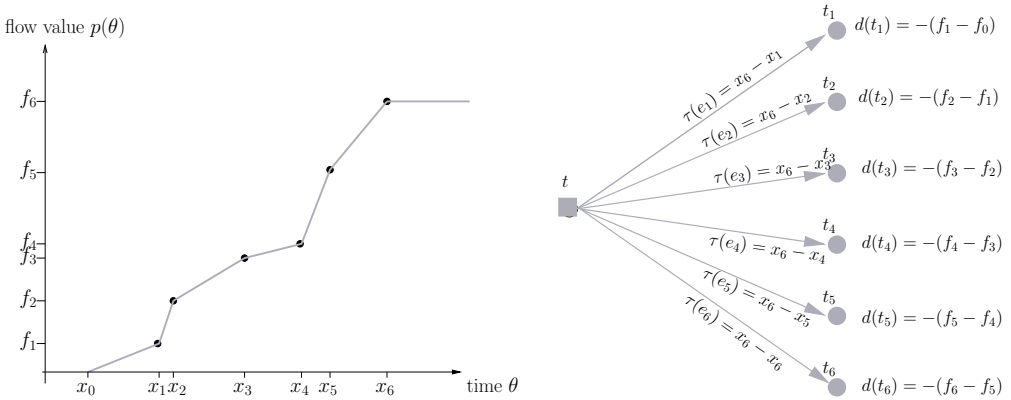


Figure 3.11: On the left hand side, we draw the earliest arrival pattern p with breakpoints (x_i, f_i), $i = 1, 2, \ldots, k = 6$. On the right hand side, the modified network is depicted. The capacity of arc $e_i = (t, t_i)$ is set to $(f_i - f_{i-1})/(x_i - x_{i-1})$.

The earliest arrival pattern p thus can also be computed using the above described techniques.

3.5 Turning the Earliest Arrival Pattern into an Earliest Arrival Transshipment

In this section we assume that we are given the piecewise linear earliest arrival pattern p of the earliest arrival transshipment problem by its breakpoints $(x_0, f_0), (x_1, f_1), \ldots, (x_k, f_k)$, that is,

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

An illustration is given in Figure 3.11. Notice that the values x_i determine points in time and the values f_i determine an amount of flow for all i.

Further notice that $x_0 < x_1 < \cdots < x_k$ and x_0 is the first point in time when flow can reach the sink (i. e., x_0 is the transit time of a shortest path leading from any source to the sink). Moreover, it holds $0 = f_0 \leq f_1 \leq \cdots \leq f_k = d(S^+)$.

We show that the problem of finding an earliest arrival transshipment can be reduced to finding a transshipment over time in a slightly modified network $\mathcal{N}'$ with k additional arcs leading from t to k new sink nodes $t_1, \ldots, t_k$. An illustration of the modification is given in Figure 3.11.

Node t is no longer a sink but just an intermediate node of the modified network $\mathcal{N}'$. For $i = 1, \ldots, k$, the demand of sink t_i is set to $-(f_i - f_{i-1})$

such that the total demand $-f_k$ of the sinks and the total supply $d(S^+)$ at the sources cancel out each other. The arc leading from t to sink t_i is called e_i. The transit time of arc e_i is defined to be $\tau(e_i) := x_k - x_i$, its capacity is $(f_i - f_{i-1})/(x_i - x_{i-1})$ and thus equal to the derivative of p within the interval $[x_{i-1}, x_i]$. Notice that the capacity of e_i is chosen such that the demand of sink t_i is fulfilled if flow is being sent at maximal rate into arc e_i within time interval $[x_{i-1}, x_i)$. As a consequence of this observation, we can state the following lemma.

Lemma 3.10. An earliest arrival flow in $\mathcal{N}$ with earliest arrival pattern p naturally induces a feasible transshipment over time with time horizon x_k, satisfying all supplies and demands in $\mathcal{N}'$

Proof. Take an earliest arrival flow in $\mathcal{N}$ and turn it into a transshipment over time in $\mathcal{N}'$ by sending all flow arriving at t in time interval $[x_{i-1}, x_i)$ to t_i along arc e_i. □

The reverse direction of Lemma 3.10 also holds:

Lemma 3.11. A transshipment over time with time horizon x_k that satisfies all supplies and demands in the modified network $\mathcal{N}'$ naturally induces an earliest arrival transshipment in $\mathcal{N}$.

Proof. We must prove that the flow passing through node t in an arbitrary feasible transshipment over time in $\mathcal{N}'$ with time horizon x_k meets the earliest arrival pattern p. Since, for $i = 1, \ldots, k$, the capacity of arc e_i equals the derivative of the earliest arrival pattern p in the time interval $[x_{i-1}, x_i]$ and since the negative of the demand of sink t_i is equal to this capacity times $x_i - x_{i-1}$, it suffices to prove the following claim:

Claim. For each $i = 1, \ldots, k$ the following holds: In a feasible transshipment over time the demand of sink t_i is satisfied by flow being sent into arc e_i within time interval $[x_{i-1}, x_i]$.

We prove this claim by induction on i. The case $i = 1$ is clear since no flow can arrive at node t before time x_0 and, due to the transit time $x_k - x_1$ of arc e_1, flow being sent into arc e_1 after time x_1 arrives at t_1 after time x_k. In order to prove the claim for $i > 1$ notice that the total amount of flow passing through node t before time x_{i-1} is bounded from above by $p(x_{i-1}) = f_{i-1}$. Since this amount is equal to the negative of the total demand of sinks $t_1, \ldots, t_{i-1}$, it follows by induction that all flow passing through t before time x_{i-1} must be used to satisfy the demands of $t_1, \ldots, t_{i-1}$. Thus, no flow is sent into arc e_i before time x_{i-1}. On the other hand, since the transit time of arc e_i is $x_k - x_i$, no flow is sent into this arc after time x_i. This concludes the proof. □

We finally prove that a transshipment over time with time horizon x_k that satisfies all supplies and demands in the modified network $\mathcal{N}'$ actually exists. As a consequence of Lemma 3.11, this yields a new proof for the existence of an earliest arrival transshipment in $\mathcal{N}$.

Lemma 3.12. There exists a transshipment over time with time horizon x_k satisfying all supplies and demands in $\mathcal{N}'$.

Proof. We denote the set of sources in $\mathcal{N}'$ by S^+ and the set of sinks by $S^- = \{t_1, \dots, t_k\}$. For an arbitrary $S' \subseteq S^+ \cup S^-$ let $\bar{o}^\theta(S')$ denote the maximum amount of flow that can be sent within time θ from sources $S^+ \cap S'$ to sinks $S^- \setminus S'$. By Lemma 2.2 we have to show that $\bar{o}^\theta(S') \geq d(S')$ for $\theta = x_k$.

Let $o^\theta(S^+ \cap S')$ denote the maximum amount of flow that can be sent within time θ from sources $S^+ \cap S'$ to t. By Lemma 3.4 we get

$$o^\theta(S^+ \cap S') + d(S^+ \setminus S') \geq p(\theta) \qquad \text{for all } \theta \geq 0.$$

This inequality can be interpreted as follows: If we assume that the total supply $d(S^+ \setminus S')$ of the sources $S^+ \setminus S'$ is already in t by time zero, then we can send $d(S^+ \cap S')$ additional flow units from the sources in $S^+ \cap S'$ (ignoring their individual supplies) into t such that the amount of flow at t is at least $p(\theta)$ at any time $\theta \geq 0$. By forwarding flow from t to the sinks in S^- (similar to the proof of Lemma 3.10), we get a flow over time with time horizon x_k that satisfies the demands of all sinks in S^-. From this flow over time we now remove the $d(S^+ \setminus S')$ flow units that we assumed to be in t at time zero. This yields a flow over time with time horizon x_k from the sources in $S^+ \cap S'$ to the sinks S^- such that the total amount of flow sent is $d(S^+ \cap S')$ and no sink in S^- gets more than its demand. Therefore the flow arriving at sinks in $S^- \setminus S'$ is at least $d(S^+ \cap S') + d(S^- \cap S') = d(S')$. We have thus shown that $\bar{o}^\theta(S') \geq d(S')$ for $\theta = x_k$. This concludes the proof. □

As a consequence we can state the following theorem.

Theorem 3.13. Given the earliest arrival pattern p with k breakpoints for network $\mathcal{N}$, an earliest arrival transshipment in $\mathcal{N}$ can be obtained by computing a transshipment over time in a modified network $\mathcal{N}'$ with k additional nodes and arcs.

In order to compute a feasible transshipment over time in the modified network $\mathcal{N}'$ we can use the algorithm of Hoppe and Tardos [40] as described in Section 2.2.1. Since the running time of this algorithm is bounded by a polynomial in the encoding size of the input $\mathcal{N}'$ and since the encoding size

of $\mathcal{N}'$ is of the same order as the encoding size of $\mathcal{N}$ plus the encoding size of p, the required running time is polynomial in the input plus output size of the earliest arrival flow problem on $\mathcal{N}$.

3.6 Tight Earliest Arrival Transshipments

In the following we consider the problem of *tight earliest arrival transshipments*. We denote earliest arrival transshipment problems as tight if the maximum amount that can be sent out of sources within the minimum time horizon θ^* equals the total supply. The tight problem is obviously a special case of the earliest arrival transshipment problem. Given such a tight earliest arrival transshipment problem, it directly follows by Corollary 3.5 that $o^\theta(S^+) = p(\theta)$ for all $\theta \leq \theta^*$. Therefore, Algorithm 1 finds the minimum time horizon $\theta^* = \theta_1$ and the earliest arrival pattern in its first iteration. A computation of a feasible transshipment in the network extended by several sinks as described in Section 3.5 determines the earliest arrival transshipment itself.

Although, we are able to solve the tight version of the problem we present an alternative algorithm due to Rauf [69] in the following. In this algorithm, the network modifications as described by Hoppe and Tardos [40] and shortly resumed in Section 2.2.1 for the quickest transshipment problem can be applied directly. In contrast to the reduced transshipment over time problem, the lexicographically maximum flow over time algorithm cannot be used in the way stated there, since it does not obey the earliest arrival property. The adaption of this algorithm to find a lexicographically earliest arrival flow is described in the following.

We first start with the formal definition of tight earliest arrival transshipment problems.

Definition 3.14. We call an earliest arrival transshipment problem *tight* if there exists a time θ^* such that $d(S^+) = o^{\theta^*}(S^+)$ and $d(X) \leq o^{\theta^*}(X)$ for all $X \subseteq S^+ \cup \{t\}$.

In order to apply the quickest transshipment algorithm described by Hoppe and Tardos [40] (and shortly revised in Section 2.2.1), first the minimum time horizon θ^*, which is necessary to send the total supply into the sink, need to be found.

As already described above, the minimum time horizon equals θ_1, which can be found by maximizing θ such that $o^\theta(S') - o^\theta(S^+) + d(S^+ \setminus S') \geq 0$ for all $S' \subseteq S^+$. Since the problem is tight, the above equation reduces to the feasibility criterion (2.1). Therefore, it suffices to check whether $o^\theta(S') \geq$

$d(S')$ for all $S' \subseteq S^+$ for a fixed value of θ. This can be done in strongly polynomial time as described in the previous sections.

Given the minimum time horizon θ^*, in a first phase the network is transformed. We use exactly the same algorithm as the one given by Hoppe and Tardos by inserting new sources and defining a chain $\mathcal{C}$ of nested subsets $S_0 \subsetneq S_1 \subsetneq \cdots \subsetneq S_\ell$ of sources of the network. This chain has the property that all subsets $S_i, i = 0, \ldots, \ell$, are tight as well, i. e., $o^{\theta^*}(S_i) = d(S_i)$. A more detailed description can be found in Section 2.2.1.

Since we are given instances having only one sink and we know that the problem is tight, we also know that set $S^+ \cup \{t\}$ is tight as well as set S^+. Considering those sets as adjacent sets in chain $\mathcal{C}$, sink t will never be touched during the modification. So, only nodes s' which are connected to sources $s \in S^+$ via uncapacitated, zero transit time arcs are inserted to build the set of sources S'^+ in this new network. We obtain a network $\tilde{\mathcal{N}}$ and a chain $\mathcal{C}$ of nested, tight subsets of the extended set of sources S'^+ by the algorithm that starts with chain $\mathcal{C} = \{\emptyset, S'^+, S'^+ \cup \{t\}\}$.

If we modify the network using the algorithm given by Hoppe and Tardos, it holds that the minimum time horizon in which the supplies can be fulfilled does not change. It further holds that the earliest arrival pattern stays the same in the case of earliest arrival transshipments.

Lemma 3.15 (Theorem 4.7 in [69]). The earliest arrival patterns obtained by feasible earliest arrival transshipments in networks $\mathcal{N}$ and $\tilde{\mathcal{N}}$ are the same.

Proof. Observe that the network modifications described by Hoppe and Tardos [40] do not change the total supply. Since the new set of sources S'^+ is still tight, i.e., $v(S'^+) = o^{\theta^*}(S'^+)$, the maximum amount that can be sent into sink t by time θ^* still equals the maximum amount of flow that can be sent into the sink by time θ^* out of original sources in S^+, i.e., $o^{\theta^*}(S'^+) = o^{\theta^*}(S^+)$. By Corollary 3.5, the earliest arrival pattern of network $\tilde{\mathcal{N}}$ is given by $p'(\theta) = o^{\theta}(S'^+)$ for $\theta \leq \theta^*$. Note that $\tilde{\mathcal{N}}$ contains $\mathcal{N}$ as a subnetwork and only arcs (s', s) were added for $s' \in S'^+$ to original sources $s \in S^+$ to obtain $\tilde{\mathcal{N}}$. Therefore, $o^{\theta}(S'^+) \leq o^{\theta}(S^+)$ for $0 \leq \theta \leq \theta^*$. Since $\tilde{\mathcal{N}}$ contains uncapacitated, zero transit time arcs (s', s) for all sources $s \in S^+$ in $\mathcal{N}$, it also holds that $o^{\theta}(S'^+) \geq o^{\theta}(S^+)$, $0 \leq \theta \leq \theta^*$. Thus $p(\theta) = o^{\theta}(S^+) = o^{\theta}(S'^+) = p'(\theta)$ for $\theta \leq x\theta^*$, i. e., the earliest arrival patterns defined by feasible transshipments in both networks are the same. □

It remains to show how to determine the earliest arrival transshipment in $\tilde{\mathcal{N}}$ which can easily be reduced to a feasible earliest arrival transshipment in $\mathcal{N}$.

For the computation we need the lexicographically earliest arrival flow algorithm which is an adaption of the lexicographically maximum flow over time algorithm as described in [40] and shortly resumed in Section 2.2.1. The main changes to the algorithm for lexicographically maximum flows over time are stated in the following.

We use network $\tilde{\mathcal{N}}$ together with chain $\mathcal{C}$. Given the chain $\mathcal{C}$, we can determine an order of the sources and the sink of $\tilde{\mathcal{N}}$ by relabeling the sources in the sense that set S_i contains sources $s_0, \dots, s_i$ for all $i = 0, \dots, \ell$. Observe that the single sink t equals the last node s_ℓ in this ordering. We add a node ψ to $\tilde{\mathcal{N}}$ with uncapacitated, zero transit time arcs (ψ, s_i) for $i = 0, \dots, \ell - 1$ and an uncapacitated arc (s_ℓ, ψ) with transit time $-\theta^*$. Let $\tilde{\mathcal{N}}_\ell$ denote the resulting network, $f^{\ell+1}$ the zero flow. By $\tilde{\mathcal{N}}_i^{f^{i+1}}$ we denote the residual network of network $\tilde{\mathcal{N}}_i$ considering flow f^{i+1}. Since $f^{\ell+1}$ is the zero flow, $\tilde{\mathcal{N}}_\ell^{f^{\ell+1}} = \tilde{\mathcal{N}}_\ell$. In a first step, we compute an earliest arrival ψ-t-flow with time horizon θ^*. in $\tilde{\mathcal{N}}_\ell^{f^{\ell+1}}$ using the successive shortest path algorithm. In contrast to the computation of a static min-cost circulation, this approach guarantees the earliest arrival pattern at node t. Next, we extend this flow to a circulation g^ℓ in $\tilde{\mathcal{N}}_\ell^{f^{\ell+1}}$ by forwarding all flow reaching node t over arc (t, ψ). We set $f^\ell = g^\ell$. During the next ℓ iterations the algorithm considers the sources $s_{\ell-1}, s_{\ell-2}, \dots, s_0$ in descending order. In each iteration $i = \ell-1, \dots, 0$, arc (ψ, s_i) gets deleted from network $\tilde{\mathcal{N}}_{i+1}^{f^{i+1}}$ to obtain network $\tilde{\mathcal{N}}_i^{f^{i+1}}$. Then we compute a maximum static ψ-s_i-flow g^i having minimum cost in $\tilde{\mathcal{N}}_i^{f^{i+1}}$. We construct the flow f^i by adding the new flow g^i to f^{i+1} at the end of each iteration.

To construct the lexicographically earliest arrival flow, we use the natural given generalized path decomposition $\mathcal{P}$ with flow values $x(P)$ for $P \in \mathcal{P}$ containing the paths found during the earliest arrival ψ-t-flow computation and the paths found computing maximum static ψ-s_i-flows, $i \in \{0, \dots, l-1\}$. Sending flow of rate $x(P)$ in a temporally repeated manner over paths $P \in \mathcal{P}$, we obtain the lexicographically earliest arrival flow.

Notice that the earliest arrival pattern obtained by the earliest arrival ψ-s_ℓ-flow computed in the first step does not change by the flow sent in later iterations. Each path from ψ to s_i using backward arc (ψ, t) only decreases the flow arriving at node t at time θ^*. Therefore, such a flow does not change the arrival pattern during the relevant time interval $[0, \theta^*)$. Other paths in $\mathcal{P}$ via node t do not change the arrival pattern as well. They only change the source from which flow is sent into the sink. The lexicographically earliest arrival flow in $\tilde{\mathcal{N}}$ naturally induces a feasible flow in network $\mathcal{N}$ having the same pattern. Thus, we can conclude the following.

Theorem 3.16 (Theorem 4.6. in [69]). Suppose $\mathcal{C}$ is a chain of nested tight subsets of sources and the single sink in the earliest arrival transshipment problem such that the last element of the ordering implied by the chain is the sink. Then the earliest arrival transshipment problem can be solved by a single lexicographically earliest arrival flow computation.

Reconsidering the algorithm for finding a lexicographically earliest arrival flow, we can state the following.

Theorem 3.17. A lexicographically earliest arrival flow with one sink and ℓ sources can be computed via ℓ static min-cost (maximum) s-t-flow computations and one earliest arrival s-t-flow computation.

Since the minimum time horizon needed to fulfill all supplies can be found in strongly polynomial time, the above lemma gives that the overall running time is determined by the running time of the earliest arrival s-t-flow algorithm.

3.7 PRACTICAL RESULTS

Airplane evacuation has become an interesting subject through media since Lufthansa made the lawfully prescribed evacuation test of the new A380 with 853 seats for passengers and 20 members of the crew in March 2006 in Hamburg.

