Generalized Temporally Repeated Flows for the Quickest Transshipment and Related Problems

Master Thesis Mathematics

Emma Ahrens

September 30, 2022

Supervised by Prof. Dr. Christina Büsing

Submitted to Lehrstuhl i7: Combinatorial Optimization,
RWTH Aachen University

Contents

List of Figures

Notations

Symbols	Description
$\mathbb{N}_0$	set of non-negative integers
$\mathbb{Z}$	set of integers
$\mathbb{R}_{\geq 0}$	set of non-negative real numbers
$G = (V, A)$	directed graph with nodes V and arcs A
P	set of paths in a graph G
C	set of cycles in a graph G
$\delta^-(v)$	set of ingoing arcs into node $v \in V$
$\delta^+(v)$	set of outgoing arcs into node $v \in V$
$u : A \to \mathbb{N}_0$	capacity function on the arcs A
$\tau : A \to \mathbb{N}_0$	transit time function on the arcs A
$c : A \to \mathbb{N}_0$	cost function on the arcs A
$\mathbf{c} : A \to \mathbb{N}_0^m$	lexicographic cost function on the arcs A
$b : V \to \mathbb{Z}$	balance function on the nodes V
$s_1, \ldots, s_n \in V$	source nodes with $b(s_i) > 0$ for $1 \leq i \leq n$
$v_1, \ldots, v_m \in V$	neutral nodes with $b(v_i) = 0$ for $1 \leq i \leq m$
$t_1, \ldots, t_h \in V$	sink nodes with $b(t_i) < 0$ for $1 \leq i \leq h$
h	number of sinks
$T \in \mathbb{N}_0$	time horizon
	Networks and Flows
(G, u), (G, u, c)	static flow network (with costs)
(G, u, τ), (G, u, τ, c)	flow over time network (with costs)
$(G, u, \tau, \mathbf{c})$	flow over time network with lexicographic costs
$x : A \to \mathbb{R}_{\geq 0}$	static flow
$f_a : \mathbb{Z} \to \mathbb{R}_{\geq 0}$, $a \in A$	flow over time
$y : P \cup C \to \mathbb{R}_{\geq 0}$	flow decomposition of a static flow x
value(x), value(f)	value of a flow
$\overleftrightarrow{G}$	reverse graph additionally containing all backward arcs
G_x	residual graph with regard to a flow x
$(G, u)_x$	static residual network with regard to a flow x

Generalized Temporally Repeated Flows

$P_a(\theta)$	set of paths with flow over arc $a \in A$ at time point $\theta \in \{0, \dots, T\}$
$\text{delay}(f, f')$	starting point of flow f' relative to end point of flow f

1 Introduction

In German hospitals, the shortage of caregivers is a well-known problem. It has existed for several years and there is no improvement in sight, see [Gö14] and [Sla22]. Employees criticize the increasing workload and the resulting decline in the quality of their work [Ger21]. Supplementary to employing additional health care workers (which seems to be easier said than done [Sla22]), the issue might be tackled by optimizing the workflows in the hospitals. The less time logistical processes take up, the more time can be devoted to the patients. One of those processes is the transportation of beds in hospitals. There exists research on the transportation of patients with their beds, e.g. [BLMN10] proposed a two-phase heuristic procedure that aims to reduce waiting time for the patients and the number of used vehicles. In [PBR17], the combined transport of patients and utilities is explored via a discrete-event simulation tool. However, we were unable to find research on the transportation of freshly made beds from the storage to the patients.

Usually, there is one place in the hospital where the beds get new sheets, which we call the *main depot*. When a new patient arrives at a station, an employee calls the main depot and then another worker transports a freshly made bed to the station. The bed makes its way through the hospital and blocks elevators and corridors in the process, compare with Figure 1.1. Since this currently happens during working hours, patients and caregivers are being held up by this transportation. A few questions may arise now: If the number of expected patients per station is known, is a distribution of the beds before the start of the workday possible? How can all demanded beds get transported as fast as possible (hence in a minimal overall timespan)? How can we engage as few employees as possible?

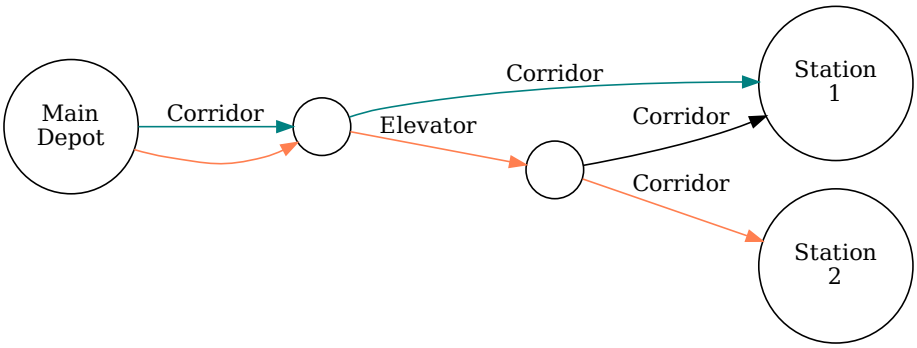

Figure 1.1: A model of the pathways in a hospital that connect the main depot with the stations. Freshly made beds are stored in the main depot and get transported to the stations on demand.