The legal restriction for airplane evacuation are very strict and the position of the seats, position of emergency exits, the width of the aisles and so on have a huge impact on the evacuation time as well as the training of the crew members. The Federal Aviation Administration (FAA) determines rules for the prescribed evacuation test and checks the adherence of these rules during the test. So, for example, each airplane has to be totally evacuated within 90 seconds. During the evacuation, only half of the exits (including emergency exits) are allowed to work. Further, no light is allowed within the airplane and in the outside. Carry-on baggage and pillows have to be distributed in aisles and in emergency exit access ways to complicate the evacuation process. The gender and age specific composition of passengers is prescribed as well as the workout each participant of an evacuation test gets.

Because of the importance of evacuation in general and the nice topology of airplanes, we would like to evaluate the outcome of the earliest arrival transshipment algorithm in airplane networks for several evacuation settings.

For computing an earliest arrival transshipment in a network, we have described several possibilities. The polynomial time algorithm described in this chapter needs to minimize submodular functions. Although, there are

fully combinatorial, strongly polynomial time algorithms for minimizing submodular functions, their implementation would be intractable. Therefore, we implemented an earliest arrival transshipment algorithm which works in the time-expanded network.

A run of the earliest arrival transshipment algorithm in a simple network that corresponds to an airplane instance is not able to take all the above described properties into account. Moreover, the times to go from one seat to the next seat and to walk along the aisle are only coarse estimates. However, we are able to compute lower bounds on the evacuation time by assuming the model to be a perfect evacuation setting. Every real world evacuation will need more time because of the before mentioned reasons. The determined lower bound gives an important impression of whether it is possible to fulfill regulations prescribed by law.

We use LEDA[3] version 5.0.1. as network generator. Further, we use the internal graph data structure from LEDA and several related data structures such as nodes and edges. Shortest path and min-cost flow algorithms already implemented in the LEDA library are used as subroutines. All networks have only integral data such as transit times, supplies, demands, and capacities.

The earliest arrival transshipment algorithm is implemented in C++ and compiled with g++ version 3.4.5 using the -O3 option. All tests were conducted on a 2.6 GHz AMD Opteron™ 252 Processor having 8 GByte memory.

To visualize the flow over time we use the graphical user interface called *Aninet*[4] implemented by Alexander Hall.

Time-Expansion of Airplane Networks. In an airplane network, the sources model the seats of the airplane and each source has a supply of one. The exits of the airplanes are modeled as nodes that are connected to a single sink. This construction is used, since we do not know the exact amount of flow that leaves an exit in advance. Therefore, all flow needs to reach the single sink, which demand is set to the negative amount of number of seats in the network. In the aisle, we place nodes at each position where passengers can step from their seats into the aisle. Nodes that are adjacent in the sense that a passenger could reach the node from another node by one step are connected by two reverse directed arcs with the same transit time and a capacity of one. Also the nodes modeling the seats in between the aisles are connected by two reverse arcs. Only the nodes modeling the seats in the rows next to the window are connected by a directed arc towards the aisle, since no reasonable person will use a backward arc here. We assign integral

[3]http://www.algorithmic-solutions.de/enleda.htm

[4]Contact Alexander Hall: alex.hall@inf.ethz.ch.

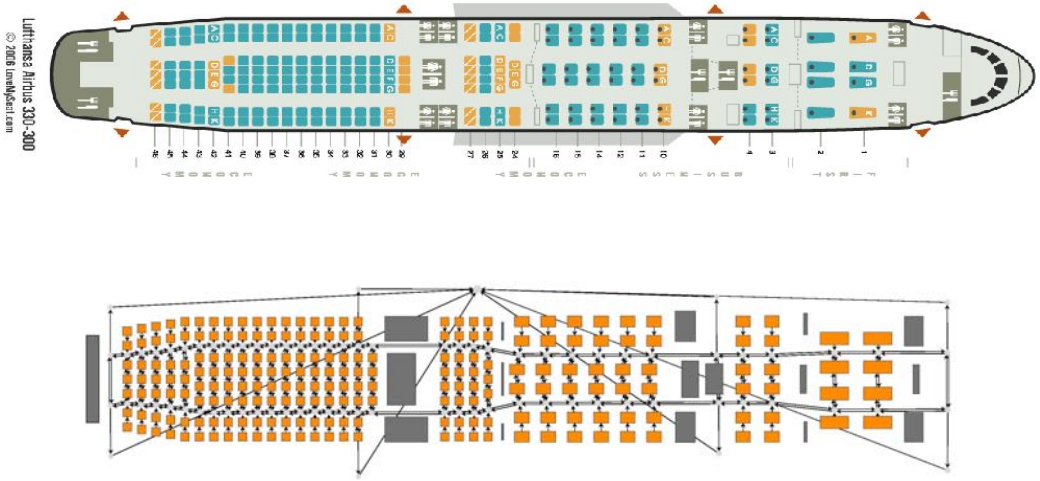


Figure 3.12: A real seat map of an Airbus A330 of Lufthansa found at http://www.lovemyseat.com/airlines/Lufthansa/Airbus_A330/Airbus_A330.html (©Consumer Travel Research Systems, Inc.)(top) and the corresponding LEDA graph with directed arcs (bottom).

data to arcs, such as transit times and capacities. In Figure 3.12, we show a real airplane seat map in the upper part and the corresponding network generated using the LEDA graph editor.

Instead of simply constructing the time-expanded network as described in Section 1.2.3, we need to take some special properties of airplanes into consideration. Although, the network contains two reversal arcs between two nodes with equal transit time and equal capacity, we do not allow to use both arcs at the same time at their full capacity. The total flow on both arcs should not exceed the capacity of one of the arcs, since the aisles in an airplane are very narrow and it is difficult to move. Moreover, we do not want to allow flow units to overtake other flow units in nodes. In particular, people sitting at a window seat are not allowed to enter the aisle before the person sitting at the aisle seat has left its seat. We further forbid overtaking in an aisle which is motivated by the narrowness in an airplane. To achieve these requirements, we need a special construction of the time-expanded network. First, we assume that each arc in a pair of reverse arcs has transit time of one. This can be obtained by splitting pairs of reverse arcs having larger transit time into a sequence of pairs of reverse arcs having transit time 1. For each node v in the network, we insert additional nodes v' into the network and connect v' and v by zero transit time arcs $(v'(\theta), v(\theta))$ for each time layer θ. If v is a source, we set the capacity of $(v'(\theta), v(\theta))$ to 1 and if v is a node in the aisle, we set the capacity to the width of the

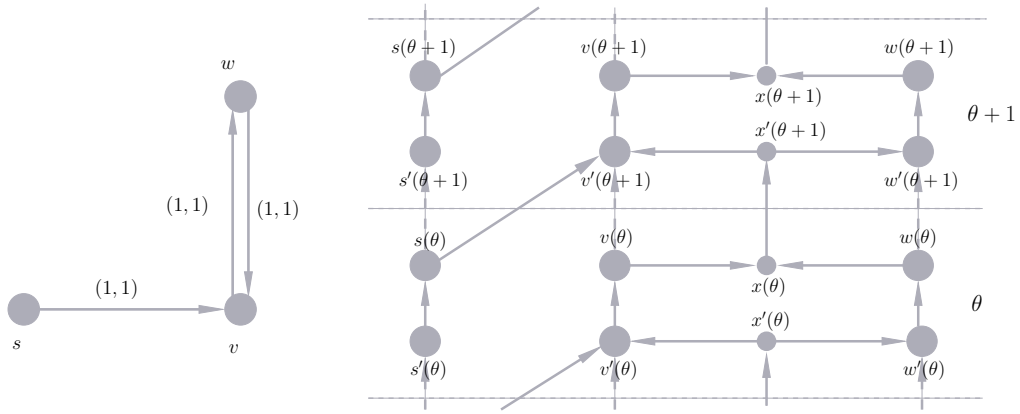


Figure 3.13: On the left hand side we have drawn a section of the airplane network modeling one seat node in the window row and two nodes in the aisle. On the right hand side, we show two time layers of the time-expanded network using the special arc construction which restricts the amount of flow on the reverse arcs to the capacity of one of those arcs and guarantees that overtaking is forbidden.

aisle, i.e., the capacity of an outgoing arc to another node in the aisle. The capacity of the arcs restricts the number of people at such a location at the same time. These arcs guarantee, that there is no overtaking in the rows with seats and no more people are located at the same place as there is capacity available. The uncapacitated arcs $(v(\theta), v'(\theta+1))$ and $(w(\theta), w'(\theta+1))$ act as holdover arcs. For each pair of reverse arcs between two nodes, say v and w, we insert two additional nodes x, x' and uncapacitated, zero transit time arcs $(x'(\theta), v'(\theta)), (x'(\theta), w'(\theta)), (v(\theta), x(\theta)), (w(\theta), x(\theta))$ into each time layer θ. By arcs $(x(\theta), x'(\theta+1))$ having capacity $u(v,w)(= u(w,v))$, we implicitly connect nodes v and w in subsequent time layers. These arcs guarantee that the total flow from v to w and from w to v at the same time is bounded by the capacity of one of the original arcs. The above described arc construction is depicted in Figure 3.13. For each pair of nodes connected by a single arc (a,b) in the original network, we only insert arcs $(a(\theta), b'(\theta+\tau(a,b))$ for $\theta = 0, \ldots, T-\tau(a,b)-1$ and $\tau(a,b)$ the transit time.

The Algorithm. Using the modified time-expanded network as described in the previous paragraph, the algorithm for finding an earliest arrival transshipment works as described in Section 2.3.2. First, a supersource s connected by zero transit time arcs to the artificial copies of sources at time layer zero is inserted, i.e., arcs $(s, v'(0))$ for $v \in S^+$. The capacity of those arcs is set to the supply of the corresponding source which is in our case one. Moreover, a supersink t is inserted which is connected from all artificial sink nodes $s'(\theta)$,

for s the single sink in all time layers $\theta \in \{0, \ldots, T-1\}$ to the supersink by uncapacitated, zero transit time arcs. Here, T is the value the guessed time horizon.

In this time-expanded network, we compute a static min-cost (maximum) s-t-flow where transit times are interpreted as costs. The LEDA Graph Library offers this algorithm. The outcome of this algorithm is a set of s-t-paths and corresponding flow values for each path. Flow sent over these paths obeys the earliest arrival property because of the special structure of the network. If the total flow value is strictly smaller than the total supply of the earliest arrival transshipment problem, then the chosen time horizon is too small and the run of the algorithm has to be done again with a higher value for the time-horizon. Otherwise, we can determine the minimum needed time horizon by the length of the longest path in the path decomposition plus one additional time unit. The additional time unit is needed since we consider the continuous flow model.

The flow over time can then be computed from the static flow in the time-expanded network by computing a (non-generalized) path decomposition of the flow on non-backward arcs. A path that uses a holdover arc in the time-expanded network can be interpreted in the flow over time setting as waiting in the node for the time interval corresponding to the time layers connected by the holdover arc.

The Tests. For the testing, several airplane seat maps are represented as networks. The LEDA GraphWindow was used to create the networks. The seat maps can be found at http://www.lovemyseat.com. In particular, we consider the following airplanes: Bombardier Q 400, Airbus A330, Boeing B747, and Airbus A380. For all those instances, we made coarse assumptions on the transit times for simplicity. The time that people need for one step from a seat to the next one and for one step in the aisle each is supposed to be one second. For each airplane we create several instances by closing certain exits. Table 3.1 shows the sizes of the instances, including number of nodes and arcs of the network, number of seats, and the number of open exits. Further, the total evacuation time, which is the optimal time horizon in an earliest arrival transshipment computation is given.

Considering Table 3.1, we can observe that in most cases it is possible to evacuate the considered airplanes in less than 90 seconds. Thus, the regulation by law can be satisfied in most cases, even if all exits on the right side of the airplane (lower part in Figure 3.12) are closed. An interesting observation is that small airplanes cannot necessarily be evacuated within a small time horizon. Since small airplanes have few emergency exits, people

Network	#nodes	#arcs	#seats	#exits	time horizon
BombardierQ400	113	189	74	4	40
A330	340	602	222	8	38
A330RightSide	340	598	222	4	60
A330BackSide	340	589	222	4	59
A330FrontSide	340	589	222	4	106
B747	689	1367	427	12	49
B747StairsClosed	687	1361	427	12	49
B747BottomFloor	689	1365	427	10	49
B747TopFloor	689	1357	427	2	419
B747RightSide	689	1361	427	6	77
A380	965	1837	573	16	50
A380RightSide	965	1829	573	8	78

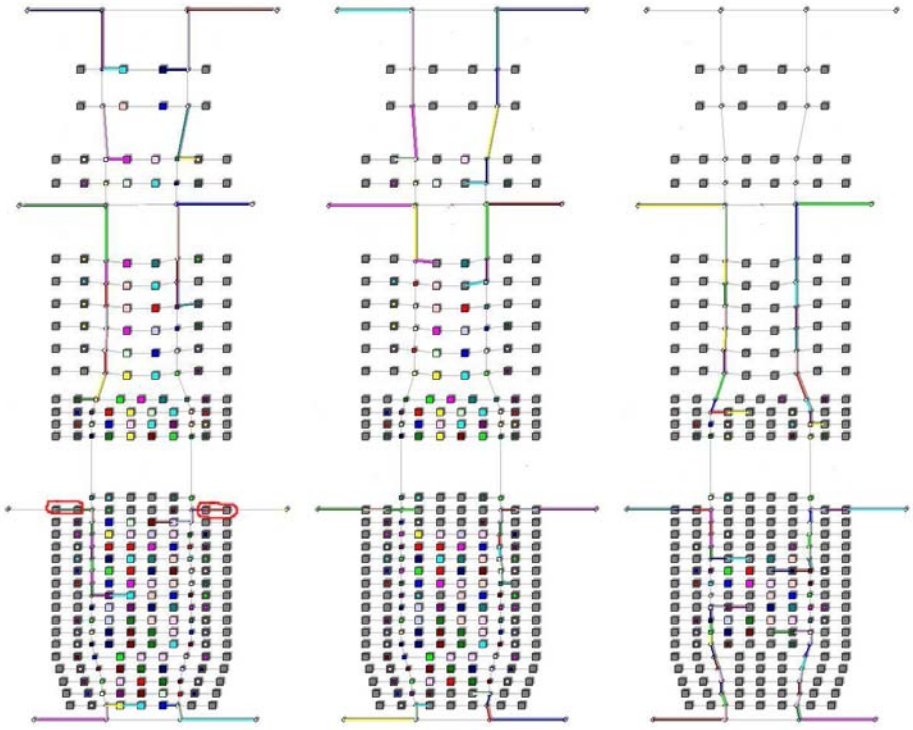
Table 3.1: This table contains the airplane network instances with their size and the optimal time horizon. The extension of the airplane name denotes the sides at which the exits are open. On the reverse side we have closed the exits. As back side in the airplane we denote the part of the airplane containing the economy and tourist class (left part of the airplane in Figure 3.12. The front side is the the part of the airplane containing business and first class (right part of airplane in Figure 3.12). The right side of the airplane contains all exits on the right when standing in the back of the airplane and looking towards first class.

need more time to leave the small Q400 than the larger Airbus A330.

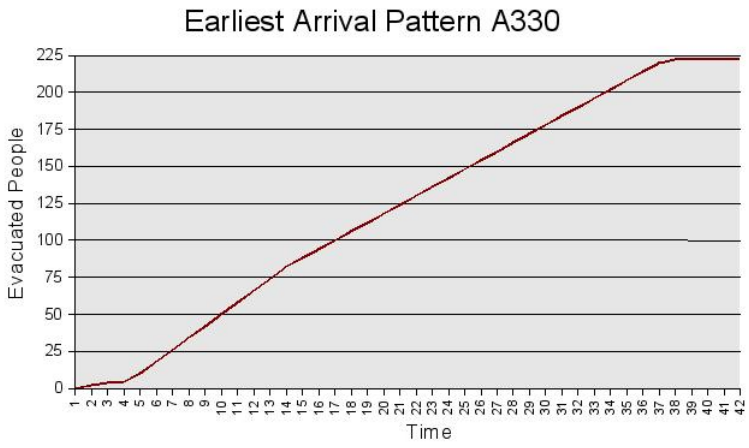
The airplane Airbus A330 can be totally evacuated within 38 seconds. Snap shots of the flow at different points in time are depicted in Figure 3.14.

A subset of sources is marked in the left-most picture of Figure 3.14 for which the flow already reached the sink and no more flow can leave these sources. This set equals set $S^+ \setminus S_1$ from Algorithm 1 and time 3 is set to θ_1. At this point in time, the seats next to the second exits from the bottom are emptied and the flow has already reached the sink. In the earliest arrival pattern, this point in time is marked by a breakpoint at time three at which the slope of the function decreases. The next breakpoint, where the slope of the pattern decreases, can be seen at time 14 which is the latest point at which flow leaves the network via the up-most exits in the first class. All the snap shots from the flow over time indicate, that the total evacuation time is strongly dependent on the bottlenecks which are the exit areas.

Another interesting observation is, that, if we close all exits on the right side of the airplane, the total evacuation time is not twice as long as the evacuation time of the airplane when all exits work. For the A330, the total evacuation needs 60 seconds which is surprisingly short compared to 38 seconds when all eight exits work. This is because the long way through the first class, which was not used during later times of the evacuation in the setting where all eight exits are opened, now plays an important role in order to evacuate as many people as early as possible. For an impression on the



(a) The left-most picture presents the situation after three seconds. The picture in the middle is made at time 6. The picture on the right shows the evacuation situation after 18 seconds.



(b) Earliest arrival pattern of the flow.

Figure 3.14: Different times of the evacuation of the A330 and the earliest arrival pattern produced by this flow. After 38 seconds the airplane is totally evacuated.

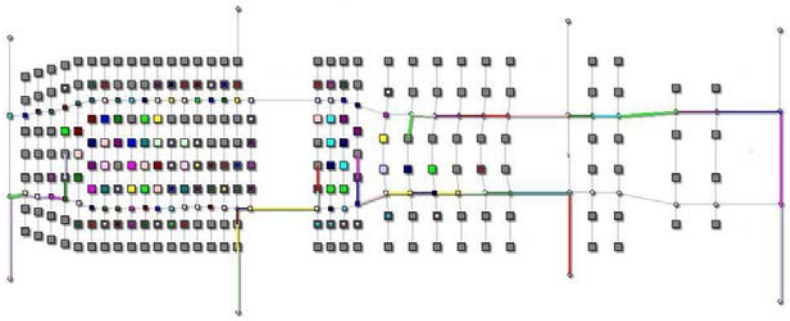


Figure 3.15: We consider the evacuation of the airplane A330 when the exits on the left side are kept closed during the evacuation. In the upper picture, the situation in the airplane at time 18 is depicted. Also the exit in the first class is used till the end. The total evacuation time for this setting is 59 seconds.