Flow Theory The described problem may be addressed with flow theory, which is a part of the mathematical field of discrete optimization and investigates the transportation of flow on flow networks. A *flow network* consists of a directed graph and several functions on the arcs and nodes of the graph. Originally, flow theory focused on so-called *static flows* [FJF62], where the transportation itself takes no time and only capacities – and possibly costs – of paths are given. Research questions include finding a maximal flow (which transports as many units as possible) or a minimal cost flow (which transports a given number of units with as little cost as possible).

Flow networks can be extended by *transit times*, which describe the time of the transportation along a path. Common research questions on these *flow over time networks* are those of finding (i) a maximal flow within a given time horizon, (ii) a minimal cost flow that satisfies a given demand within a fixed time horizon, or (iii) a flow which satisfies a given demand as fast as possible. The last research problem is called the *Quickest Transshipment Problem*, and it applies best to the question of how to transport the freshly made beds in a hospital as fast as possible for a given demand of each station.

Generally, flows on flow over time networks do not have a certain structure. But there exist special flows over time, the so-called *temporally repeated flows*, that are constructed by repeatedly sending flow along a fixed set of paths in a given time horizon. In a hospital, the usage of a temporally repeated flow for the distribution of beds results in a transportation plan which is easy to remember and execute due to its structure. In Figure 1.1, a teal-colored path leads from the main depot to the first station, and an orange-colored path leads from the main depot to the second station. Sending along those two paths as often as possible results in a temporally repeated flow. For problem (iii), the Quickest Transshipment Problem, there exists an algorithm that computes optimal solutions in polynomial time [HT00]. However, in general, these solutions do not have any structure. The execution of such a plan in a hospital might be too complicated and could make the process even longer instead of quicker.

Herein, we investigate the mentioned research problems on flow over time networks for different types of – more or less structured – flows.

Contribution In this work, we define *uniform flows* which are very similar to temporally repeated flows, but the flow is sent equally often over all paths. Then, we aim to find a compromise between optimal solutions for the Quickest Transshipment Problem and well-structured flows (which might be far from optimal). Therefore, we define k-temporally repeated and k-uniform flows, for $k \in \mathbb{N}_0$, as the consecutive combination of k temporally repeated (resp. uniform) flows.

We analyze the mentioned research problems and show whether there always exist integer optimal solutions, and reduce the Quickest Transshipment Problem to the problem of finding a flow with minimal costs in a given time horizon (Min Cost Flow over Time Problem). Then, we analyze the Quickest Transshipment Problem for k-uniform flows on tree networks from different perspectives. Therefore, we present a naive algorithm that enables us to calculate lower and upper bounds for an optimal solution. Furthermore, we define an equivalence relation on tree networks and show that we may focus our research on *almost-binary tree*

networks, where the underlying graph has a special structure. We compute optimal solutions for small almost-binary tree networks and give an intuition on why those solutions cannot be efficiently reused to calculate optimal solutions for larger networks (e.g. via a greedy algorithm). Finally, we show how to compute an integer solution from the optimal solution of the linear relaxation of the problem.

Additionally, we consider flow over time networks with a single source, a single sink, and costs, and extend the algorithm for finding maximal flows over time (which computes temporally repeated flows [FJF58]) such that it computes maximal flows where the maximal costs over all points in time are minimized. In a hospital, such an algorithm may be used to compute a transportation plan from the main depot to a single station which moves all beds as fast as possible while minimizing the number of working employees.

Organization In Chapter 2, we present the definitions of different types of flows and flows over time, and compare the definition using discrete time points or continuous time. In the subsequent chapter, we state the common research problems, reduce the Quickest Transshipment to the Min Cost Flow over Time Problem and examine the integrality of the given problems. In Chapter 4, we analyze the Quickest Transshipment Problem for k-uniform flows on tree networks. Last but not least, Chapter 5 contains an optimality criterion for lexicographic flows, where the costs are tuples and ordered lexicographically, and the algorithm for finding maximal flows over a given time horizon with minimal costs at each point in time.

2 Theory of Flows

In this chapter, we look at the basic definitions of flow theory specified in other works and extend the theory by defining special types of *flows over time*. Those types are more structured than arbitrary flows over time but not as rigidly structured as temporally repeated flows. The structure of the flows makes them easier to remember and more practical for real-life applications, but at the same time, we are able to find flows that yield better solutions for the problems considered in Chapter 3 than simple temporally repeated flows.

Before we start with the theory of flows, we state a few basic mathematical notions that are used in the rest of this work.

We consider directed graphs $G = (V, A)$, where $\delta^-(v) = \{a \in A \mid a = (w, v)\}$ is the set of ingoing arcs for any node $v \in V$ and $\delta^+(v) = \{a \in A \mid a = (v, w)\}$ is the set of outgoing arcs.

We use the notion of subsets for tuples and write $(u_1, \dots, u_n) \subseteq (v_1, \dots, v_m)$ for $u_i, v_j \in \mathbb{N}_0$, $1 \leq i \leq n$, $1 \leq j \leq m$, if $\{u_1, \dots, u_n\} \subseteq \{v_1, \dots, v_m\}$.

We denote by P a set of paths in a graph G, and for $p \in P$ we write $p = (v_1, \dots, v_m)$ for $v_1, \dots, v_m \in V$. Furthermore, when observing a subpath of p from some node $v_i \in V$ to another node $v_j \in V$, we write $p_{[v_i, v_j]} = (v_i, \dots, v_j) \subseteq p$.

2.1 Static Flows and Flows over Time

In the following, we define the *capacity, cost, transit time*, and *balances* on the nodes and arcs of a graph G which can be found among others in [HT00], [SS14].

Definition 2.1. *For a graph $G = (V, A)$,*

- *the* capacity *function is $u : A \to \mathbb{N}_0$,*
- *the* cost *function is $c : A \to \mathbb{N}_0$, and*
- *the* transit time *function is $\tau : A \to \mathbb{N}_0$.*

Often, we abbreviate the functions and write u_a, c_a, or τ_a instead of $u(a)$, $c(a)$, or $\tau(a)$ for an arc $a \in A$.

Whereas the previous three functions are defined on the arcs of a graph, the next function maps a value to each node.

Definition 2.2 (Balances)**.** *The function $b : V \to \mathbb{Z}$ with $\sum_{v \in V} b(v) = 0$ is a so-called* balance function*. It represents the supply (if $b(v) > 0$) or demand (if $b(v) < 0$) of each node $v \in V$.*

Via the balance function, the set of nodes may be divided into the *sources*, the *sinks*, and the *neutral* nodes in a graph. We name the sources $s_1, \ldots, s_n \in V$ and they each have a supply $b(s_i) > 0$ for all $1 \leq i \leq n$. We call the sinks $t_1, \ldots, t_h \in V$ and they have a demand $b(t_j) < 0$, $1 \leq j \leq h$. The remaining nodes $v_1, \ldots, v_m \in V$ have $b(v_\ell) = 0$, $1 \leq \ell \leq m$.

Throughout this work, we consider only balance functions with exactly one source and name this node $s \in V$.

Remark 2.3. Let us consider the application on hospitals which was discussed in Chapter 1. In the hospital, the freshly made beds are stored in the main depot usually somewhere in the basement. Then, they are transported to various smaller depots at individual stations. We analyze the transportation from the main depot to the smaller depots.

The $h \in \mathbb{N}_0$ smaller depots can be modeled as the sinks $t_1, \ldots, t_h \in V$ and the main depot is the single source $s \in V$ in the graph G. The remaining nodes $v_1, \ldots, v_m \in V$ model the intersections of corridors, elevator exits, and similar crossings. The arcs in the graph G represent the pathways, corridors, elevators, and the like, which connect the places in the hospital. The interpretation of the functions may look like this:

- The balance function b contains the demand for each smaller depot and the corresponding supply for the main depot. The function is set to zero for all other nodes.
- The capacity function u describes for each pathway the maximal number of beds that may be transported through it at the same time. In an elevator, there may not be place for more than one bed, whereas a large corridor could contain many more beds.
- The costs c of each pathway may be used in different ways. We could use the cost function to describe the volume of the transportation of a bed through a pathway. Reducing the costs of a solution therefore reduces the overall volume of the transportation and hence the stress that is imposed on the patients.
- The transit time τ represents the duration of the transportation from one bed along each pathway. Here, transportation via an elevator might be slower than through a corridor.

The composition of a graph with one or more of the previously described functions yields a so-called *network*. We differentiate between networks with or without the transit time function.

Definition 2.4 (Flow Network)**.** *For a directed graph $G = (V, A)$, a capacity function u, and a cost function c, we call (G, u) a* flow network *and (G, u, c) a* flow network with costs*.*

Figure 2.1 shows an exemplary flow network and a corresponding balance function with one source $s \in V$ and two sinks $t_1, t_2 \in V$. We often write *static* flow network if we want to emphasize the difference to the *flow over time network* defined next.