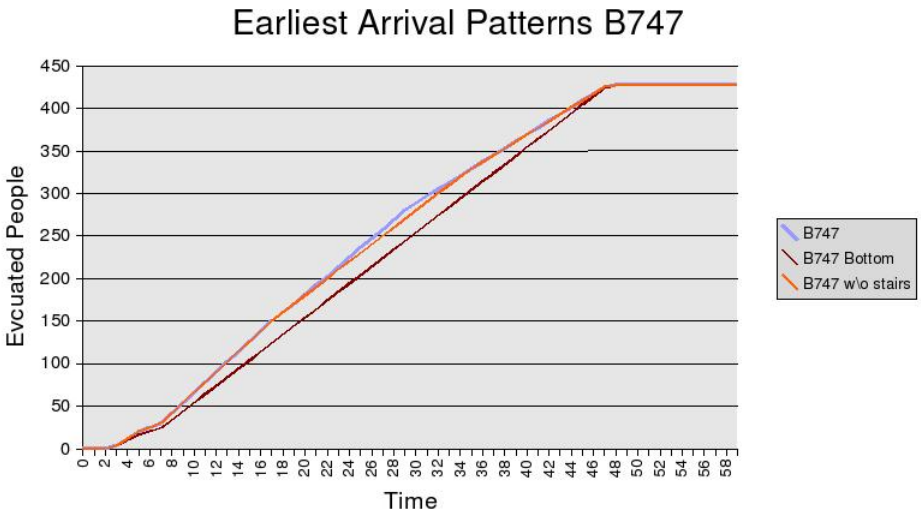


Figure 3.16: Earliest arrival patterns of different instances of the B747 airplane all having the same evacuation time.

flow see Figure 3.15. Since flow leaves the airplane continuously via all four open exits, the pattern is a linear function for times later than 3. For time 3, there is the breakpoint in the slope of the function as in the instance for all exits open.

Considering the B7474 instances, we can observe in Table 3.1 that the total evacuation times are the same for the instances with all exits open, with stairs closed and with all exits closed at the top floor. This is somehow surprising. Figure 3.16 shows the earliest arrival patterns that occur at the single sink, i. e., the cumulated amount of flow leaving all exits up to each time $\theta \geq 0$. Again, we can observe the decrease of the slope at time 3 in all

instances. At time 3 only flow units sent out of the sources directly next to the emergency exits reach the sink. Remaining flow units need another time unit to reach some of the exits.

The worst instance, considering the arrival pattern, is the bottom floor instance of the B747. In this instance, the slope of the patterns increases linearly up to the end of the evacuation. The instances B747StairsClosed and B747 result in a better earliest arrival pattern. Considering for example time 24, more than 200 people have already left the airplane in those settings, whereas only 190 people have left the airplane, if the exits on the upper floor are kept close. This behavior occurs, since all people from the upper floor need to wait till people from the bottom floor have left the airplane. The pattern, in which also the stairs are allowed to use, is slightly better than the one with closed stairs. In this case, people from the bottom floor can leave the airplane by climbing up to the upper floor. There, only few people can use a comparably large number of exits. The more flow can be sent out of those exits, the higher is the increase in the slope of the pattern. The slope of the latter two patterns decrease when the last flow out of the upper floor has left the network.

The A380, for which we used a seat map for 573 passengers, can be optimally evacuated in 78 seconds using the coarse assumptions on the transit time within the airplane. In the real world evacuation experiment for this airplane in March 2006, even 853 people have left the airplane within approximately 80 seconds. In contrast to our optimal and perfect setting, such a real-life experiment cannot be optimal and not at all perfect since there are people of different speed, luggage in the aisles and so on. Thus, we can conclude that the transit times in our airplane network instances are chosen too large. Even then the computed optimal evacuation times are surprisingly small.

In Table 3.2, we present the size of the time-expanded network for the optimal time horizon for the given instances and the times needed for the computation of an earliest arrival transshipment using these time-expanded networks. We distinguish three different computation times: the time to construct the time-expanded network, the time to compute the static min-cost s-t-flow in the time-expanded network, and the time needed to compute a path decomposition. Obviously, the time needed increases with larger networks and a larger number of sources. In particular, the running time for computing the path decomposition is strongly dependent on the lengths of the computed paths. Although the number of paths are the same, only instances with open exits in the front of the airplane need significantly longer time for the path decomposition. In those instances, the paths found are significantly longer than the paths in other instances.

Network	#nodes	#edges	construction	flow comp.	path dec.
BombardierQ400	14482	24372	<1s	1s	<1s
A330	44082	74682	<1s	1s	3s
A330RightSide	69602	118508	<1s	2s	5s
A330BackSide	68442	116509	1s	2s	5s
A330FrontSide	122962	210462	0s	3s	6s
B747	120444	210685	<1s	3s	11s
B747StairsClosed	119954	209771	<1s	4s	11s
B747BottomFloor	120444	210587	1s	4s	12s
B747TopFloor	1029904	1821915	4s	35s	98s
B747RightSide	189268	332471	1s	7s	22s
A380	164302	285831	1s	5s	28s
A380RightSide	256310	447663	1s	11s	33s

Table 3.2: Listing of the sizes of the time-expanded networks for the instances given in Table 3.1 having the optimal time horizon and the times needed to compute the earliest arrival transshipment in those networks, distinguished into construction of the time-expanded network, the flow computation in the time-expanded network, and computation of the path decomposition in the time-expanded network.

Conclusion. We are able to compute the earliest arrival transshipment in airplane networks. By using the earliest arrival transshipment problem, we maximize the number of people leaving the airplane at every point in time. As the visualization implies, we do not have a strict influence on the behavior of flow in the interior of the airplane. This does not change our goal of optimizing the arrival pattern, but it is a goal to forbid unnecessary movement within the airplane. However, this does not model reality in a concise way, but we hope that this helps to find good heuristics that do not need the time-expansion. Thereby, we would save memory which causes computational problems, especially for huge instances as the A380 when it needs a very large time horizon.

Using this kind of time-expansion, not only airplane settings can be modeled in a quite realistic way but also the evacuation of buildings like office buildings and apartment buildings.

Chapter 4

Earliest Arrival s-t-Flows with Flow-Dependent Transit Times

4.1 Introduction

In the following we will consider the earliest arrival s-t-flow problem in networks where each arc is given a transit time function. The earliest arrival s-t-flow problem describe an evacuation setting, where an unknown and therefore maximum amount of people need to be evacuated from a single site at each point in time. For buildings or airplanes this seems to be unrealistic. However, this restriction is acceptable when considering flow-dependent transit times which already complicate the problem significantly.

Different models of flow-dependent transit times are studied, as for example the one by Merchant and Nemhauser [63] or the graph expansion of Carey and Subrahmanian [11]. For the models considering inflow-dependent transit times introduced by Köhler, Langkau, and Skutella [54] and load-dependent transit times introduced by Köhler and Skutella [55], already algorithms for flow over time problems are given. While flow over time problems such as the quickest transshipment problem and the maximum s-t-flow over time problem are studied in the context of flow-dependent transit times, there seem to be no results for the related problem of earliest arrival s-t-flows.

In Section 4.2, it is shown that earliest arrival s-t-flows do not exist in general for the case of inflow-dependent and load-dependent transit times. In Section 4.3 a relaxed version of the earliest arrival s-t-flow problem (the α-earliest arrival s-t-flow problem) is defined and studied. The objective here is to find an s-t-flow over time that needs only α-times longer to send the maximum amount of flow into the sink up to each time $\theta \in [0, T)$. In particular, both lower and upper bounds for the value α are shown. Section 4.4 presents an approximation algorithm for the α-earliest arrival s-t-flow problem and an idea about the practicability of this approximation for this problem is given.

The results of this chapter are based on joint work with Ekkehard Köhler. An extended abstract of the results already appeared in [5] and a full version will appear in [4].

4.2 NON-EXISTENCE OF EARLIEST ARRIVAL s-t-FLOWS

Earliest arrival s-t-flows are those maximum s-t-flows over time that send for each time $\theta \in [0, T)$ the maximum amount of flow from s to t. While the existence of a maximum s-t-flow over time for a fixed time horizon T is obvious, it is not that clear that there are always flows over time that are not only optimal for T but also for each $\theta \in [0, T)$. Gale [28] showed the existence of earliest arrival s-t-flows for general networks with constant transit times on the arcs and, more generally, for networks with time-dependent (but not flow-dependent) transit times and capacities on the arcs. He made use of the fact that one can model flows over time with constant or time-dependent transit times using static flows in the time-expanded network. There the standard max-flow min-cut theorem yields the result.

For the case of flow-dependent transit times, there exists for any fixed time horizon T an s-t-flow over time that sends the maximum amount of flow from s to t. It is quite natural to ask whether there is again such a maximum s-t-flow over time that is maximal also for each $\theta \in [0, T)$. Unfortunately, there is no simple time-expanded model for flows over time with flow-dependent transit times that allows to reduce the flow over time problem to a static one. In fact, as will be shown by the following simple counterexamples, there cannot be a similarly nice existence result for earliest arrival s-t-flows for the case of flow-dependent transit times as the one by Gale.

Consider the one-arc network, shown in Figure 4.1, together with the simple linear transit time function given next to it and a capacity two. We consider a flow over time model with inflow-dependent transit times. Let $T = 3$ be the considered time horizon. When sending flow from s to t at a flow rate of 2 in time interval $[0, 1)$ and at flow rate linearly decreasing from 2 to 0 in time interval $[1, 3)$, then by time 3 a flow of 4 units has reached the sink t. In fact, this is the maximum amount of flow that can be sent from s to t in this time horizon.

To construct an earliest arrival s-t-flow, we have to make sure that the maximum possible amount of flow has reached the sink for any $\theta \in [0, T)$. To show that this is not possible for this example, we examine just two values of θ. Sending flow at a flow rate linearly decreasing from 2 to 0 in time interval $[0, 2)$ shows that an earliest arrival s-t-flow must send at least 2 units of flow to t up to time $\theta = 2$. In fact, sending any of the flow in this time interval at a higher flow rate would result in a decrease of the flow value reaching t up to time $\theta = 2$. It follows easily that any flow over time sending the maximum amount of flow up to $\theta = 2$ into t cannot send more than 2.5 units of flow into t up to $\theta = 3$. Since, however, the value of a maximum s-t-flow over time for time horizon 3 is 4, this implies the following theorem.

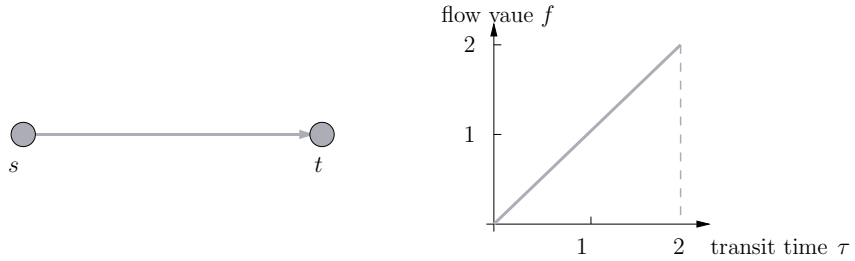


Figure 4.1: Network with inflow-dependent transit time given for arc (s, t).

Theorem 4.1. For inflow-dependent transit times, earliest arrival s-t-flows do not exist in general.

Similarly as for inflow-dependent transit times, one can show that also load-dependent transit times do not allow for earliest arrival s-t-flows. For better readability we give the proof for this result, using as transit time function a simple left-continuous step function (i.e., a piecewise constant left-continuous function). A more technical argument helps to show that also continuous load-dependent transit time functions do not admit earliest arrival s-t-flows.

Consider again a graph consisting of just one edge (s, t) with (inflow) capacity one. The load-dependent transit time function τ for this arc is given as follows. At a load value in $[0, 1]$ a unit of flow needs 3 time units to travel from s to t, for a load value in $(1, 5]$ a flow unit needs 6 time units for crossing the arc, and for any higher load value the transit time is ∞.

Now consider the maximum s-t-flow over time that can be sent from s to t for the two time horizons $T_1 = 4$ and $T_2 = 9$.

- For time horizon T_1, at most 1 flow unit can reach t up to time T_1. This can only be achieved, if flow is sent during time interval $[0, 1]$ with rate 1 and then no further flow is sent into the arc before time 3 anymore.
- For time horizon T_2, at most 3 flow units reach t up to T_2, if flow is sent constantly at rate 1 up to time 5.

Using these observations, we can show the following lemma.

Lemma 4.2. If flow is sent at rate 1 during time interval $[0, 1]$ and then no more flow is sent during time interval $(1, 3]$, then at time 9 at most 2 flow units can reach the sink.

Proof. After sending the first unit of flow into the arc, we can start sending flow only after time 3, otherwise the flow is not optimal at time T_1. If we

start sending flow at rate 1 right after time 3, then up to time 4 only a load of 1 is on the arc. This second flow unit cannot reach the sink completely before time 7.

Any flow put on the arc after time 6 cannot reach the sink before time 9. So it remains to show that a flow unit that is put on the arc during interval $(4, 6]$ cannot reach the sink before time 9 either.

Observe, that any flow that is sent on the arc during interval $(4, 6]$ will increase the flow over load 1, since the second unit of flow starts leaving the arc not before time 6. If we start sending the third unit onto the arc at time 4, then the second unit of flow will completely reach the sink not before time 10. In fact, to make sure that the second unit reaches t up to time 9, we should not start sending the third unit of flow before time 5. If we start the third unit at time 5, then the second unit leaves the arc during time 7 to 9. Hence, at least between time 5 and 7, the arc has a load greater than 1, implying that the third flow unit has transit time 6 during this time interval. As a consequence, it can travel at most one third of the arc up to time 7 and thus will not be able to reach the sink before time 9. □

Hence, we have shown that in this simple network with load-dependent transit times there exists no flow over time that sends both the maximum amount of flow up to time 4 and up to time 9 into the sink. As a consequence we get:

Theorem 4.3. For load-dependent transit times, earliest arrival s-t-flows do not exist in general.

4.3 α-EARLIEST ARRIVAL s-t-FLOWS

As shown above, in the case of flow-dependent transit times, there are instances where there exists no earliest arrival s-t-flow. Because of that, we are interested in related optimization problems that determine "almost earliest arrival s-t-flows". As for the case of constant transit times, there are two possible ways. One option is to consider the related optimization problem that relaxes the amount of flow sent into the sink up to each $\theta \in [0, T)$ as it is done in the easier case of constant transit times by Hoppe and Tardos [39]. We follow the different track by relaxing the time component. Instead of relaxing the amount of flow we rather relax the time up to when a certain amount of flow has to reach the sink. More precisely, we introduce the problem of finding an *α-earliest arrival s-t-flow* to minimize the lateness of flows. Note that the basic idea for this kind of relaxation is somewhat similar to what has been done for optimizing single-source-single-sink quickest transshipments for

various kinds of flow over time models (see [55, 54, 22]). In what follows, we show that there is a constant approximation of this new problem, i.e., the optimal value of α can be bounded from above by a constant. Moreover, we also estimate lower bounds.

In the sequel, we will consider the following optimization problem: Find the minimum α such that there is a flow over time f that sends for each $\theta \in [0, T)$ at least as much flow into the sink t as can be sent into t up to time $\frac{\theta}{\alpha}$ by a maximum s-t-flow over time within time horizon $\frac{\theta}{\alpha}$, i.e.,

$$\text{value}(f, \theta) \geq o^{\frac{\theta}{\alpha}}(\{s\}) \ .$$

Here, we denote again by $o^{\frac{\theta}{\alpha}}(\{s\})$ the maximum amount of flow that can be sent into sink t by time $\frac{\theta}{\alpha}$ and by $\text{value}(f, \theta)$ the amount of the flow f that has entered the sink till a specified time θ where θ is not necessarily the time horizon. A flow over time fulfilling the above requirement will be called an *α-earliest arrival s-t-flow*. We require that there is no flow in the network before time 0 or after time T.

4.3.1 Upper Bound

In this section, we show that there is always a 4-earliest arrival s-t-flow. More precisely, we prove that there is always a flow over time that sends up to every time $\theta \in [0, T)$ at least as much flow into the sink as a maximum s-t-flow over time with flow-dependent transit times sends within a time horizon of $\frac{\theta}{4}$. For showing this result we make use of a construction called *interval stacking*.

Consider a time horizon $[0, T)$. A 2-*interval stacking* of this time horizon is given by a logarithmic subdivision of this interval into a sequence of subintervals. The first $\lfloor \log(T+1) \rfloor - 1$ intervals are defined such that the i^{th} interval I_i is given by $I_i = [(2^i - 1), (2^{i+1} - 1))$ for $i \in \{0, 1, \ldots, \lfloor \log(T+1) \rfloor - 1\}$. Since for the remaining part of the time horizon this scheme cannot be continued, it is filled up with smaller intervals of size 2^i, for $i \in \{0, 1, \ldots, \lfloor \log(T+1) \rfloor - 1\}$, where subsequently each of these intervals is chosen as large as possible. An example is given in Figure 4.2.

Using this 2-interval stacking, we now construct a flow over time f_{IS} as follows. For every time horizon 2^i $(i = 0, 1, \ldots, \lfloor \log(T+1) \rfloor - 1)$ a maximum s-t-flow over time f_i is determined. Then the flow f_i is sent within the corresponding time interval of length 2^i in the interval stacking. The so constructed flow f_{IS} in the interval stacking will now be shown to give a 4-approximation for our problem.

For the beginning we assume that a shortest s-t-path in our network with respect to the zero flow transit times of the arcs has length at least 1. The

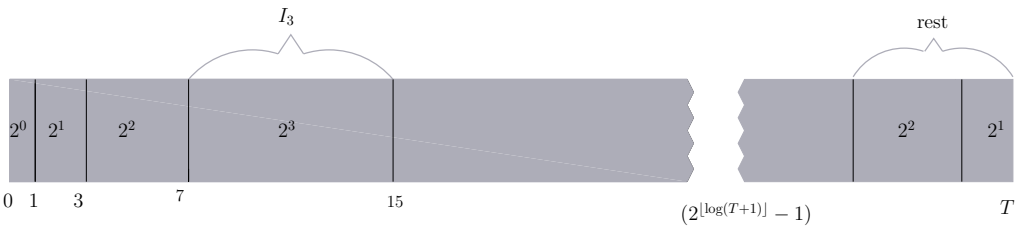


Figure 4.2: A possible 2-interval stacking.

case of shorter s-t-paths will be considered later.

Define for $\theta \in [1, T)$ the value α_θ by $\alpha_\theta = \frac{\theta}{\text{length}(I)}$ where I is the largest interval of the interval stacking lying in front of θ. For $\theta \in [0, 1)$ we define α_θ to be 1, since no flow can reach the sink before time 1 by the above assumption and, thus, there is no 'time delay' of flow before time 1.

For example, the value α_θ for $\theta = 10$ would be $\frac{10}{4}$ (see Figure 4.2). Using this definition we get the following simple lemma. By value(f_{IS}, θ) we will denote the value of the flow f_{IS} compounded for the whole time horizon T up to time θ.