Definition 2.5 (Flow Over Time Network, [KW04])**.** *For a directed graph $G = (V, A)$, a capacity function u, a transit time function τ, and a cost function c, we call (G, u, τ) a* flow over time network *and (G, u, τ, c) a* flow over time network with costs*.*

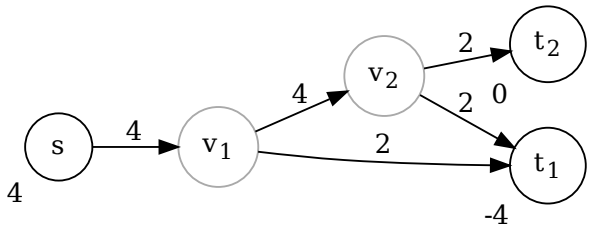

Figure 2.1: A flow network (G, u) with a balance function b. The labels on the arrows describe the capacities and the labels on the nodes specify the balances. Hence, it is $b(s) = 4$, $b(v_1) = 0$, $b(v_2) = 0$, $b(t_1) = -4$, and $b(t_2) = 0$.

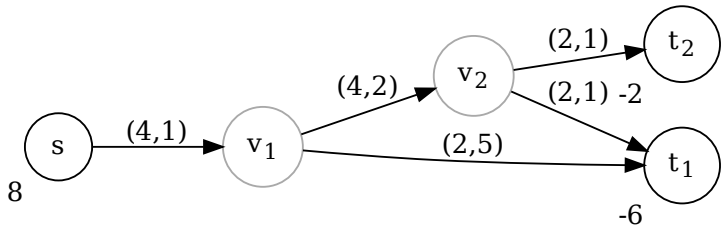

Figure 2.2: A flow over time network (G, u, τ) with a balance function b. The labels (u, τ) on the arrows describe the capacities and the transit times and the labels on the nodes specify the balances.

An example of a flow over time network is given in Figure 2.2. We assume that there exist no arcs $a \in A$ with capacity $u(a) = 0$. Otherwise, they could be deleted from the graph without loss of generality for all considered problems in this work.

In the rest of this section, we give the definitions of the most general flows on static flow networks and flow over time networks. We consider the value of a flow and last but not least, we define residual networks.

We state the definition of a static flow on a static flow network.

Definition 2.6 (Static Flow, [FJF62])**.** *For a flow network* (G, u) *and a balance function* b*, a feasible* static b-flow *is a function* $x : A \to \mathbb{R}_{\geq 0}$*, which satisfies*

1. *the* capacity constraint $0 \leq x(a) \leq u(a)$ *for all* $a \in A$,
2. *and the* flow conservation $\sum_{a \in \delta^-(v)} x(a) - \sum_{a \in \delta^+(v)} x(a) = -b(v)$ *for all* $v \in V$.

The definition of a b-flow for a flow network *with costs* is similar. If the balances are arbitrary, we simply say *flow* instead of *b-flow*.

Example 2.7. When trying to find a static flow that solves the network in Figure 2.1, there exists only one possible solution. The flow x with $x(s, v_1) = 4$, $x(v_1, t_1) = 2$, $x(v_1, v_2) = 2$, $x(v_2, t_1) = 2$ and $x(v_2, t_2) = 0$ fulfills the demand and supply constraints of the balance function, while also satisfying the capacity constraint and flow conservation.

Note that we write $x(v, w)$ instead of $x((v, w))$ for $v, w \in V$ if the meaning is clear. In subse-

quent sections, we use the property that every flow can be decomposed into a set of paths and cycles, called a *flow decomposition*.

Lemma 2.8 (Flow Decomposition, [Wil19])**.** *Let x be a feasible static b-flow. Then, there exists a set of paths P, a set of cycles C, and a function $y : P \cup C \to \mathbb{R}_{\geq 0}$, the so-called* flow decomposition*, which satisfies*

$$x(a) = \sum_{\substack{p \in P, \\ \text{if } a \in p}} y(p) + \sum_{\substack{q \in C, \\ \text{if } a \in q}} y(q) \qquad \forall a \in A.$$

Each flow can be represented by at most $|A|$ paths and cycles $P \cup C$ and the corresponding flow decomposition y, see [Wil19]. We call a flow composition $y : P \cup C \to \mathbb{R}_{\geq 0}$ of a flow x a *path decomposition* if $C = \emptyset$.

The concept of static flows can be generalized to *flows over time* (sometimes also called *dynamic flows*), where it takes a unit of flow a certain amount of time to get from one node to an adjacent node. This is represented in flow over time networks via the transit time function. The common definition of flows over time uses integrals to measure how much flow actually went through the network and assumes that the time is continuous. Here, we consider only discrete time steps. We show the equivalence of both definitions in Section 2.3.