Lemma 4.4. Up to time θ at least $o^{\frac{\theta}{\alpha_\theta}}(\{s\})$ units of flow can reach the sink using flow f_{IS}, i.e., value$(f_{IS}, \theta) \geq o^{\frac{\theta}{\alpha_\theta}}(\{s\})$.

Consequently, α_θ is an upper bound on the value α that we get for the flow f_{IS} at time θ. Thus, if we can bound α_θ for all $\theta \in [0, T)$ by a constant, we have a bound on α. However, within a particular interval I_i, $i \in \{1, \ldots, \lfloor \log(T+1) \rfloor - 1\}$ the ratio $\alpha_\theta = \frac{\theta}{\text{length}(I_{i-1})}$ is maximal for $\theta = (2^{i+1} - 1) - \varepsilon$ and $\varepsilon > 0$ small. Thus, we get:

$$\alpha_\theta \quad = \quad \frac{(2^{i+1} - 1) - \varepsilon}{2^{i-1}} = 4 - \frac{1 + \varepsilon}{2^{i-1}} \quad ,$$

implying that α_θ is bounded by 4 for this case. To prove the same ratio for θ within the remaining part of the interval stacking (i.e., for $\theta \in [2^{\lfloor \log(T+1) \rfloor} - 1, T)$) we simply prolong this last part of the interval stacking to an interval of length $2^{\lfloor \log(T+1) \rfloor}$ and use the same calculation as above. The bounds of this interval are then obviously $[2^{\lfloor \log(T+1) \rfloor} - 1, 2^{\lfloor \log(T+1) \rfloor + 1} - 1)$.

This proves our intended bound of 4 for the case of s-t-paths having a length with respect to the zero flow transit times of at least 1. Similarly, one can prove this bound for the case of positive s-t-path length. More precisely, if the shortest s-t-path has length ℓ with $0 < \ell < 1$, then simply scaling all transit times in the network by $\frac{1}{\ell}$ reduces this case to the previous one.

For the case of $\ell = 0$, this approach is not directly applicable. Note that, depending on the model, an s-t-path length of 0 for a positive amount of flow can imply that an infinite amount of flow can be sent from s to t in time 0. However, when the flow rate entering a particular arc per time unit is bounded by some capacity, this behavior can be excluded. Although we cannot prove a strict bound of 4 as in the previous cases, it is easy to show that for any $\varepsilon > 0$ there is a flow that sends at time $\theta + \varepsilon$ at least as much flow from s to t as is maximal for $\frac{\theta}{4}$. Consequently, we have the following theorem and corollary.

Theorem 4.5. For any $\varepsilon > 0$, there exists an s-t-flow over time f_ε that sends for any $\theta \in [0, T)$ at least as much flow into the sink t up to time $\theta + \varepsilon$ as is maximal for time $\frac{\theta}{4}$.

Corollary 4.6. For s-t-flows over time with flow-dependent transit times and positive free-flow travel time between s and t, there exists always a 4-earliest arrival s-t-flow.

It can be shown that the choice of 2 as the basis of the logarithm in the interval stacking is the best possible. A very similar scheme has been used for an online scheduling problem by Hall, Schulz, Schmoys, and Wine [34].

4.3.2 Lower Bound

As shown above, there exists a constant approximation bound for the α-earliest arrival s-t-flow problem with flow-dependent transit times. For the interval stacking method there are easy examples showing that this approach cannot achieve a better α-value than 4. In order to find lower bounds for the general problem, we analyzed the inflow-dependent and the load-dependent transit time model independently.

Inflow-Dependent Transit Times

Theorem 4.7. In the case of inflow-dependent transit times, there are instances having no α-earliest arrival s-t-flow for $\alpha \leq \frac{3}{2} - \varepsilon$, for all $\varepsilon > 0$.

Proof. For the given $\varepsilon > 0$ choose an integer $k > 1$ such that $\frac{3k-1}{2k} > \frac{3}{2} - \varepsilon$. Now consider a single arc network with arc $a = (s, t)$, capacity $2k$, and the transit time function $\tau(f(a))$, with $\tau(f(a)) = 1$ for $f \leq 1$ and $\tau(f(a)) = 3k-1$ for $f(a) > 1$. For this network, the value of a maximum s-t-flow over time reaching the sink t up to time $\theta = 2k$ has value $2k - 1$ and up to time $\theta = 4k$ has value $(k + 1)2k + (3k - 2) = 2k^2 + 5k - 2$.

Suppose there is an α-earliest arrival s-t-flow f^* with $\alpha = \frac{3k-1}{2k}$. Thus, at time $\theta_1 = 3k-1$, f^* has to send at least as much flow into the sink as is maximal for time $\frac{\theta_1}{\alpha} = 2k$ and, similarly, at time $\theta_2 = 6k-2$, f^* has to send at least as much flow into t as is maximal for time $\frac{\theta_2}{\alpha} = 4k$.

Since f^* has to satisfy $\text{value}(f^*, 3k-1) \geq o^{2k}(\{s\}) = 2k-1$, it has to send at least $2k-1$ units of flow at the lower rate of 1 from s to t during the interval $[0, 3k-1)$. Thus, up to time $3k-1$, the flow f^* can send no more than $(2k-1)\cdot 1 + ((3k-1)-(2k-1))\cdot 2k = 2k-1+2k^2$ units of flow out of the source. On the other hand, f^* cannot send more than $3k-2$ units from s to t during the interval $[3k-1, 6k-2)$. This implies that $\text{value}(f^*, 6k-2) \leq 3k-2+2k-1+2k^2 = 2k^2+5k-3$. This is a contradiction to f^* being an α-earliest arrival s-t-flow, since $o^{4k}(\{s\}) = 2k^2+5k-2 > 2k^2+5k-3$. □

Note that the transit time function in the example can easily be transformed into a continuous function without changing the bound given in Theorem 4.7.

Load-Dependent Transit Times

A similar example as for the inflow-dependent case can also be constructed for the load-dependent model.

Theorem 4.8. For load-dependent transit times, there are instances having no α-earliest arrival s-t-flow for $\alpha \leq \frac{5}{4}$.

Proof. Consider again an one arc network with the following simple load-dependent transit time function on arc $a = (s,t)$. At a small load of y the transit time is set to $\frac{1}{\alpha} = \frac{4}{5}$; for larger load of up to z units ($z >> y$) the transit time is set to $1+\varepsilon$ for some later to be determined $\varepsilon > 0$. For any larger load than z, the transit time is set to infinity.

First, observe that the maximum amount of flow that can reach the sink up to time $\frac{1}{\alpha} = \frac{4}{5}$ is y units. Thus, in an α-earliest arrival s-t-flow, y units of flow have to reach the sink up to time 1. The maximum amount of flow that can reach the sink up to time $1+\varepsilon$ is z. Consequently, an α-earliest arrival s-t-flow has to send at least z units up to time $\alpha(1+\varepsilon)$ into the sink.

If one would start sending flow such that the load on the arc is higher than y right from the beginning, the first units of flow would reach the sink not before $1+\varepsilon$, implying that the required y units of flow do not reach the sink by 1. Therefore, one can assume that the first flow is sent such that the load is not higher than y and only later the higher load option is used. However, in order to send a sufficient amount flow up to time $\alpha(1+\varepsilon)$ into the sink, at least $1+\varepsilon$ time units earlier the higher load option has to be

used. Thus, at most during the first $\alpha(1+\varepsilon)-(1+\varepsilon)$ time units, the arc has a load less than or equal to y. An easy calculation shows that for setting ε to some value like $\varepsilon = 0.15$, flow that enters the arc at time 0 cannot reach the end of the arc by time 1.

Consequently, there is no α-earliest arrival s-t-flow for $\alpha = \frac{5}{4}$ for this network with load-dependent transit times. □

4.4 An Approximation Algorithm

Using the interval stacking approach, one can not only show upper bounds for the α-earliest arrival s-t-flow but can also create algorithms for determining such a flow. In contrast to the case of flows over time with constant transit times, the maximum s-t-flow over time problem for flow-dependent transit times is an NP-hard problem (see [33, 55]). Although no approximation algorithms for this problem are known, there are approximation algorithms for the closely related single-source-single-sink quickest transshipment problem as described in Section 2.2.2. A c-approximation algorithm for this problem determines a flow over time that sends the given amount of flow in no more than c times the optimal time from s to t. As we will show in the following, these approximation algorithms for the single-source-single-sink quickest transshipment problem can be used together with the interval stacking to get approximation results also for the α-earliest arrival s-t-flow problem.

Since we will use the interval stacking method from Section 4.3, we have to again worry about the shortest path distance from s to t. We will assume in the following that this distance is at least 1.

Lemma 4.9. Suppose there is a flow over time algorithm $\mathcal{A}$ that computes for any given time horizon $c \cdot T$ ($c \geq 1$) a flow of value at least as large as the value for a maximum s-t-flow over time for time horizon T. Then there exists a $4c$-approximation algorithm for the α-earliest arrival s-t-flow problem.

Proof. In a first step, algorithm $\mathcal{A}$ is used to compute for each $T = 2^i$, with $i \in \{0, \ldots, \lfloor \log(T+1) \rfloor - 1\}$, a flow over time for time horizon cT. By the assumption of the lemma, each of those flows has a value at least as large as a maximum s-t-flow over time for time horizon T.

In a second step, the so computed flows are put together in the same way as the flow f_{IS} in Section 4.3.1, with the difference that the length of the intervals of the stacking is not 2^i but $c2^i$ for $i \in \{0, \ldots, \lfloor \log(T+1) \rfloor - 1\}$. By the same argument as used for proving Theorem 4.5, the so constructed flow can be shown to be a $4c$-earliest arrival s-t-flow. □

Although the above lemma seems to be rather restrictive, it is applicable

both for the inflow-dependent and the load-dependent model. In both cases, the known approximation algorithms for the single-source-single-sink quickest transshipment problem satisfy the conditions of Lemma 4.9. The following theorems from [54] and [55] summarize the corresponding results for the two models.

Theorem 4.10 (Theorem 1 in [54]). Consider a single-source-single-sink instance of the quickest transshipment problem with inflow-dependent transit times where all transit time functions are non-decreasing step functions. If there is a flow over time with inflow-dependent transit times sending D units of flow from s to t within time T, then there exists a temporally repeated flow with inflow-dependent transit times satisfying demand D within time horizon at most $2T$. Moreover, such a flow can be computed in strongly polynomial time.

The algorithm to compute such an approximated quickest transshipment uses the bowgraph as described in Section 2.2.2. The next theorem states the corresponding result for load-dependent transit times.

Theorem 4.11 (Theorem 3.1 in [55]). If there is a flow over time which sends D units of flow from s to t within time T, then there exists a temporally repeated flow satisfying demand D within time horizon at most $2T$. Moreover, for every $\varepsilon > 0$, one can compute a temporally repeated flow in polynomial time which satisfies demand D within time horizon at most $(2+\varepsilon)T$.

Also the approximation scheme of Hall, Langkau, and Skutella [33] for the single-source-single-sink quickest transshipment problem in the inflow-dependent model can be used for approximating an α-earliest arrival s-t-flow, following the approach of Lemma 4.9. The idea of the approximation scheme in [33] is to first compute a relaxed flow for a time horizon T. This relaxed flow is not necessarily feasible for the inflow-dependent setting, however, the construction can be used to show that this relaxed flow gives an upper bound on the amount of flow that a maximum s-t-flow over time can send during that time horizon in the inflow-dependent model. In a second step, the relaxed flow is augmented to be a feasible flow in the inflow-dependent model where this augmentation does not change the value of the flow and enlarges the necessary time horizon by no more than a factor of $1+\varepsilon$ (we omit further details of this algorithm and refer to [33]).

Summing up the observations on the single-source-single-sink quickest transshipment algorithms from [33, 54, 55], we can draw the following conclusion.

Corollary 4.12. There exist $4c$-approximation algorithms for the α-earliest arrival s-t-flow problem, where $c = 2$ for the inflow- and load-dependent flow over time model, and $c = 1+\varepsilon$ for the inflow-dependent flow over time model for every $\varepsilon > 0$.

4.5 Practical Results

In the previous section, we provided the method of using an interval stacking to find a constant upper bound for the α-earliest arrival s-t-flow problem in the case of flow-dependent transit times. Since the given upper bound of 4 obviously only roughly estimates the quality of the approximation, it is interesting to ask which values of α are reachable in practice with this approach. Due to the fact that there are no known exact algorithms to compute maximum s-t-flows over time for flow-dependent transit times, we implemented the 8-approximation algorithm for the inflow-dependent model using the 2-interval stacking together with the quickest transshipment approximation algorithm that uses the bowgraph as described in Section 2.2.2 (see Corollary 4.12) and conducted some computational tests.

We only used a small example and it does not allow to draw conclusions about the practical quality of the approximation in general, however, it proves that the approach is not only of theoretical interest. It also shows a rather big gap between the theoretical and the practical ratio.

The network consists of 22 nodes and 66 arcs. As transit time function we used the B.P.R. function which is a well accepted transit time function for traffic networks (see [74] for further details on the B.P.R. function[1]). This function is given by $\tau(f(a)) = t_a^0 \left(1 + \gamma \left(\frac{f(a)}{c'_a}\right)^\beta\right)$, where t_a^0 is the transit time of arc a when a is empty, c'_a is the "practical capacity" of a; the parameter γ is set to $\gamma = 0.15$ and for β the values $\beta \in \{1, 2, 4, 6\}$ are chosen. As zero flow transit time t_a^0 we used three different values to simulate the situation for three different groups of people: a group of senior citizens, a group of 'normal' people, and a group of rather sportive people. The aim is to send as many people as possible as early as possible from a central hall of the building to the area in front of the building within a time horizon of 5000 seconds. Figure 4.3 below shows a sketch of the floor plan.

Theoretically, we know that at least as much flow reaches the sink at a time θ as is maximal for a time $\frac{\theta}{8}$. However, the actual performance of the

[1] To our knowledge, there are no similar easy functions considering pedestrian's behavior that describe the influence of the amount of flow entering an arc to the total transit time on an arc. For more details on pedestrian behavior see for example [73].

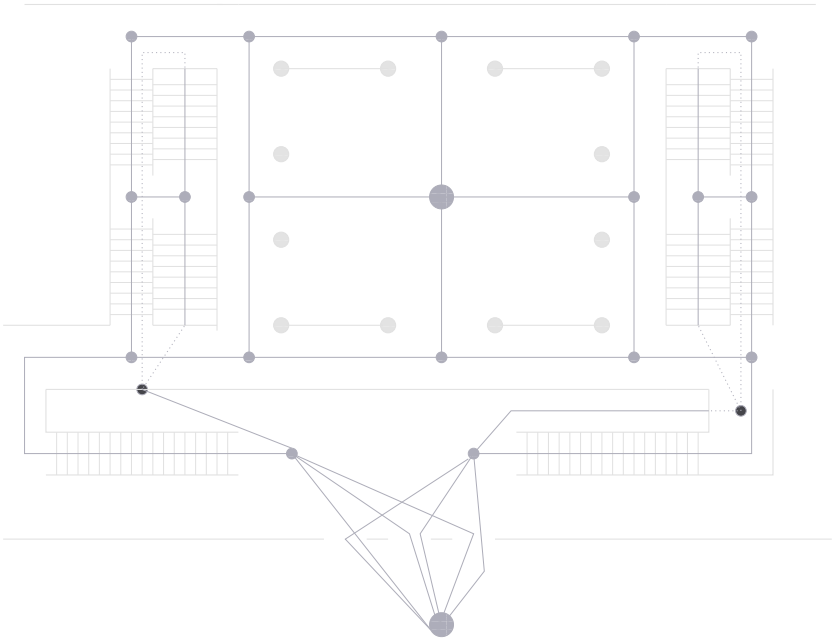


Figure 4.3: Floor plan of a large building together with its network representation.

algorithm for our small example turns out to be much better; for big θ even near-optimal. The fastest people in the group of sportsmen need at least 96 seconds and start with an α-value of 4.8; senior citizens need nearly 400 seconds but have an α-value of no more than 4.0. For later points in time the α-value decreases to a nearly constant value of about 1.3 for sportsmen (after 1500 seconds) and for the slow people to a value between 2.0 and 1.5 (after 2800 seconds). These computed values for α are surprisingly small compared to the theoretical bound of 8.

Another value of interest is the relation between the value of a maximum s-t-flow reaching the sink over time compared to the amount of flow sent into the sink by the approximation algorithm for each θ ($0 \leq \theta < T$). Note that here flow values are compared while for determining the α-value one has to compare time steps. We observe that the fraction of the flow sent by the approximation compared to the value of a maximum s-t-flow over time[2] is small for small values of θ but rather high for large θ. Starting at a ratio of

[2]Since there is no known polynomial time algorithm that can compute maximum s-t-flows over time in networks with inflow-dependent transit times, we used the bowgraph, which is the relaxation from [54] (compare also Section 2.2.2 for details on the bowgraph), for getting an upper bound on the value of a maximum s-t-flow over time for this comparison.

zero at the already stated points in time (96 seconds for sportsmen and 400 seconds for senior citizens) the sportsmen's curve reaches a ratio of one half already at time 450 and a ratio over 80 percent at time 2800. The flow that models senior citizens reaches a value of 50 percent of the maximal flow at time 3200 and increases this value to a maximum of slightly more than 60 percent at time 4000 (we considered a time horizon of 5000 seconds). Thus, a comparably good amount of flow reached the sink.

Summing up, the comparably good quality of the practical approximation ratio and the good ratio when comparing the maximal flow to the flow computed by the approximation algorithm suggest that the developed approximation method gives a practically efficient tool for computing good approximate solutions to the α-earliest arrival s-t-flow problem also for larger instances than just this small example.

Chapter 5

Data Evacuation on a Path

5.1 Introduction

In this chapter, we consider the problem where data needs to be sent through a network of processors from a single source to a single sink as it is the case in sensor networks, the web graph transmitting applications as video on demand, wireless transmission of data, and so on. Such a network consists of processors given a certain capacity and connections between these processors having a certain length (transit time). We assume that there is a huge amount of data packages at the source that needs to be transmitted into the sink having as few loss as possible. Considering data transmission, we have to modify the notion of emergency and evacuation. In seldom instances it happens that a mainframe computer breaks down or catches fire such that all data stored on that computer need to be evacuated as it is the case in evacuation of people. More often, the shut down of the connection during a transmission constitutes an emergency and has immediate consequences to the data in the network. All data currently transmitted on broken connections is lost. The worst case is the shut down of all connections in a network at the same time. After a reconnection, it should be possible to send as much lost data packages as possible into the sink. In particular, we would like to avoid sending them from the source again. This requirement is demanded in some real life applications, for example the transmission of pictures from Mars' surface to the Earth or other data streams, where each data is unique and not stored in the source.

So, speaking of data evacuation, we need to evacuate data before a shut down occurs. Therefore, we aim at storing the data packages in the processors along the transmission path. In order to send the data packages also into the sink, we do not store the data package itself, but a copy of it. This copy is stored at a safe site whereas the original data package is sent towards the sink along the insecure connections. The capacity of a processor is bounded. If there are that many copies of data packages stored in a processor such that it is at its full capacity, some already stored data package can get deleted and a copy of the newly arrived one can get stored. The storage and deletion of

copies of data packages in the processors shall be done in such a way, that at each point in time the maximum number of *different* data packages is stored. Further, we require that the data packages stored are as new as possible. Therefore, a steady exchange in the data packages stored is necessary. This requirement is provoked by the applications. During a data transmission, all data packages that have already reached the sink are assumed to be safe. This data is no longer needed to be stored along the path. This must be taken into account when storing and deleting data packages in the processors. If all connections break down at the same time and the processors store the maximum number of different fresh data packages, the evacuation of data can be seen as optimal.

Notice that we only consider the problem of storing the data before an emergency occurs. The resending of data packages after the reconnection is not part of our research.

In Section 5.2, we define the problem of evacuating data more formally. Here, we restrict ourselves to simple paths. We show that the problem of data evacuation classes in the context of network flow models and their relation to earliest arrival flows. In particular, we point out, that the problem reduces to find adequate storage rules for each node, i.e., rules for each processor that know at each point in time, whether to store an arriving data package and which one to delete, if necessary, in order to maximize the number of stored different and fresh data packages at each point in time. Therefore, we overview different kind of storage rules which we will consider in Section 5.3. Especially, we concentrate on preprocessed storage rules. Optimal storage rules for two special types of paths are determined in Section 5.4. Moreover, we show that storage rules for arbitrary paths can directly be derived from storage rules for the presented capacity excess paths and transit time excess paths. In Section 5.6, we analyze some practical results especially on the size of preprocessed storage rules.

5.2 DATA FLOWS

In this chapter we have a formal look at the problem of data evacuation. Considering the problem, we can observe that the processors and its connection can be represented by a network consisting of nodes and arcs. Nodes model the processors and directed arcs model the connections. Since we are only considering the problem of data evacuation on a path, we are given a linear network $\mathcal{N}$ consisting of nodes $s := 0, 1, 2, \ldots, n, t := n + 1$ and arcs $a_i := (i, i + 1)$ for all nodes $i = 0, 1, \ldots, n$. In this path, node s is the source that sends out data packages regularly and node t acts as

Figure 5.1: Path representing a data transmission from the source to the sink via several processors. The numbers in nodes represent the capacity of the processor and number on arcs represent the transit time data needs to travel from one processor to the next one.

the sink to which all data packages need to be transmitted. Intermediate nodes $i = 1, 2, \ldots, n$ model the processors having a certain positive capacity $c(i) > 0$. The total node capacity on the path we denote by C, i.e., $C := \sum_{i=1}^{n} c(i)$. A positive transit time of $\tau(a_i) > 0$ is assigned to each arc a_i, $i \in \{1, \ldots, n-1\}$. The transit time describes, how long a data package needs to be transmitted from one node to the next one.

Such a path is given exemplarily in Figure 5.1. Notice that source s, sink t and arcs $(s, 1)$ and (n, t) are redundant in order to store the maximum number of different data packages, since the nodes s and t are considered to have no capacity to store data. Moreover, all data sent out of source s can be interpreted as sent out of node 1 and data reaching node t also has to reach node n before. Therefore, we restrict ourselves in the following to 1-n-paths where each node is given a capacity, node 1 acts as source, and node n acts as sink.

Regularly, one data package is sent out of source 1 at every integer point in time $\theta \geq 1$ and forwarded along the path to sink n. We call this behavior a *data flow*. To distinguish the different data packages, we assign each data package sent out of the source a unique *attribute*. We do this by numbering them serially in the order they enter the network, thus the attribute is an integer. At time $\theta = 1$ we send the data package with attribute 1 into the network, at time $\theta = 2$ the data package with attribute 2 is sent out of the source and, more general, at time $\theta \geq 1$ the data package with attribute θ enters the network.

The task is to find a way to save as many different data packages in the node storage such that during a later resending a fairly large amount of data packages can reach the sink. However, in order to store the maximum number of different data packages in all nodes together, we need rules that decide for each node which arriving data package to store and, if the node storage is at its full capacity, which one to delete. One reason to delete data can be that data packages are stored multiple times. Another and more important reason is the requirement of fresh data packages. So, it must be allowed to delete data packages with very small attributes in order to allow data packages with higher attributes to get stored. Such a rule that decides about storage and deletion is called *storage rule*. The importance of these storage rules is

obvious since they are the only determinable variables in our problem. We give a classification of types of rules and describe some well-known storage rules in Section 5.3.

With the above definitions we can formally define the *maximum data evacuation problem* as follows.

Problem 5.1. Given a data flow along an 1-n-path, the *maximum data evacuation problem* looks for a storage rule for each node such that the maximum number of newest different data packages is stored at each integral point in time $\theta \in \{0, 1, \dots\}$ in all nodes together.

In order to store as many different data packages as possible, we start by making a trivial but important observation.

Observation 5.2. The storage of C different data packages can occur at first time at time $\theta = C$, if possible by the path topology. At this time, C different data packages have entered the path.

Since time $\theta = C$ plays an important role in the sense that there will be a change in the rules that need to be applied, we call times $\theta = 1, \dots, C$ the *initial phase*.

Relation to Earliest Arrival Flows. The relation of the data evacuation problem to network flow problems and, in particular, to earliest arrival flow problems does not seem very obvious. First of all, the problem of sending one data package at each point in time into the network and forwarding it towards the sink can be seen as a discrete flow over time. If we assume a capacity of one assigned to each arc, such a flow of data packages obviously fulfills capacity constraints and flow conservation constraints. The special property of such a flow is that there is only one possibility for flow to go from source 1 to sink n. In contrast to classical discrete flows over time, this flow cannot be optimized by certain routing techniques. For the problem of data evacuation, we have to specify rules that establish a storage of data packages into the nodes that is optimal for each point in time. This objective value strongly resembles the objective of earliest arrival flows as described in Chapter 2. For each point in time, we require the maximum number of different data packages stored in the nodes. In contrast to network flows, where we only consider the flow value in total, we have to take special care considering data packages. As data is copyable, not only the maximum number of stored data packages but the maximum number of different data packages is required. Speaking in terms of copyable data packages, the *earliest arrival property of a data flow* is obtained, if the maximum number of different data packages is stored in the nodes at each point in time.

5.3 Storage Rules

The storage rules are the core of good solutions to the problem of maximizing the number of stored different data packages. In this section, we want to overview some storage methods.

In general, we can distinguish between *local* and *global* rules. Local rules are rules that decide for some node whether to store a data package or not without taking the contents and actions of other nodes into account. As *actions* we denote the acts of storing and deleting data packages in a node. As local rules, we can think of storage principles already known from classical warehousing. There are for example the first-in-first-out storage rule (FIFO) where the commodity stored at first is taken from the warehouse when it is needed.

Using the first-in-first-out storage rule in a node means that, if the newly arrived data package needs to be stored in a node already at its full capacity, the one to be deleted is the oldest one, i.e., the one with smallest attribute. Another principle is analogously the last-in-first-out storage rule (LIFO).

Global rules act with the knowledge which data packages are stored in all nodes along the path. An obvious global rule can be a first-in-first-out global rule on the path, i.e., store data packages that are not stored elsewhere and delete the oldest one that can be found in any of the nodes, maintaining node capacity restrictions. Obviously, local rules are preferable, since a screening of all nodes in every discrete point in time is very expensive.

In some cases where these global rules are unmanageable, it is possible to reduce the global rule to local rules on nodes via preprocessing. *Preprocessed rules* are defined in advance by the knowledge of the network topology. Virtually conducting the global rule in advance will determine the actions that are done at a point in time $\theta \geq 1$ at node i. Preprocessed rules are only of value if the virtual conducted run of the global rule stops. This means that only actions of finitely many points in time need to get stored but rules for all points in time are determined by this limited number of points in time. In order to establish preprocessed rules, we need some useful definitions.

Definition 5.3. We define an *assignment* to be a snap shot of the situation on the path at a discrete point in time θ. An assignment is uniquely determined by the attributes of the data packages stored in the nodes at time θ. Further, we call two assignments κ_1 and κ_2 *equivalent* if the assignment κ_2 equals an assignment which can be constructed from the assignment κ_1 by simply adding a constant $\Delta \in \mathbb{N}$ to each of the attributes of data packages in nodes. The order of the data packages within a node is neglected.

An example for two equivalent assignments is given in Figure 5.5 on

page 110 when considering the stored data packages at times $\theta = 5$ and $\theta = 8$. The following definition helps us to determine the finiteness of a run of the virtual conduction of the global rule.

Definition 5.4. We say that there is a *period of length* Δ in the assignments when there exists a time $\theta_0 \geq 1$ for which the assignment at time $\theta_0 + c\Delta + i$ is equivalent to the assignment at time $\theta_0 + i$, for all integers $c > 0$, $i \in \{0, \ldots, \Delta - 1\}$. If a period exists, we call the smallest value of θ_0 the *starting point of the period*.

Assume, we are given for each node the actions which it does at each time $\theta \geq 1$, e.g., storage of the newly arrived data package or not and the deletion of data package with a determined attribute κ. If we further know that there will be a periodical behavior in the assignments and we are given the length Δ of the period and its starting point θ_0, then we define the action at time $\theta \geq \theta_0 + \Delta$ as the analogous action done in the first occurrence of the period. We can observe that if we are given actions for all times 1 to $\theta_0 + \Delta - 1$ where θ_0 is the starting point of a period of length Δ, then for each time $\theta \geq \theta_0 + \Delta$ the action done at time $\theta - \lfloor \frac{\theta - \theta_0}{\Delta} \rfloor \Delta$ shall be used. This indicates that, if we can find a period during the virtual conduction of the global rule, we can guarantee the finiteness of this conduction.

Remark: The above description of preprocessed rules bases on the fact that the rule in each node is defined for times $\theta = 1, 2, \ldots, \theta_0 + \Delta - 1$ which are globally known. A rewriting of these rules to local times depending on the first time at which a data package arrives at a node can be done easily. Instead of storing the actions for global times $\theta \in \{1, \ldots, \theta_0 + \Delta - 1\}$ which are points in time regarding the beginning of sending a data flow, we store for each node $k \in \{1, \ldots, n\}$, the actions regarding local times $\theta'(k) \in \{1, \ldots, \theta'_0(k), \ldots, \theta'_0(k) + \Delta - 1\}$. These points in time are fixed regarding to the first point in time at which a data package reaches node k. For each node k, the correspondence between the local time $\theta'(k)$ and the global time θ is done by adding the transit time to node k to the local time, i.e., $\theta = \theta'(k) + \sum_{i=1}^{k-1} \tau(a_i)$. In order to find storage rules, we need the global view on all nodes at each point in time. Therefore, we will only consider global known points in time θ. Here, $\theta = 1$ is the point in time at which the data flow starts and we assume that the pulsing of the time is globally known. A redefinition to times $\theta(k)$ for each node k can be done simply afterwards.

5.4 Determining Optimal Storage Rules

In this section, we want to analyze different path topologies and determine adequate storage rules for each node or all nodes together, respectively. We consider two very special path topologies, *transit time excess paths* and *capacity excess paths*. We will give storage rules for the nodes of such paths. In the last part of this section, we will show that a storage rule for the nodes in arbitrary paths can be constructed by using the results for the special path topologies.

5.4.1 Transit Time Excess Paths

We define *transit time excess paths* to be such paths where for all nodes $i = 1, \ldots, n-1$ and outgoing arcs a_i it holds that

$$c(i) \leq \tau(a_i).$$

This property indicates that the capacity of a node is smaller than or equal to the number of different data packages that can be carried on its outgoing arc at each point in time.

Intuitively, it is promising to store each arriving data package in a node and delete the oldest data package, if node storage is needed, i.e., to use the first-in-first-out (FIFO) storage rule as described in Section 5.1. We can see that, by the path definition, a data package gets deleted from a node before it gets stored in the subsequent node.

The following theorem shows that the intuitively good FIFO storage rule obtains an optimal storing at each point in time.

Theorem 5.5. Assume that for all nodes i, $i \in \{1, \ldots, n-1\}$ and outgoing arcs a_i it holds that $c(i) \leq \tau(a_i)$. Given a data flow in this network, the maximum number of different data packages is stored at each point in time, if all nodes use the FIFO storage rule.

Proof. First observe that a data package κ arriving at node i at time θ will arrive at node $i+1$ exactly $\tau(a_i)$ time units later, i.e., at time $\theta + \tau(a_i)$. Using the FIFO storage rule in node i means that data package κ gets deleted exactly $c(i)$ time units later, i.e., at time $\theta + c(i)$. At this point in time, the data package with attribute $\kappa + c(i)$ arrives at node i.

Now consider a transit time excess path where it holds that $c(i) \leq \tau(a_i)$ for all nodes i. Assume that data package κ arrives at node i at time θ. At time $\theta + c(i)$ data package κ gets deleted from node i. Before data package κ gets deleted at time $\theta + c(i)$, it has not yet arrived at node $i+1$ since $\theta + c(i) \leq$

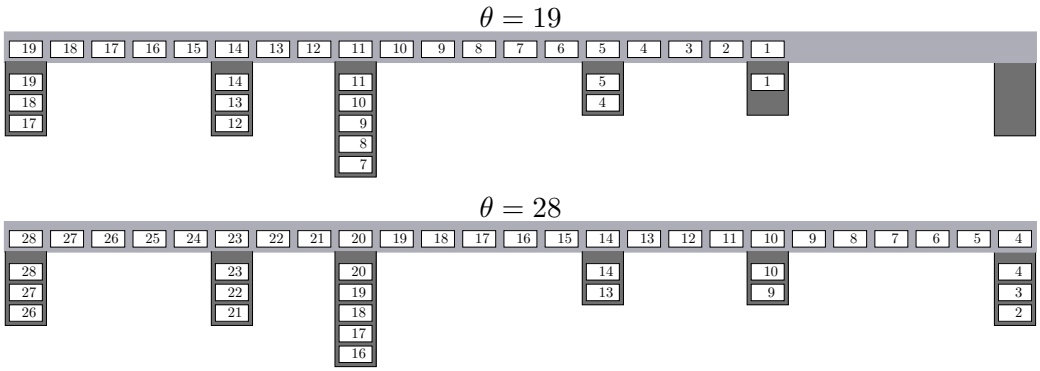


Figure 5.2: Data flow on the transit time excess path as depicted in Figure 5.1 using a first-in-first-out storage rule in each node exemplarily for times $\theta = 19$ and $\theta = 28$.

$\theta + \tau(a_i)$. Therefore, it is stored in no node or just gets stored at node $i+1$ at time $\theta + c(i)$. With the additional property that a node never stores the same data package twice and always is used to full capacity, we have proven the statement of the theorem. Since we use the FIFO storage rule continuously, we further guarantee that there is a steady refreshing of data packages. □

Example: *Figure 5.2 shows exemplarily the assignment of data packages to nodes including the data flow for points in time* $\theta = 19$ *and* $\theta = 28$ *in the transit time excess path as given in Figure 5.1 (ignoring nodes* s *and* t*) using a FIFO storage rule in each node. In the first assignment for* $\theta = 19$*, the first data package has only reached node* 5*. In node* 4 *currently data packages with attributes* 5 *and* 4 *are stored. Using the FIFO storage rule in each node means, that at time* 18 *data packages with attributes* 4 *and* 3 *were stored in node* 4*. Because of the large transit time* $\tau(a_4) > c(4)$ *of arc* a_4*, the data packages with attribute* 3 *has not yet reached node* 5 *and therefore is stored nowhere at time* 19*. Data packages with attributes* $14, 13, 12$ *are currently stored in node* 2*. At time* $\theta = 18$*, data packages with attributes* $13, 12, 11$ *were stored in this node. Data package with attribute* 11 *is currently stored in node* 3*. We can observe that data package* 11 *was stored at the previous point in time and is currently stored, but in the subsequent node. For node* 2 *it holds that the transit time* $\tau(a_2)$ *equals the capacity of node* 2 *which guarantees this permanent storage. At time* $\theta = 28$ *we can see that the first data package has already left the path. This indicates that the stored data packages along the path are refreshed using a FIFO storage rule in each node. Nevertheless, the maximum possible amount of different data packages is stored at both considered points in time since no data package is stored twice.*

The above theorem also holds for the special case where for all nodes i, $i = 1, \ldots, n-1$, $c(i) = \tau(a_i)$ holds. In this case we have special knowledge

of the stored data packages.

Corollary 5.6. When for all nodes $i = 1, \ldots, n-1$ on the path $c(i) = \tau(a_i)$ holds and the FIFO storage rule is used in each node to store data packages, then at every point in time $\theta \geq 1$ data packages with attributes $\max\{1, \theta - C + 1\}$ to θ are stored.

Proof. The correctness follows directly from the observation made in the proof of Theorem 5.5 that, when a data package κ gets deleted at node i, it gets stored at node $i+1$ exactly $c(i)$ time steps later. Thus, all data packages that have not yet left the path are stored. If we consider the initial phase, then all data packages with attributes 1 to θ, which is the data package that entered the network last, are stored. This is the maximum number of different data packages that can be stored for time $\theta \leq C$. For times $\theta > C$, some data packages have already left the path. By construction, we know that the newest C many data packages are stored in the path which is also the maximum possible number of different data packages. Since the data package that entered the path at latest has attribute θ, the oldest data package still stored in some node on the path has attribute $\theta - C + 1$. □

5.4.2 Capacity Excess Paths

Transit time excess paths have the disadvantage that there is a loss of data because of the large transit time between subsequent nodes. In the following, we will analyze a more general setting. For this setting, we will show that always the C newest data packages can be stored. So called *capacity excess paths* are defined by

$$\sum_{i=1}^{k} c(i) \geq \sum_{i=1}^{k} \tau(a_i) \qquad \text{for all } k \in \{1, \ldots, n-1\}. \tag{5.1}$$

The property (5.1) requires that an excess will be guaranteed for all subsequences from the beginning even if there is a transit time on an arc which is larger than the capacity of the node where the arc starts. This property allows us to store as many different data packages as possible in each of the subsequences of nodes $1, \ldots, k$, namely $\sum_{i=1}^{k} c(i)$ many, without loosing data because of too large transit times on single arcs.

The following easy example shows that an apparent first-in-first-out storage rule in each node does not solve the problem of maximizing the number of different data packages at every point in time. The small transit times — small compared to the capacity of the start node of the arc — are the reason

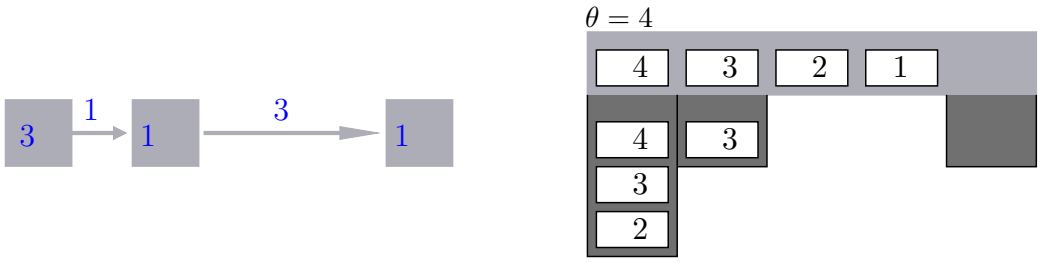


Figure 5.3: Example of a capacity excess path and using a FIFO storage rule for each node. At time $\theta = 4$ there are multiple data packages in the node storage. The assignment and the current data flow of this point in time is depicted on the right hand side.

why this storage rule fails. Using this rule on a capacity excess path, data packages are sometimes stored several times whereas unique data packages can get deleted.