Definition 2.9 (Flow Over Time, [KW04])**.** *For a flow over time network (G, u, τ), a time horizon $T \in \mathbb{N}_0$, and a balance function b, a feasible b-*flow over time *is a family of functions $f_a : \mathbb{Z} \to \mathbb{R}_{\geq 0}$, $a \in A$, which satisfy*

1. *the* capacity constraint $0 \leq f_a(\theta) \leq u(a)$ *for all $a \in A$ and $\theta \in \{0, \dots, T\}$,*
2. *the* flow completion $f_a(\theta) = 0$ *for all $a \in A$ and $\theta > T - \tau_a$,*
3. *the* weak flow conservation *for all $v \in V \setminus \{s\}$ and $\theta \in \{0, \dots, T\}$*

$$\sum_{a \in \delta^-(v)} \sum_{\xi=0}^{\theta - \tau_a} f_a(\xi) - \sum_{a \in \delta^+(v)} \sum_{\xi=0}^{\theta} f_a(\xi) \geq 0,$$

4. *and the* strict flow conservation *for all $v \in V$*

$$\sum_{a \in \delta^-(v)} \sum_{\xi=0}^{T - \tau_a} f_a(\xi) - \sum_{a \in \delta^+(v)} \sum_{\xi=0}^{T} f_a(\xi) = -b(v).$$

Furthermore, $f_a(\theta) = 0$ for $\theta \notin \{0, \dots, T\}$.

For a flow over time network (G, u, τ), a balance function b, and a time horizon T, we define $\mathcal{F}(G, u, \tau, b, T)$ as the set of all feasible flows over time.

Similar to a static flow, a flow over time needs to fulfill the capacity constraint for each arc and (additionally) each point in time. A flow over time also needs to satisfy two flow

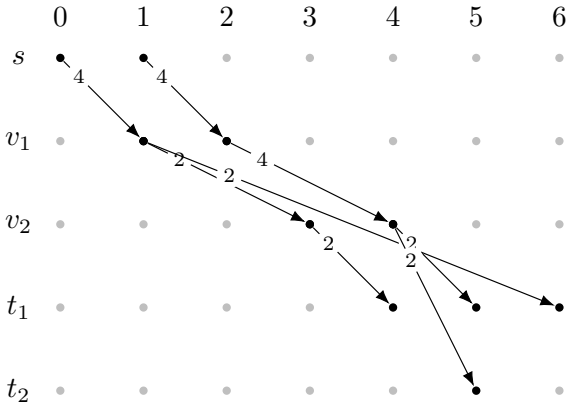

Figure 2.3: A visualization of a flow over time on the network and balance function depicted in Figure 2.2 with time horizon 6. On the x-axis we have the discrete points in time, and on the y-axis the nodes of the network. Here, we send units at time 0 along two paths from the source s to the sink t_1 and at time 1 along two paths from the source s to the sinks t_1 and t_2.

conservation constraints: The weak flow conservation ensures that the outflow of each node is smaller or equal to the inflow. We fix a time horizon that specifies the overall time during which flow is sent along arcs and the strict flow conservation constraint guarantees that for each node, the amount that is received or sent until the end of the time horizon equals exactly the amount specified by the balance function. Furthermore, we ensure that every unit of the flow reaches a node before the time horizon is over. Therefore, we specify that a unit can only be sent along an arc if the transit time is not longer than the remaining time. Take a look at Figure 2.3 for a visualization of a flow over time. For a flow over time network *with costs*, the b-flow over time is defined similarly.

In this work, we consider flows over time without storage in nodes that are neither sources nor sinks. In [KW04], it is shown that every flow over time with storage can be transformed into a flow over time without storage. Hence, we do not need hold-over arcs and the weak flow conservation might also be strict (but for this work, there is no harm in defining it as weak). Furthermore, we also omit the balances if they are irrelevant and simply write *flow over time*.

Now, we define the *value* of static flows and flows over time as the overall number of units that is sent from the source to the sinks.

Definition 2.10 (Value of a Static Flow)**.** *The value of a static b-flow x on a network (G,u) is defined as*

$$\text{value}(x) := \sum_{a\in\delta^+(s)} x(a) - \sum_{a\in\delta^-(s)} x(a) = \sum_{i=1}^{h} \sum_{a\in\delta^-(t_i)} x(a) - \sum_{i=1}^{h} \sum_{a\in\delta^+(t_i)} x(a).$$

The value of a static flow is well-defined, see [Wil19]. It holds that $\frac{1}{2}\sum_{v\in V}|b(v)| = \text{value}(x)$

for any b-flow x. The value of a flow over time also equals the number of sent units.

Definition 2.11 (Value, [SS14])**.** *The* value *of a b-flow over time f on a network (G, u, τ) is defined as*

$$\text{value}(f) := \sum_{a \in \delta^+(s)} \sum_{\theta=0}^{T} f_a(\theta) - \sum_{a \in \delta^-(s)} \sum_{\theta=0}^{T} f_a(\theta) = \sum_{i=1}^{h} \sum_{a \in \delta^-(t_i)} \sum_{\theta=0}^{T} f_a(\theta) - \sum_{i=1}^{h} \sum_{a \in \delta^+(t_i)} \sum_{\theta=0}^{T} f_a(\theta).$$

For a point in time $\theta \in \{0, \dots, T\}$, we also define the value until θ *of a flow over time f as*

$$\text{value}(f(\theta)) := \sum_{i=1}^{h} \sum_{a \in \delta^-(t_i)} \sum_{\xi=0}^{\theta} f_a(\xi) - \sum_{i=1}^{h} \sum_{a \in \delta^+(t_i)} \sum_{\xi=0}^{\theta} f_a(\xi).$$

For example, consider the flow in Figure 2.3 with value 8. It is not obvious that the value of a flow over time is well-defined, hence we show the correctness in the next lemma.