Example: *Figure 5.3 shows an example where the simple FIFO storage rule does not work out to be good in the case of capacity excess paths. On the left hand side of Figure 5.3, a capacity excess paths is depicted. On the right hand side, the situation on the arcs and in the storage for time* $\theta = 4$ *is shown. It can be seen that because of the large transit time between node* 2 *and* 3*, we loose the data package with attribute* 1 *— as in the case of transit time excess paths — but in this setting we have stored the data package with attribute* 3 *twice, in nodes* 1 *and* 2*. In one of those two storage slots, the missing data package with attribute* 1 *could have been stored.*

Considering capacity excess paths, a storage rule needs to take the information stored in all nodes of the path into account to decide which data package has to be stored and which one can be deleted from a considered node.

In the following, we describe a global rule called *delete-smallest-known rule* where, in general, we delete the data package with smallest attribute which is already stored in some other node in case it is necessary to delete a data package from a node. We consider every node one after the other beginning from node n to node 1 at every discrete point in time. As long as the considered node is not at its full capacity, the newly arrived data package can get stored there. If the currently considered node is at its full capacity, we look for a data package in the current node which is also stored in some other node. To find such a data package, we consider each data package κ in the current node in increasing order of its attributes. If data package κ is already stored in some other node, then we delete it in the currently considered node and store the newly arrived data package there. A data package that is stored more than once in nodes along the path at the same time is called a *multiple*

data package.

In order to refresh the data when we have already stored C different data packages, we further check whether the data package with smallest attribute in the whole path (at the beginning of the iteration for time θ) is in the currently considered node. This check is only needed for times θ large enough, which will be shown to be $\theta > C$. If the data package with globally smallest attribute is in the currently considered node, we delete it and store the newly arrived one instead.

The algorithm in pseudocode has the simple structure as depicted in Algorithm 2 on page 103.

Algorithm 2: Delete-Smallest-Known Rule

```
1  foreach time θ = 1, 2, ... do
2      determine oldest data package κ in the path
3      foreach node k = n to 1 do
4          if there is still storage capacity available in node k then
5              store arriving data package in k
6          else
7              if θ > C and κ in node k then
8                  delete κ and store newly arrived data package in k
9              else
10                 foreach data package κ' in k (κ' increasingly sorted) do
11                     foreach node i = 1 to n, i ≠ k do
12                         if κ' in i then
13                             delete κ' from k and store newly arrived data
                               package
14                             go to line 3
```

Figure 5.4 on page 104 applies the delete-smallest-known rule to the capacity excess path as given in the left of Figure 5.3. Looking at the single steps of the algorithm as depicted in Figure 5.4 we can observe that for $\theta \leq C$ all data packages with attributes 1 to θ are stored. For times $\theta \geq C$, the newest C many data packages having attributes $\theta - C + 1$ to θ are stored. We will prove that this observation is a valid invariant after each iteration of the algorithm.

Lemma 5.7 (Invariant). At each point in time after a pass of the delete-smallest-known rule on a capacity excess path, it holds that the data packages

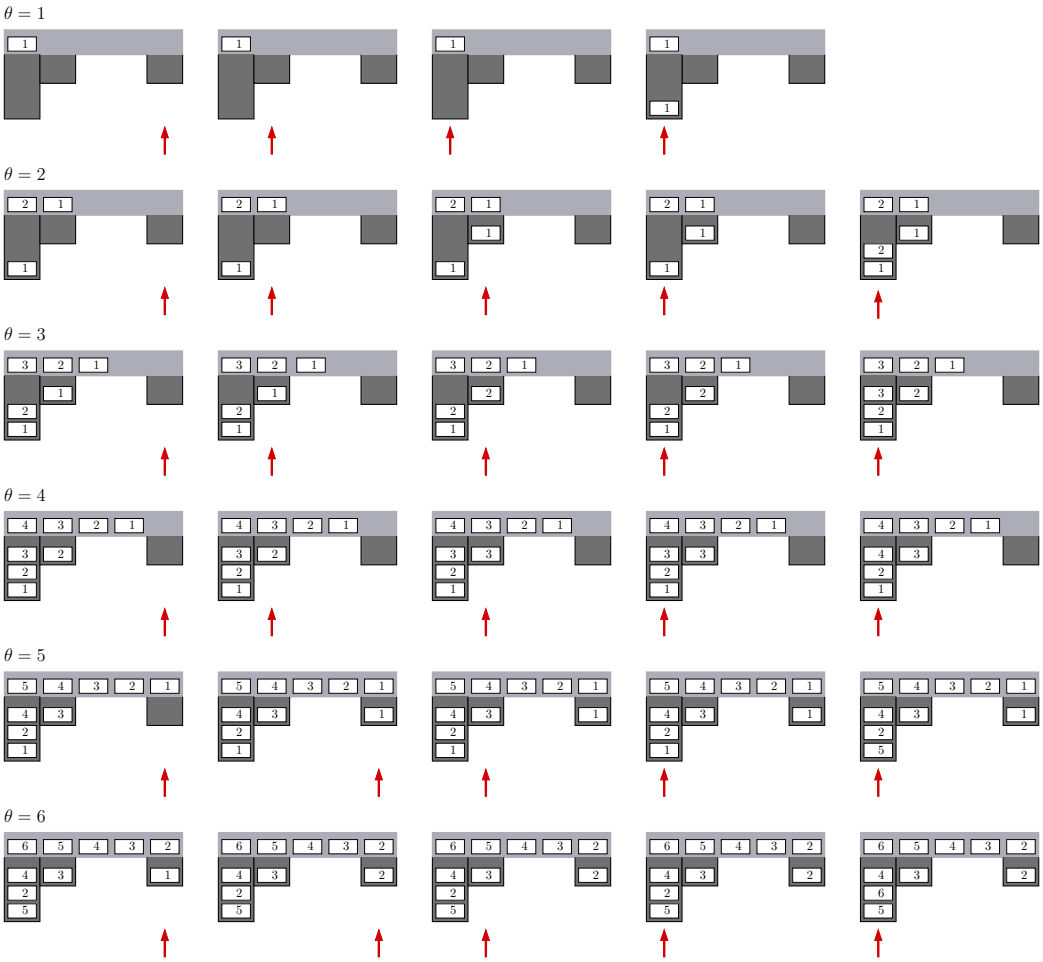


Figure 5.4: Example of the functioning of the delete-smallest-known rule for each time $\theta = 1, 2, 3, 4, 5, 6$. At time $\theta = 1$, a data package arrives only at the first node. There the algorithm checks whether there is capacity available and since it is, it stores the newly arrived data package with attribute 1 which just enters the network. At times $\theta = 2, 3, 4$, new data packages arrive only at nodes 2 and 1. For $\theta = 2$, the algorithm checks whether there is node storage available in those nodes and stores the newly arrived data packages. Checking the second node at time $\theta = 3$, the algorithm finds it at its full capacity and searches whether this data package is stored somewhere else. It finds the data package with attribute 1 in node 1 and thus replaces it in the second node by the newly arrived data package with attribute 2. In the first node, there is still storage capacity available to store the arrived data package with attribute 3. At time $\theta = 4$, the algorithm has to check again whether the data package with attribute 2 currently stored in node 2 can be found in some other node. Since the answer is yes, it replaces this data package by the newly arrived one with attribute 3. The first node is at its full capacity at time $\theta = 4$ and therefore the algorithm has to check in node 1, too, whether there is a data package which is stored multiple times along the path. It starts checking this for data package with attribute 1 and does not find another copy. The same happens for the data package with attribute 2. The data package with attribute 3 is stored in the second node and thus will be deleted from the first node in order to store the newly arrived data package with attribute 4. At time $\theta = 5$ the algorithm starts checking node 3 which still has capacity available. The data package with attribute 1 gets stored there. Checking for another node also containing a data package having attribute 3 fails and therefore the algorithm continues searching for multiple storages of data packages stored in the first node. The first data package checked – the one with attribute 1 – can be found also in node 3 and gets deleted. The newly arrived data package with attribute 5 gets stored in the first node. After this iteration of the algorithm, we have finished the initial phase and this is the first time when we are storing total capacity many different data packages. Therefore at time $\theta = 6$, the algorithm begins again in the last node and searches for the data package having attribute 1. It finds it in the last node and deletes it in order to store the newly arrived data package with attribute 2. In the second node there is only a unique data package stored but in the first node the algorithm finds the data package with attribute 2 which is stored twice. Thus it deletes this data package and stores the arriving one with attribute 6.

with attributes 1 to θ for $1 \leq \theta \leq C$ and $\theta - C + 1$ to θ for $\theta \geq C$ are stored.

Notice that for time $\theta = C$ we can apply both cases of the invariant. Simple computation shows that both intervals of attributes have the same size.

Assuming the correctness of the invariant given in Lemma 5.7, we can directly state the following theorem.

Theorem 5.8. At each point in time, the maximum number of different data packages is stored, if the delete-smallest-known rule is used on a capacity excess path.

Proof of Lemma 5.7. We prove the invariant by induction. We do two parts of induction, one for the first C time steps and then we separately show the invariant for times later than that.

In the very first time step $\theta = 1$, the first data package enters the network. Since no other data package has reached the first node before and we require positive capacities for each node, the arriving data package gets stored. Thus, the beginning of the induction is true. Assume for time θ, $\theta < C$, that all data packages with attributes 1 to θ are already stored; the induction hypothesis. Now consider time $\theta + 1$.

First, we need to show that there is still storage capacity left for the newly arrived data package. This means, we have to look whether there is a storage slot which is still empty or there is at least one data package stored multiple times. At time $\theta+1$, the data package with attribute $\theta+1$ enters the network; let $\bar{k}$ be the node with largest index at which a new data package arrives at time $\theta + 1$. Hence, it holds

$$\sum_{i=1}^{\bar{k}-1} \tau(a_i) \leq \theta + 1 < \sum_{i=1}^{\bar{k}} \tau(a_i).$$

Property (5.1) guarantees that the capacity of nodes 1 to $\bar{k}$ is large enough to carry all $\theta + 1$ data packages that have already entered the network. Assume this is not true, then

$$\theta + 1 \; > \; \sum_{i=1}^{\bar{k}} c(i) \; \geq \; \sum_{i=1}^{\bar{k}} \tau(a_i) \; .$$

The second inequality follows from property (5.1) and thus the choice of $\bar{k}$ is contradicted.

By the choice of $\theta+1 \leq C$ and the above argument, there are either some nodes in $\{1, \ldots, \bar{k}\}$ which are not at full capacity or there are data packages

with the same attribute stored in several nodes. Further, we have to show that the data package entering the network gets stored in some node. This node has to be node 1, since we are only considering positive transit times.

In order to prove the invariant, it suffices to show the following claim, since the algorithm never deletes unique data packages during the initial phase.

Claim. After each consideration of a node $k > 1$ during the iteration for time $\theta + 1$, there exists a node $k' < k$ where it holds that either there is still storage capacity available or it is at its full capacity containing a multiple data package.

Assuming the correctness of the claim, the invariant follows, since the claim also holds for $k = 2$. Then, the algorithm is able to store data package $\theta + 1$ in node 1.

Proof of the Claim. The claim will be shown by induction. By the above observation, there exists either a node not at its full capacity or one containing a multiple data package before starting the iteration of the algorithm for time $\theta + 1$. Assume that the claim is true for some node $k + 1$. In the next step, the algorithm considers node k. If k is at its full capacity and does not contain a multiple data package, it holds that there still exists a node $k' \neq k < k + 1$ containing a multiple data package or having capacity available. Therefore, also $k' < k$ holds and the claim is still true. If k still has capacity available or contains a multiple data package, the algorithm stores the newly arrived data package with attribute in $\{1, \ldots, \theta\}$. By induction hypothesis, this data package is already stored somewhere else. Since it just arrived at node k, the second node $\neq k$ containing this data package lies before node k. Therefore, the claim is true for node k, i.e., there exists some $k' < k$ containing a multiple data package.

Thus, the statement made by the invariant is true for times $1 \leq \theta \leq C$.

Using the correctness of the invariant for the first C times $\theta = 1, \ldots, C$, we will now show the correctness of the invariant for times θ later than that.

The correctness of the invariant for time $\theta = C$ as shown above determines the correctness of the beginning of the second induction. Assume that the invariant is correct for time $\theta \geq C$, i. e., the data packages with attributes $\theta - C + 1$ to θ are stored. In order to prove the invariant, it suffices to show the following claim, since the algorithm never deletes unique data packages during the initial phase.

Claim. After each consideration of a node $k > 1$ during the iteration for

time $\theta + 1$, there exists a node $k' < k$ where it holds that it contains either the oldest data package with attribute $\theta - C + 1$ or a multiple data package.

Assuming the correctness of the claim, the invariant for times $\theta \geq C$ follows, since the claim also holds for $k = 2$. Then, the algorithm is able to store data package $\theta + 1$ in node 1.
Proof of the Claim. The claim will be shown again by induction. By the induction hypothesis there exists a node containing the data package with attribute $\theta - C + 1$ before starting the iteration of the algorithm for time $\theta + 1$. Assume that the claim is true for some node $k + 1$. In the next step, the algorithm considers node k. If k does not contain data package $\theta - C + 1$ or a multiple data package, it holds that there still exists a node $k' \neq k < k + 1$ containing the data package with attribute $\theta - C + 1$ or a multiple data package. Therefore, also $k' < k$ holds and the claim is true for node k. If k contains data package $\theta - C + 1$ or a multiple data package, the algorithm deletes the corresponding data package and stores the newly arrived data package having an attribute in $\{\theta - C + 2, \ldots, \theta\}$. Notice that, by property (5.1), the data package with attribute $\theta - C + 1$ has already left the path at time $\theta + 1$. Again it holds by induction hypothesis, that there exists some $k' < k$ containing a multiple data package, i.e., the induction is true.

This concludes the proof of the invariant. □

Remarks: Reconsidering the proof of the invariant, several observations can be made concerning the deletion and storage for times $\theta > C$. First, the algorithm will definitely delete the oldest data package, since there are no multiple data packages stored along the path before each iteration. Its attribute before the iteration for time θ is determined by the invariant as $\theta - C$. Deleting this data package from its node, say k, the data package stored instead of the oldest one has attribute $\kappa := \theta - \sum_{i=1}^{k-1} \tau(a_i)$. We know by the invariant, that this one was already stored somewhere else, if $k \neq 1$. Moreover, since before the deletion, there were C different data packages stored, there is now one that occurs exactly twice along the path. The remaining $C - 1$ data packages are not touched and therefore still different. Thus, it suffices to look for a data package having attribute κ in nodes $k' < k$ during the whole iteration instead of checking for each data package whether it is stored multiple times. Using this argument repeatedly, it suffices to only look for the data packages that were newly stored in some node $k \neq 1$, replacing a multiple data package or the oldest one, during the following steps of the iteration. If the considered node $k = 1$, we have obviously stored the newest data package that just entered the network.

To use the global delete-smallest-known rule online for every time step would make the computation very complex. Therefore, we are interested in finding a possibility to use the rule as a preprocessing algorithm to define storage rules for each node locally. In the following, we will show that the delete-smallest-known rule acts deterministically and that there is a period after which the assignment of attributes to nodes at time $\theta_0 := C$ recurs. This guarantees the finiteness of a preprocessing variant of the algorithm. Before, we make an observation concerning the correlation between the difference in the attributes of data packages and the time difference considering two equivalent assignments.

Observation 5.9. Given two equivalent assignments computed by the delete-smallest-known rule on a capacity excess path. Then the difference in times at which these assignments occur equals Δ, if the difference in attributes of data packages equals Δ.

Proof. We are given two equivalent assignments which occurred at times θ_1 and θ_2, $\theta_1 < \theta_2$. First, assume that $\Delta < \theta_2 - \theta_1$. Consider the assignment at time θ_2. There the data package with largest attribute has attribute θ_2. The corresponding data package in the equivalent assignment at time θ_1 then should have attribute $\kappa := \theta_2 - \Delta > \theta_1$. By the invariant, the data package with attribute κ has not yet entered the network at time θ_1 which contradicts the assumption. Now assume that $\Delta > \theta_2 - \theta_1$. In this case, consider the data package with smallest attribute, i. e., $\theta_2 - C + 1$, in the assignment at time θ_2. At time θ_1, the corresponding data package should have attribute $\kappa' := \theta_2 - C + 1 - \Delta < \theta_1 - C + 1$. By the invariant, the data package carrying attribute κ' has already left the path at time θ_1 and this contradicts again the assumption. Thus, $\Delta = \theta_2 - \theta_1$.

If two assignments are equivalent with respect to a difference Δ in the attributes, then they appear at times that differ by Δ. □

Lemma 5.10. The delete-smallest-known rule acts deterministically in the sense that it will do the same actions in the same nodes for two equivalent assignments at times θ_1 and θ_2. The subsequent assignments at times $\theta_1 + 1$ and $\theta_2 + 1$ are equivalent as well.

Proof. Assume again, we are given two equivalent assignments which occurred at times θ_1 and θ_2. Observe that equivalent assignments can only occur for times $\theta \geq C$, i. e., the maximum number of different data packages is stored. Line 7 of the delete-smallest-known rule given in Algorithm 2 determines the node in which the oldest data package is found. By the equivalence

of the two assignments, the data package with smallest attribute at time θ_2 will be found in the same node k as the one for the assignment at time θ_1. The newly arrived data package at node k at time θ_2 has attribute $\theta_2 - \sum_{i=1}^{k-1} \tau(a_i)$ which by Observation 5.9 corresponds to the attribute $\theta_1 - \sum_{i=1}^{k-1} \tau(a_i)$ in the assignment at time θ_1. By the equivalence of the assignments at times θ_1 and θ_2, data packages carrying these attributes will be found in a node $k' < k$ which is, by equivalence of the assignments at times θ_1 and θ_2, the same for both assignments. Hence, the algorithm does the same action in the same node in both assignments. Using this argument repeatedly for this new data package which has to get deleted from some node $k' < k$, if $k \neq 1$, the first statement of the lemma is proven. Further, it directly follows that the assignments at times $\theta_1 + 1$ and $\theta_2 + 1$ are equivalent as well. □

The above lemma implies that, if we find once a pair of equivalent assignments, the overall behavior of the assignment of data packages to nodes is periodical as will be shown in Lemma 5.11. That is, there will be a time θ_1 and a time θ_2 for which the assignments are equivalent. If θ_2 is the smallest point in time for which an equivalent assignment to the one at time θ_1 can be found, we will then denote the length of the period by $\Delta := \theta_2 - \theta_1$. The following lemma indicates that such a period can be found. After that we show how to determine a pair of equivalent assignments with minimal difference in time and how to adapt the algorithm to become a preprocessed one.