Lemma 2.12. *The value of a flow over time is well-defined.*

Proof. We have to show that for any feasible b-flow over time f, the equality in Definition 2.11 holds. We have

$$\sum_{v \in V} b(v) = 0 \quad \text{and} \quad \frac{1}{2} \sum_{v \in V} |b(v)| = \sum_{\substack{v \in V \\ b(v) > 0}} b(v) = \sum_{\substack{v \in V \\ b(v) < 0}} -b(v).$$

Since $s \in V$ is the only node with $b(s) > 0$ and $t_1, \dots, t_h \in V$ are the only nodes with $b(t_i) < 0$, we derive that $b(s) = \sum_{i=1}^{h} -b(t_i)$.

Due to the strict flow conservation constraint, we know that

$$\begin{aligned} \text{value}(f) &:= \sum_{a \in \delta^+(s)} \sum_{\theta=0}^{T} f_a(\theta) - \sum_{a \in \delta^-(s)} \sum_{\theta=0}^{T} f_a(\theta) \\ &= \sum_{a \in \delta^+(s)} \sum_{\theta=0}^{T} f_a(\theta) - \sum_{a \in \delta^-(s)} \sum_{\theta=0}^{T-\tau_a} f_a(\theta) - \underbrace{\sum_{a \in \delta^-(s)} \sum_{\theta=T-\tau_a+1}^{T} f_a(\theta)}_{=0} = b(s). \end{aligned}$$

Furthermore, it holds that

$$\begin{aligned}\text{value}(f) &= \sum_{i=1}^{h} \sum_{a \in \delta^-(t_i)} \sum_{\theta=0}^{T} f_a(\theta) - \sum_{i=1}^{h} \sum_{a \in \delta^+(t_i)} \sum_{\theta=0}^{T} f_a(\theta) \\ &= \sum_{i=1}^{h} \sum_{a \in \delta^-(t_i)} \sum_{\theta=0}^{T-\tau_a} f_a(\theta) + \underbrace{\sum_{i=1}^{h} \sum_{a \in \delta^-(t_i)} \sum_{\theta=T-\tau_a+1}^{T} f_a(\theta)}_{=0} - \sum_{i=1}^{h} \sum_{a \in \delta^+(t_i)} \sum_{\theta=0}^{T} f_a(\theta) = \sum_{i=1}^{h} -b(t_i)\end{aligned}$$

and thus the equality in Definition 2.11 follows. □

Lastly, we recapitulate the notion of residual networks for static flows as given in [Wil19]. They are needed briefly in Chapter 5. For this purpose, we define the *reverse graph* as the graph that also contains the backward arc for any arc $a \in A$.

Definition 2.13 (Reverse Graph)**.** *For a network (G,u), the reverse graph is defined as $\overleftrightarrow{G} := (V, \overleftrightarrow{A})$ with $\overleftrightarrow{A} := A \cup \overleftarrow{A}$, where $\overleftarrow{A} := \{(w,v) \mid (v,w) \in A\}$.*

Not all backward arcs in a reverse graph are relevant. The *residual graph* contains only the relevant backward arcs.

Definition 2.14 (Residual Graph)**.** *Given a network (G,u) and a static flow x, we define the residual graph $G_x := (V, A_x)$ as*

$$A_x := \{a \in A \mid u(a) - x(a) > 0\} \cup \{(v,w) = \overleftarrow{a} \in \overleftarrow{A} \mid x(w,v) > 0\}.$$

The residual network contains the residual graph together with an adapted capacity function and (possibly) an adapted cost function.

Definition 2.15 (Static Residual Network)**.** *Given a network (G,u) and a static flow x on this network, then the residual network $(G,u)_x$ consists of the residual graph $G_x = (V, A_x)$ and the capacity function*

$$u_x : A_x \to \mathbb{N}_0,\ a \mapsto \begin{cases} u(a) - x(a), & a \in A \\ x(a), & a \in \overleftarrow{A}. \end{cases}$$

Given a cost network (G,u,c) and a flow x, then the residual network is $(G,u,c)_x$ consisting of the residual graph G_x, the capacity function u_x and the residual cost function

$$\mathbf{c}_x : A_x \to \mathbb{Z},\ a \mapsto \begin{cases} \mathbf{c}(a), & a \in A \\ -\mathbf{c}(\overleftarrow{a}), & a \in \overleftarrow{A}. \end{cases}$$