Lemma 5.11. The delete-smallest-known rule will find a periodical behavior in the assignments of data packages to nodes. A period has length at most $C!/\prod_{i=1}^{n}(c(i)!)$.

Proof. An amount of C data packages can be ordered in $C!$ different ways in the nodes. Since we do not care about the ordering of data packages within a node, we have to divide this number of assignments, i.e., $\prod_{i=1}^{n}(c(i)!)$. So, after at most $1 + C!/\prod_{i=1}^{n}(c(i)!)$ time units, one ordering has to reappear.

Using the deterministic behavior of the delete-smallest-known rule as shown in Lemma 5.10, we know that once two equivalent assignments are found at times θ_1 and θ_2, the assignments at times θ_1+1 and θ_2+1 are equivalent, too. Extending this argument, it is clear that assignments at times θ_1+i and $\theta_2 + i$, $i \geq 0$, are equivalent. Hence, for every time $\theta_1 + i$, $i \geq \Delta$, there exists always an equivalent assignment in $\{\theta_1, \ldots, \theta_1 + \Delta - 1\}$, namely the assignment at time $\theta_1 + i - \lfloor \frac{i}{\Delta} \rfloor \cdot \Delta$. Therefore we get a periodical behavior of the assignments. □

Example: *In Figure 5.5 we can observe a periodical behavior for the capacity excess path given in Figure 5.3. The assignments at times* $\theta_1 = 5$ *and* $\theta_2 = 8$

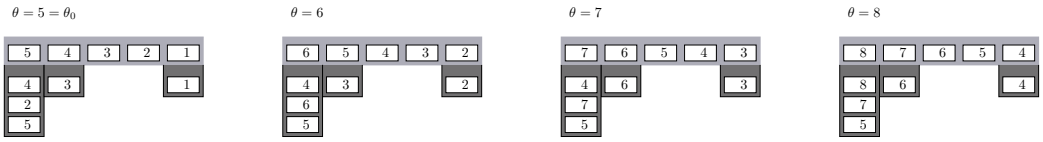


Figure 5.5: Periodical behavior of the example given in Figure 5.3. Here $\Delta = 3$.

are equivalent with respect to a difference of 3 *in the attributes which equals the length of the period* Δ*. The period starts at time* C*. This is not an coincidence but always the case as it is shown in Lemma 5.12.*

In order to use the delete-smallest-known rule as a preprocessing algorithm we need to find the period in the assignments. Therefore, we need to compare every pair of assignments that occur during the run of the algorithm. The next lemma states that it is enough to compare every new assignment computed for some time θ with the assignment of time $\theta_0 := C$.

Lemma 5.12. Using the delete-smallest-known rule for the nodes of a capacity excess path, there can always be found an equivalent assignment to the one at time $\theta_0 := C$.

Proof. By Lemma 5.11 we know that there is a periodical behavior and hence, there are two equivalent assignments at times θ_1 and θ_2. Let θ_1 be the minimal point in time for which an assignment reappears with respect to a difference in the attributes. Assume that $\theta_0 \neq \theta_1$. Since θ_0 is, by choice, the first point in time at which C different data packages are stored and all later assignments also store that many different data packages, it follows that $\theta_1 > \theta_0$. By the minimal choice of θ_1, it has to hold that the assignment at time $\theta_1 - 1$ is not equivalent to the assignment at time $\theta_2 - 1$. By subtracting $\Delta = \theta_2 - \theta_1$ from each attribute of the data packages in the assignments at times θ_2 and $\theta_2 - 1$, we yield that it is possible to obtain the same assignment at time θ_1 starting from two different assignments at time $\theta_1 - 1$. By proving the following claim, we show that this is impossible and therefore contradict the assumption that $\theta_1 \neq \theta_0$.

Claim. The predecessor assignment at time $\theta - 1$ of an assignment at time $\theta > C$ is uniquely determined when using the delete-smallest-known rule.

Proof of the claim. Consider the assignment at time θ. We know that the iteration of the algorithm for time θ has touched all nodes $\{k_1, \ldots, k_\ell\} \subseteq \{1, \ldots, n\}$ containing a data package with attribute $\theta - \sum_{i=1}^{k_j - 1} \tau(a_i)$, $j \in \{1, \ldots, \ell\}$. These data packages are stored in the nodes during this iteration and some other data packages are deleted instead. Further, we know by the

delete-smallest-known rule, that the touched node with largest index, i.e., k_ℓ, must have stored the oldest data package of the previous iteration for time $\theta - 1$, namely data package with attribute $\theta - C$. Thus, the position of the oldest data package in the assignment at time $\theta-1$ is uniquely determined. By the same argumentation and the remark made following the proof of the invariant, that the data packages to exchange during an iteration are uniquely determined once the oldest data package is found, we can conclude, that the predecessor assignment at time $\theta - 1$ is uniquely determined as well. □

Given the above theoretical results about the existence of a period, the starting point of such a period and the observations made for the second phase of the rule where we have already stored C many different data packages, we can rewrite the Algorithm 2 and determine the local rules for each node as depicted in Algorithm 3 at page 119.

So, the storage rule can be described as doing up to time θ_0 all actions as remarked and for all later times θ do the action stored for time $\theta - \lfloor \frac{\theta-\theta_0}{\Delta} \rfloor \cdot \Delta$.

Theorem 5.13. Given a capacity excess path and using the storage rules defined by the delete-smallest-known preprocessing algorithm for each node, the maximum number of newest different data packages is stored at every discrete point in time.

Proof. The proof of this theorem follows directly by the correctness of the invariant (Lemma 5.7) and the finiteness of Algorithm 3 which is guaranteed by Lemma 5.12 together with Lemma 5.11. □

The following theorem goes a step farther and motivates the choice of capacity excess paths as they are defined.

Theorem 5.14. It is possible to store the maximum number of newest consecutive data packages that entered the path if and only if the path is a capacity excess path fulfilling property (5.1).

Proof. The sufficiency of the statement is obvious by the existence of the delete-smallest-known rule. It remains to show, that the path fulfills property (5.1) if the maximum number of newest consecutive data packages are stored. If always the maximum number of newest consecutive data packages can be stored, then there is no loss of data packages, in particular during the initial phase. This means that there are no large transit times between two subsequent nodes that cannot be absorbed by high capacity of nodes before those two nodes. Thus, obviously property (5.1) must be fulfilled. □

5.5 ARBITRARY PATHS

In the above sections, we have described very special path topologies – transit time excess paths and capacity excess paths. In the following we will consider arbitrary paths. Each arbitrary path consists of nodes with time excess, i. e., $c(i) \leq \tau(a_i)$ for some $i \in \{1, \ldots, n-1\}$, and of nodes with capacity excess, i. e., $c(i) \geq \tau(a_i)$ for some $i \in \{1, \ldots, n-1\}$ in arbitrary order. A possible partition would be one which finds sequences of nodes building transit time excess path as defined in Section 5.4.1 and sequences of nodes building capacity excess paths as defined in Section 5.4.2. In order to uniquely assign a node to a sequence, we further add some special properties. It will be shown that for such a partition the use of the adequate storage rules on the sequences already determines an optimal storage rule for the total path. Before showing the optimality, we need to show that such a partition into largest possible sequences forming a capacity excess path and sequences forming a transit time excess path is always possible.

Lemma 5.15. Each arbitrary path can be partitioned into disjoint sequences $I_1, \ldots, I_\ell$ of nodes with the property that for each $I_h = \{j, \ldots, j'\}, h \in \{1, \ldots, \ell\}$ either

$$c(j) > \tau(a_j) \tag{5.2}$$

$$\text{and} \quad \sum_{i=j}^{k} c(i) \geq \sum_{i=j}^{k} \tau(a_i) \qquad \forall k \in \{j+1, \ldots, j'-1\} \tag{5.3}$$

$$\text{and} \quad \sum_{i=j}^{j'} c(i) < \sum_{i=j}^{j'} \tau(a_i) \qquad \text{if } j' \neq n \tag{5.4}$$

or

$$c(i) \leq \tau(a_i) \qquad \forall i \in \{j, \ldots, j'-1\} \tag{5.5}$$

$$\text{and} \quad c(j') \leq \tau(a_{j'}) \qquad \text{if } j' \neq n \tag{5.6}$$

$$\text{and} \quad c(j'+1) > \tau(a_{j'+1}) \qquad \text{if } j' \neq n \tag{5.7}$$

holds. Such a partition is unique for every path.

The proof of this lemma can be done by a simple induction on the number of nodes and showing that each additional node with arbitrary transit time on its outgoing arc at each position can either be dedicated to one existing sequence, splits one sequence into two, or will build its own sequence. Each touched sequence will then again fulfill either properties (5.2)–(5.4) or properties (5.5)–(5.7). Figure 5.6 gives an example for such a partition.

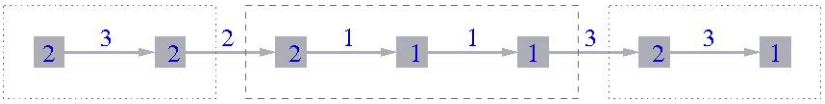


Figure 5.6: Example for a partition of an arbitrary path into disjoint sequences of nodes building a capacity excess node sequence (dashed) and of nodes building transit time excess node sequences (dotted).

Note that we make the additional restriction that capacity excess node sequences start with a node which has a strict excess in capacity compared to the transit time of its outgoing arc by property (5.2). This property ensures that the partition is unique. Each sequence of nodes with equal capacity and transit time of their outgoing arc as the beginning of a capacity excess path now has to be seen as an independent sequence having transit time excess. Conditions (5.4) and (5.7) guarantee that the sequences are chosen as long as possible. Observe that a sequence can consist of only one node. For such a single node i, $i \neq 1$, it obviously holds that $c(i) < \tau(a_i)$ since otherwise it would belong to a longer sequence of increasing capacity fulfilling properties (5.2)–(5.4). For $i = 1$, it can hold that $c(1) \leq \tau(a_1)$ and $c(2) > \tau(a_2)$. Further notice, by the same argument, that each disjoint sequence I_i in the path ends with a node j' where the transit time $\tau(a_{j'})$ of its outgoing arc is larger than or equal to its capacity $c(j')$. We will denote node sequences fulfilling conditions (5.2)– (5.4) by *capacity excess node sequences* and node sequences fulfilling conditions (5.5)– (5.7) by *transit time excess node sequences.*

The following theorem states that the optimal storage rules in such a path can be determined by just using the storage rules for the different sequences of nodes. Intuitively, this gives an optimal storage of data packages because we can compound the nodes of each capacity excess node sequence to a single node i with property $c(i) < \tau(a_i)$. For a capacity excess node sequence we know that at every point in time the newest total capacity many data packages are stored which is also the case for a node using the FIFO storage rule. The maximum number of stored data packages then follows from the maximum number of stored data packages in transit time excess paths. This is one of the main results of this chapter.

Theorem 5.16. Consider an arbitrary 1-n-path with arc transit times and node capacities (positive and integral). If the path is partitioned into capacity excess node sequences and transit time excess node sequences, then the following holds:
If for capacity excess node sequences the delete-smallest-known preprocessing algorithm is used to define storage rules for the nodes and for transit

time excess node sequences the first-in-first-out storage rule is used in each node, then the maximum number of different data packages is stored at each integral point in time.

Proof. The proof partly follows from the correctness of Theorems 5.5 and 5.13, respectively. There the optimality within the sequences of nodes is already given. The global correctness follows from the fact that the considered data packages in each sequence are disjoint. Thus, at every point in time the number of different data packages over all sequences is maximal.

It remains to prove the following claim.

Claim. The data packages currently stored in the different sequences are disjoint.

Proof of the Claim Consider two subsequent non-empty node sequences $A = \{j, \dots, \bar{j} - 1\}$ and $B = \{\bar{j}, \dots, j'\}$ fulfilling properties (5.2)–(5.4) or (5.5)–(5.7) each. As mentioned before, sequence A has to end with a node with outgoing transit time strictly greater than or equal to its capacity. We have to show that a data package entering sequence B is already deleted from sequence A. This together with the optimality in each sequence and that the argumentation holds for every two subsequent sequences in the path we have proven the claim. Now let us look at the different combinations of two subsequent sequences of nodes.

If sequence A fulfills the conditions (5.5)–(5.7), it obviously holds — by the same argumentation as in the proof of Theorem 5.5 — that every data package arriving at sequence B is no longer stored in sequence A.

Now assume that sequence A beginning with node j fulfills conditions (5.2)–(5.4). The first node in sequence B is denoted by $\bar{j}$. By the invariant given in Lemma 5.7, we know — notice that the transit time to the first node in sequence A is $\sum_{i=1}^{j-1} \tau(a_i)$ and therefore data package with attribute 1 reaches node j at time $\sum_{i=1}^{j-1} \tau(a_i) + 1$ — that in sequence A at time $\theta > \sum_{i=1}^{j-1} \tau(a_i)$ the data packages with attributes

$$1 \text{ to } \theta - \sum_{i=1}^{j-1} \tau(a_i)$$

for

$$\sum_{i=1}^{j-1} \tau(a_i) < \theta \leq \sum_{i=j}^{\bar{j}-1} c(i) + \sum_{i=1}^{j-1} \tau(a_i)$$

and attributes

$$\theta - \sum_{i=j}^{\bar{j}-1} c(i) + 1 - \sum_{i=1}^{j-1} \tau(a_i) \text{ to } \theta - \sum_{i=1}^{j-1} \tau(a_i)$$

for

$$\theta \;\geq\; \sum_{i=j}^{\bar{j}-1} c(i) + \sum_{i=1}^{j-1} \tau(a_i)$$

are stored. First, consider all times

$$\theta \;\leq\; \sum_{i=j}^{\bar{j}-1} c(i) + \sum_{i=1}^{j-1} \tau(a_i) \;\; .$$

By the invariant, we know that during the iteration for time

$$\theta_1 \;=\; \sum_{i=j}^{\bar{j}-1} c(i) + 1 + \sum_{i=1}^{j-1} \tau(a_i) \;\; ,$$

the data package with attribute 1 gets deleted from sequence A. This data package with attribute 1 reaches node $\bar{j}$ at time $\theta_2 := \sum_{i=1}^{\bar{j}-1} \tau(a_i) + 1$. Using condition (5.4), we can easily determine that $\theta_2 > \theta_1$. Second, consider times

$$\theta \;\geq\; \sum_{i=j}^{\bar{j}-1} c(i) + \sum_{i=1}^{j-1} \tau(a_i) \;\; .$$

The oldest data package in sequence A has at time θ attribute

$$\kappa_1 := \theta - \sum_{i=j}^{\bar{j}-1} c(i) + 1 - \sum_{i=1}^{j-1} \tau(a_i) \;\; .$$

The data package currently arrived at node $\bar{j}$ at time θ has attribute

$$\kappa_2 := \theta - \sum_{i=1}^{\bar{j}-1} \tau(a_i) \;\; .$$

Comparing the oldest attribute in sequence A and the newly arrived one in sequence B using conditions (5.3) and (5.4), it follows: $\kappa_1 > \kappa_2$. Putting all together, it holds for all times θ that the largest attribute of a data package stored in sequence B is smaller than the smallest one in sequence A. Thus, the newest data package stored in sequence B is even older than the oldest data package stored in sequence A.

This concludes the proof of the theorem.

□

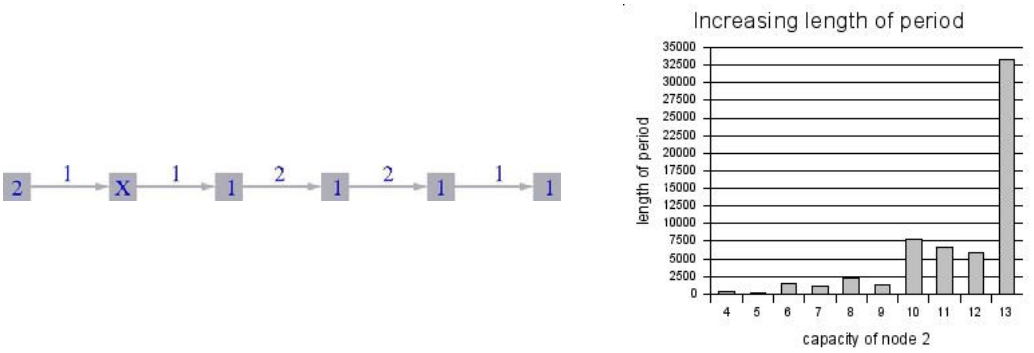


Figure 5.7: (Left) A capacity increasing path with capacities and transit times as depicted. The capacity of the second node is variable. (Right) Lengths of the periods for different values of the capacity of node 2.

5.6 PRACTICAL RESULTS

In the following, we consider the problem of the maximum length of a period. We have shown in Lemma 5.11 that the length of a period for the delete-smallest-known rule can be upper bounded by $C!/\prod_{i=1}^{n}(c(i)!)$. Practical tests on different capacity excess paths show that there are examples where the real length is often smaller than the theoretical bound. However, some instances imply the conjecture that there is no bound of the length of the period which is exponential in C.

In order to analyze the length of the period for some instances of capacity excess paths, we implemented the delete-smallest-known rule. For a given path, it determines the assignments of data packages to nodes for all times $\theta \geq 1$. It stores the assignment at time $\theta_0 := C$ and compares every following assignment with this one. If the algorithm finds an equivalent one, it computes the length of the period and stops.

For comparing the assignments, we order the data packages in each node in increasing order of their attributes. This makes it easier to compare different assignments.

For a test that computes the lengths of periods for a permanent increase of the capacity of one node, we used the instance depicted in Figure 5.7. Here, the second node has a variable capacity denoted by X for which we have used values 4 to 84. Increasing the capacity X of node 2 from 4 to 13, which corresponds to an increase in total capacity of nearly 100%, the length of the period for the latter instance already increases by approximately a factor 100 compared to the length of the period of the first instance where $X = 4$. This can be seen on the right hand side of Figure 5.7. The increase goes on for all higher capacities. For a capacity of 24 of the second node we already

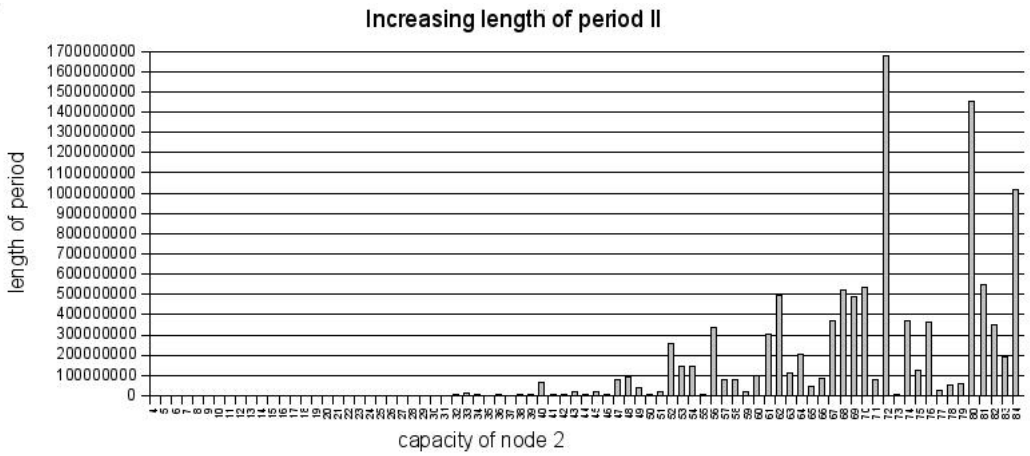


Figure 5.8: Lengths of the periods for the example given in the left of Figure 5.7 for capacities $X = 4, \ldots, 84$.

have a length of the period of 3.212.088 which is nearly 9000 times larger than the length of the period for a capacity of 4 of the second node. In Figure 5.8, we put the chart depicting the length of the periods for all computed values of $X = 4, \ldots, 84$. We have also computed the theoretical bound on the lengths of the period for all these instances, namely $C!/\prod_{i=1}^{n}(c(i)!)$. Analyzing the quotient of the upper bound of the length of the period to the real length of the period, we can see a large distribution of the quotients. On the one hand, there are instances where the real length of the period is 10.000 times smaller than the theoretical upper bound. On the other hand, there are also instances, where the period has length of $\frac{1}{32}$ of the length of the theoretical upper bound. This factor we get for $X = 14$ where the theoretical upper bound has a value of 13.953.600 and the real length of the period is 425.880. There exists also a large number of instances having a period which is only 50 times smaller than the upper bound. On the basis of this observation and other results, we conjecture that there is no bound on the length of a period which is exponential in C.

An interesting observation looking at all those numbers is that nearly every period has an even length and in most cases they are multiples of ten. This can be seen in particular in the example given in Figure 5.9. Since we can observe this phenomenon in different examples, we conjecture that there is a rule to compute the length of the period.

Another interesting observation that can be made in this example is that there are only few odd numbers and most of them are multiples of 5. Further, these multiples only occur in two different settings, namely where the transit time of arc a_2 is set to 4 and to 6. Such an accumulation of multiples of 5

$Y \setminus X$	1	2	3	4	5	6	7
10	500						
9	360	324					
8	900	1120	320				
7	2160	1701	480	343			
6	2500	577	240	96	252		
5	600	2400	512	735	24	175	
4	4900	2100	960	224	540	36	28
3	5040	2268	576	1225	216	24	56
2	3600	576	3136	1008	540	252	84
1	6300	5103	3136	1715	1260	375	112

Figure 5.9: Table of the example as depicted in the right where the transit time is increased at two different positions.

in very special settings was also observed in other examples, as for example the one depicted in Table 5.10. Unfortunately, there is no obvious pattern in order to compute the length of a period.

$Y \setminus X$	9	8	7	6	5	4	3	2	1
2				60					
3			35	280	490	672	1324	2800	3465
4		240	336	560	1008	1134	2240	1782	960
5	315	144	1575	864	567	2400	2970	5760	9009
6	512	2800	5040	7560	6720	11880	27000	6864	15120
7	1155	3696	2205	2640	5940	5760	15444	8064	11700

Figure 5.10: Example of capacity excess path where transit times and capacities are changed. Here an accumulation of multiple of 5 can be observed for $X = 9, 7$.

Conclusion. These practical tests in small instances imply that there are rules to determine the length of a period. Unfortunately, there is no obvious pattern which will help us to find the rules. Finding these rules will be part of further research as well as the analysis of the upper bounds on the length of the period. The task is to find instances, where the length of the period can be proven to not be bounded exponentially in C.

Algorithm 3: Delete-Smallest-Known Preprocessing Algorithm

```
1  foreach time θ = 1, 2, ... do
2      set κ'' = nil
3      foreach node k = n to 1 do
4          consider the newly arriving data package κ at node k at time θ
5          if θ ≤ C − 1 then
6              if there is still storage capacity available then
7                  store arriving data package in node k
8                  storage rule for k at time θ: store arriving data
9              else
10                 foreach data package κ' in k (κ' ≠ κ, increasingly
                   sorted) do
11                     foreach node i = 1 to n, i ≠ k do
12                         if κ' in i then
13                             delete κ' from k and store newly arriving
                               data package κ
14                             storage rule for k at time θ: delete κ'
                               and store arriving data
15                             go to line 3
16         else
17             if κ'' = nil then
18                 set κ'' = θ − C
19             if data package κ'' is in node k then
20                 delete κ'' and store the newly arriving data package κ
21                 set κ'' = κ
22                 storage rule for k at time θ: delete κ'' and store
                   arriving data
23     if θ = C then
24         save assignment and set θ_0 := C
25     if θ > C then
26         if assignment is equivalent to the stored assignment then
27             determine period Δ := θ − θ_0
28             STOP algorithm
```

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows. Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1993. 5

[2] E. J. Anderson and A. B. Philpott, *A continuous-time network simplex algorithm*, Networks **19** (1989), pp. 395–425. 14, 15

[3] J. E. Aronson, *A survey of dynamic network flows*, Annals of Operations Research **20** (1989), pp. 1–66. 5

[4] N. Baumann and E. Köhler, *Approximating earliest arrival flows with flow-dependent transit times*, Special issue of Discrete Applied Mathematics related to MFCS 04. To appear. 5, 79

[5] N. Baumann and E. Köhler, *Approximating earliest arrival flows with flow-dependent transit times*, in Proceedings of Mathematical Foundations of Computer Science 2004: 29th International Symposium MFCS'04, 2004, pp. 599–610. 5, 79

[6] N. Baumann and M. Skutella, *Solving evacuation problems efficiently: Earliest arrival flows with multiple sources*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), 2006. To appear. 5, 44

[7] *A380 evacuation test is approved*. BBC News Online, March 2006. http://news.bbc.co.uk/2/hi/business/4858484.stm. 1

[8] G. N. Berlin, *The use of directed routes for assessing escape potential*, National Fire Protection Association, Boston, MA, 1979. 25

[9] R. E. Burkard, K. Dlaska, and B. Klinz, *The quickest flow problem*, ZOR — Methods and Models of Operations Research **37** (1993), pp. 31–58. 34

[10] R. G. Busaker and P. J. Gowen, *A procedure for determining minimal-cost network flow patterns*, Tech. Report 15, Operational Research Office, John Hopkins University, Baltimore, MD, 1961. 37

[11] M. Carey and E. Subrahmanian, *An approach to modeling time-varying flows on congested networks*, Transportation Research Part B **34** (2000), pp. 157–183. 20, 34, 79

[12] L. G. Chalmet, R. L. Francis, and P. B. Saunders, *Network models for building evacuation*, Management Science **28** (1982), pp. 86–105. 25

[13] W. CHOI, R. L. FRANCIS, H. W. HAMACHER, AND S. TUFECKI, *Network models of building evacuation problems with flow dependent exit capacities*, in Proceedings of the Tenth IFORS International Conference on Operational Research (IFORS'84), Operational Research, 1984, pp. 1047–1059. 40

[14] W. CHOI, H. W. HAMACHER, AND S. TUFECKI, *Modeling of building evacuation problems by network flows with side constraints*, European Journal of Operational Research **35** (1988), pp. 98–110. 40

[15] W. H. CUNNINGHAM, *Testing membership in matroid polyhedra*, Journal of Combinatorial Theory **Ser. B**, no. 36 (1984), pp. 161–188. 22

[16] W. H. CUNNINGHAM, *On submodular function minimization*, Combinatorica **5** (1985), pp. 185–192. 21, 22

[17] L. K. FLEISCHER, *Universally maximum flow with piecewise-constant capacities*, in Proceedings of the 7th Conference on Integer Programming and Combinatorial Optimization (IPCO'99), LNCS 1610, 1999, pp. 151–165. 5

[18] L. K. FLEISCHER, *Faster algorithms for the quickest transshipment problem*, SIAM Journal on Optimization **12** (2001), pp. 18–35. 5, 38, 40

[19] L. K. FLEISCHER, *Universally maximum flow with piece-wise constant capacity functions*, Networks **38**, no. 3 (2001), pp. 1–11. 5, 38

[20] L. K. FLEISCHER AND M. SKUTELLA, *Quickest flows over time*, SIAM Journal on Computing. To appear. 5, 126

[21] L. K. FLEISCHER AND M. SKUTELLA, *The quickest multicommodity flow problem*, in Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO'02), LNCS 2337, 2002, pp. 36–53. 5, 40

[22] L. K. FLEISCHER AND M. SKUTELLA, *Minimum cost flows over time without intermediate storage*, in Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'03), 2003, pp. 66–75. 5, 37, 83

[23] L. K. FLEISCHER AND É. TARDOS, *Efficient continuous-time dynamic network flow algorithms*, Operations Research Letters **23** (1998), pp. 71–80. 5, 28, 31, 37

[24] L. R. FORD AND D. R. FULKERSON, *Constructing maximal dynamic flows from static flows*, Operations Research **6** (1958), pp. 419–433. 1, 3, 9, 14, 15, 18, 30, 36

[25] L. R. FORD AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962. 1, 3, 48

[26] S. FUJISHIGE, *Submodular Functions and Optimization*, Elsevier, 2 ed., 2005. 22

[27] D. Gale, *A theorem on flows in networks*, Pacific J. Math. **7** (1957), pp. 1073–1082. 28

[28] D. Gale, *Transient flows in networks*, Michigan Mathematical Journal **6** (1959), pp. 59–63. 3, 36, 80

[29] A. V. Goldberg and R. E. Tarjan, *Finding minimum-cost circulations by cancelling negative cycles*, Journal of the ACM **36** (1989), pp. 873–886. 11

[30] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Algorithms and Combinatorics 2, Springer, Berlin, 1988. 21

[31] S. Gwynne, E. R. Galea, M. Owen, P. J. Lawrence, and L. Filippidis, *A review of the methodologies used in computer simulation of evacuation from the built environment*, Building and Environment **34** (1999), pp. 741–749. 41

[32] B. Hajek and R. G. Ogier, *Optimal dynamic routing in communication networks with continuous traffic*, Networks **14** (1984), pp. 457–487. 39

[33] A. Hall, K. Langkau, and M. Skutella, *An FPTAS for quickest multicommodity flows with inflow-dependent transit times*, in Proceedings of Approximation, Randomization, and Combinatorial Optimization (APPROX'03), LNCS 2764, 2003, pp. 71–82. 5, 35, 87, 88

[34] L. Hall, A. Schulz, D. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line approximation algorithms*, in Proceedings of the 7th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'96), 1996, pp. 142–151. 85

[35] H. W. Hamacher and S. A. Tjandra, *Earliest arrival flow model with time dependent capacity for solving evacuation problems*, in Pedestrian and Evacuation Dynamics, M. Schreckenberg and S. D. Sharma, eds., 2002, pp. 267–276. 2, 40

[36] H. W. Hamacher and S. A. Tjandra, *Mathematical modelling of evacuation problems: A state of the art*, in Pedestrian and Evacuation Dynamics, M. Schreckenberg and S. D. Sharma, eds., 2002, pp. 227–266. 2, 40, 41

[37] H. W. Hamacher and S. Tufecki, *On the use of lexicographic min cost flows in evacuation modeling*, Naval Research Logistics **34** (1987), pp. 487–503. 40

[38] B. Hoppe, *Efficient Dynamic Network Flow Algorithms*, PhD thesis, Cornell University, 1995. 2, 5, 37

[39] B. Hoppe and É. Tardos, *Polynomial time algorithms for some evacuation problems*, in Proceedings of the 5th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'94), 1994, pp. 433–441. 2, 37, 82

[40] B. HOPPE AND É. TARDOS, *The quickest transshipment problem*, Mathematics of Operations Research **25** (2000), pp. 36–62. 3, 27, 30, 31, 32, 33, 64, 65, 66, 124

[41] M. IRI, *A new method of solving transportation-network problems*, Journal of the Operations Research Society of Japan **26** (1960), pp. 27–87. 37

[42] S. IWATA, *A fully combinatorial algorithm for submodular function minimization*, Journal of Combinatorial Theory, Ser. B **84** (2002), pp. 203–212. 22

[43] S. IWATA, L. K. FLEISCHER, AND S. FUJISHIGE, *A combinatorial strongly polynomial algorithm for minimizing submodular functions*, Journal of ACM **48** (2001), pp. 761–777. 22

[44] J. JARVIS AND H. RATLIFF, *Some equivalent objectives for dynamic network flow problems*, Management Science **28** (1982), pp. 106–108. 25

[45] P. A. JEWEL, *Optimal flow through networks*, Tech. Report 8, Operations Research Center, MIT, Cambridge, MA, 1958. 37

[46] G. KALAFATAS AND S. PEETA, *Network design for evacuation planning*, in Proceedings of the 16th World Congress of the International Federation of Automatic Control (IFAC'06), Elsevier, 2006. To appear. 40

[47] L. G. KHACHIYAN, *A polynomial algorithm in linear programming*, Soviet Mathematics Doklady **20** (1979), pp. 191–194. 21

[48] T. M. KISKO, R. L. FRANCIS, AND C. R. NOBEL, *EVACNET4 - A program to determine the minimum building evacuation time*, 1998. http://www.ise.ufl.edu/kisko/files/evacnet. 1, 41

[49] M. KLEIN, *A primal method for minimum cost flows with application to the assignment and transportation problem*, Management Science **14** (1967), pp. 205–220. 11

[50] B. KLINZ. Cited as personal communication (1994) in [40]. 3, 28

[51] B. KLINZ AND G. J. WOEGINGER, *Minimum cost dynamic flows: The series-parallel case*, in Proceedings of the 4th Conference on Integer Programming and Optimization (IPCO'95), LNCS 920, 1995, pp. 329–343. 17

[52] H. KLÜPFEL AND T. MEYER-KÖNIG, *Simulation of the evacuation of a football stadium*, in Proceedings of Traffic and Granular Flow '03, 2003, pp. 423–430. 1, 41

[53] H. L. KLÜPFEL, *A Cellular Automaton Model for Crowd Movement and Egress Simulation*, PhD thesis, Universtität Duisburg-Essen, Fakultät Naturwissenschaften, 2003. 41

[54] E. Köhler, K. Langkau, and M. Skutella, *Time-expanded graphs for flow-dependent transit times*, in Algorithms - ESA 2002. Proceedings of the 10th European Symposium on Algorithms, LNCS 2461, 2002, pp. 599–611. 5, 20, 34, 35, 79, 83, 88, 90

[55] E. Köhler and M. Skutella, *Flows over time with load-dependent transit times*, in Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'02), 2002, pp. 174–183. 5, 20, 35, 79, 83, 87, 88

[56] B. Korte and J. Vygen, *Combinatorial Optimization*, Springer, 2001. 5

[57] K. Langkau, *Flows over time with flow-dependent transit times*, PhD thesis, Technische Universität Berlin, Fachbereich Mathematik, 2003. 5

[58] Q. Lu, Y. Huang, and S. Shekhar, *Evacuation planning: A capacity constrained routing approach*, in Proceedings of Intelligence and Security Informatics: First NSF/NIJ Symposium (ISI'03), LNCS 2665, 2003, pp. 111–125. 40

[59] S. T. McCormick, *Submodular function minimization*, in Discrete Optimization, K. Aardal, G. L. Nemhauser, and R. Weismantel, eds., Handbooks in Operations Research and Management Science 12, Elsevier, 2005, ch. 7, pp. 321–391. 22

[60] S. T. McCormick and M. Queyranne, *Finding all optimal solutions for submodular function minimization.* Talk given at Oberwolfach workshop ID 0545 on Combinatorial Optimization, November 2005. 60

[61] N. Megiddo, *Optimal flows in networks with multiple sources and sinks*, Mathematical Programming **7** (1974), pp. 97–107. 31

[62] N. Megiddo, *Combinatorial optimization with rational objective functions*, Mathematics of Operations Research **4** (1979), pp. 414–424. 22, 23

[63] D. K. Merchant and G. L. Nemhauser, *A model and an algorithm for the dynamic traffic assignment problems*, Transportation Science **12**, no. 3 (1978), pp. 183–199. 20, 79

[64] T. Meyer-König, H. Klüpfel, J. Wahle, and M. Schreckenberg, *Bypass: Evakuierungssimulation für Fahrgastschiffe*, OR News **7** (1999), pp. 5–8. 1, 41

[65] E. Minieka, *Maximal, lexicographic, and dynamic network flows*, Operations Research **21** (1973), pp. 517–527. 3, 31, 36, 38

[66] R. G. Ogier, *Minimum-delay routing in continuous time dynamic networks with piecewise-constant capacities*, Networks **18** (1988), pp. 303–318. 38

[67] A. B. Philpott, *Continuous-time flows in networks*, Mathematics of Operations Research **15** (1990), pp. 640–661. 36

[68] W. B. POWELL, P. JAILLET, AND A. ODONI, *Stochastic and dynamic networks and routing*, in Network Routing, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, eds., Handbooks in Operations Research and Management Science 8, North-Holland, Amsterdam, The Netherlands, 1995, ch. 3, pp. 141–295. 5

[69] I. RAUF, *Earliest arrival flows with multiple sources*, master's thesis, Universität des Saarlandes, Fachbereich Informatik, 2005. 43, 44, 65, 66, 67

[70] D. RICHARDSON AND É. TARDOS. Cited as personal communication (2002) in [20]. 38, 39

[71] A. SCHRIJVER, *A combinatorial algorithm minimizing submodular functions in strongly polynomial time*, Journal of Combinatorial Theory, Series B **80** (2000), pp. 346–355. 22

[72] A. SCHRIJVER, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer, Berlin, 2003. 5, 11, 60

[73] A. SEYFRIED, B. STEFFEN, AND T. LIPPERT, *Basics of modelling the pedestrian flow*, Physica A **368** (2006), pp. 232–238. 89

[74] Y. SHEFFI, *Urban Transportation Networks*, Prentice-Hall, New Jersey, 1985. 89

[75] S. TJANDRA, *Dynamic Network Optimization with Application to the Evacuation Problem*, PhD thesis, Universität Kaiserslautern, Shaker Verlag, Aachen, 2003. 38, 40

[76] G. TRAUFETTER AND D. DECKSTEIN, *Emergency exit test could make or break superjumbo.* Spiegel Online, May 2005. http://www.spiegel.de/international/0,1518,354366,00.html. 1

[77] W. L. WILKINSON, *An algorithm for universal maximal dynamic flows in a network*, Operations Research **19** (1971), pp. 1602–1612. 3, 36

[78] N. ZADEH, *A bad network problem for the simplex method and other minimum cost flow algorithms*, Mathematical Programming **5** (1973), pp. 255–266. 11, 37

[79] A. ZILIASKOPOULOS, *A linear programming model for the single destination system optimum dynamic traffic assignment problem*, Transportation Science **34** (2000), pp. 37–49. 40