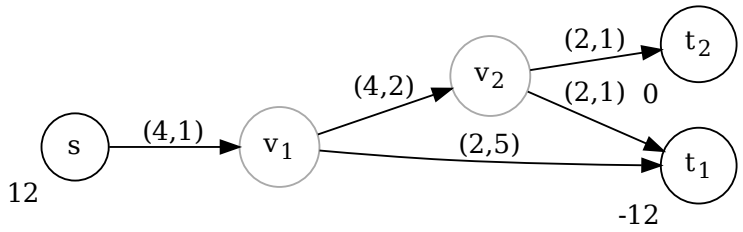

Figure 2.4: Here, we have the same flow over time network as in Figure 2.2, but a new balance function. In Figure 2.5 and Figure 2.6, there are two corresponding flows over time.

2.2 Generalized Temporally Repeated Flows

In this section, we define some special types of flows over time. The temporally repeated flow is quite common in the literature, whereas we introduce uniform, k-temporally repeated and k-uniform flows.

A temporally repeated flow is constructed by using a static flow, decomposing this flow into paths and sending along those paths as often as possible, hence as long as a unit reaches the end of the path before the time horizon is over.

Definition 2.16 (Temporally Repeated Flow, [GKL+18], [Ham89])**.** *Let x be a feasible static flow over a network (G, u, τ) and $y : P \to \mathbb{R}_{\geq 0}$ a corresponding path decomposition.*

Then, the associated temporally repeated flow *f with time horizon T is a flow over time defined as*

$$f_a(\theta) := \sum_{p \in P_a(\theta)} y(p) \qquad \forall a \in A,\ \theta \in \{0, \dots, T\},$$

where $P_a(\theta) := \{p \in P \mid a \in p,\ \tau(p_{[s,v]}) \leq \theta,\ \tau(p_{[w,t]}) \leq T - \theta\}$ for $a = (v, w)$ and $p = (s, \dots, t)$. We set $f_a(\theta) := 0$ for $\theta \notin \{0, \dots, T\}$.

Figure 2.5 shows an example of a temporally repeated flow on the network depicted in Figure 2.4. In the next lemma, we show that a temporally repeated flow is indeed a feasible flow over time.

Lemma 2.17. *A temporally repeated flow f with time horizon T is a feasible flow over time.*

Proof. We show that the constructed flow f satisfies the capacity constraint, flow completion, the weak flow conservation, and the strict flow conservation.

- Since y is a path decomposition of a static flow on the network (G, u, τ) which satisfies the capacity constraint, it holds that

$$f_a(\theta) = \sum_{p \in P_a(\theta)} y(p) \leq \sum_{\substack{p \in P, \\ a \in P}} y(p) = x(a) \leq u(a)$$

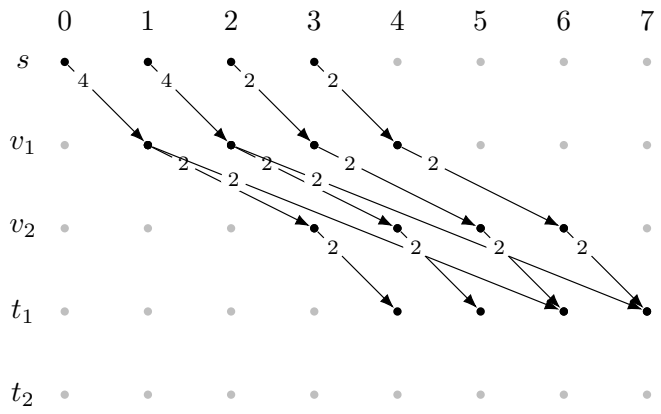

Figure 2.5: A temporally repeated flow on the network given in Figure 2.4 with time horizon 7. Again, points in time are on the x-axis and the nodes on the y-axis. This flow is sent along two different paths, where one path has a total transit time of 4 and the other one has a total transit time of 6. Units can be sent via the shorter path four times, but only two times via the longer path.

for all $a \in A$, $\theta \in \{0, \dots, T\}$. Hence, the capacity constraint is fulfilled.

- For $a \in A$ and $\theta > T - \tau_a$, we obtain

$$f_a(\theta) = \sum_{p \in P_a(\theta)} y(p) = 0,$$

since $P_a(\theta) = \emptyset$ if $\theta > T - \tau_a$. Thus, the flow completion constraint is satisfied.

- The flow over time is defined via paths from the source to the sinks, hence we can define a corresponding balance function b and the weak and strict flow conservation constraints hold accordingly. This can be easily verified.

Hence, a temporally repeated flow is a feasible flow over time. □

The following special type of a flow over time is inspired by the figures in [Ham89]. The flows in the figures are similar to temporally repeated flows with the difference that each path is repeated the same number of times (see Figure 2.6 for an example).

Definition 2.18 (Uniform Flow)**.** *Let x be a feasible static flow over a network (G, u, τ) and $y : P \to \mathbb{R}_{\geq 0}$ a corresponding path decomposition. Furthermore, we set $t := \max_{p \in P} \tau(p)$.*

Then, the associated uniform flow *f with time horizon T is a flow over time defined as*

$$f_a(\theta) := \sum_{p \in P_a(\theta)} y(p) \qquad \forall a \in A,\ \theta \in \{0, \dots, T\},$$

where $P_a(\theta) := \{p \in P \mid a \in p,\ 0 \leq \theta - \tau(p_{[s,v]}) \leq T - t\}$ for $a = (v, w)$ and $p = (s, \dots, t)$. We set $f_a(\theta) := 0$ for $\theta \notin \{0, \dots, T\}$.

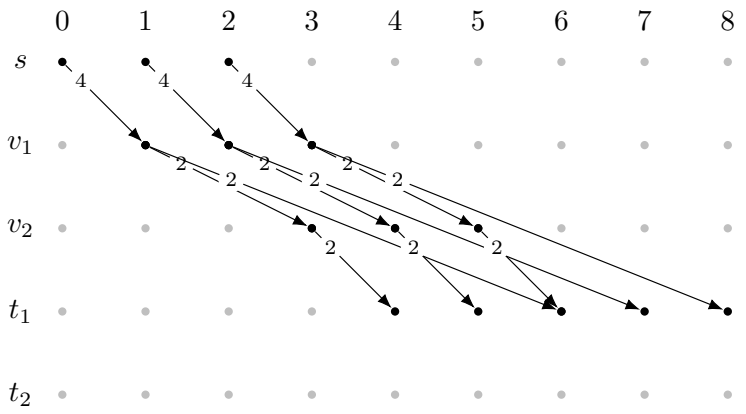

Figure 2.6: A uniform flow on the network given in Figure 2.4 with time horizon 8. The flow is sent along two different paths, where flow is sent along each path the same number of times.

A uniform flow is called *uniform*, since the amount of the new supply is uniform for all points in time where flow is sent from the sink. In Figure 2.6, there is repeatedly a new supply of 4 units in the first time period and then no new supply afterward. We call each point in time when we send flow from the sink along a path an *iteration* and the flow in the example has 3 iterations.

Compared to a temporally repeated flow, a uniform flow is even more intuitive for most humans, since units are sent along all paths the same number of times.

In the rest of this work, we consider a weakened definition of uniform flows where the supply in the last iteration may be smaller (but not zero) for each path. This allows us to find uniform flows for a larger set of balance functions.

Lemma 2.19. *A uniform flow f with time horizon T is a feasible flow over time.*

A uniform flow with time horizon T is very similar to the temporally repeated flow with the same time horizon created from the same static flow.

Proof. The proof of the feasibility is very similar to the proof of Lemma 2.17 and can be verified analogously. □

When comparing the flows in Figures 2.5 and 2.6, we can see that the uniform flow needs at least the same time horizon as the temporally repeated flow to fulfill the same demands given by the balance function, if both flows are created from the same static flow. This holds in general, but the proof is left to the reader.

We may *merge* two temporally repeated (resp. uniform) flows to create a new temporally repeated (resp. uniform) flow by executing both flows simultaneously.

Definition 2.20 (Merge Flows)**.** *For two flows f, f' on the same network (G, u, τ) with time horizon T,* merging *them results in a new flow f'' defined as*

$$f''_a(\theta) := f_a(\theta) + f'_a(\theta) \qquad \forall a \in A,\ \theta \in \{0, \dots, T\}.$$

Merging two flows only results in a feasible flow if the addition of the flows does not violate the capacity constraint. If both flows f and f' are temporally repeated (resp. uniform) flows, then the resulting flow is also temporally repeated (resp. uniform if the number of iterations is equal).

We may also *combine* two temporally repeated (resp. uniform) flows to create a new flow over time by executing both flows consecutively.

Definition 2.21 (Combine Flows)**.** *For two flows f, f' on the same network (G, u, τ) with time horizons T and T',* combining *them results in a new flow f'' defined as*

$$f''_a(\theta) := \begin{cases} f_a(\theta), & \theta \leq T, \\ f'_a(\theta - T), & \theta > T, \end{cases}$$

for all $a \in A$, $\theta \in \{0, \dots, T + T'\}$.

Again, the resulting flow does not need to be feasible. Furthermore, if both flows f, f' are temporally repeated (resp. uniform) this does not necessarily imply that the resulting flow is also temporally repeated (resp. uniform). If the flow is feasible, we call the two flows f, f' *subflows*.

In the rest of this section, we define k-temporally repeated and k-uniform flows which combine multiple temporally repeated (resp. uniform) flows. If we use the previously defined combination, then the sum of the time horizons of the individual flows is the time horizon of the combined flow. However, we generally search for quick flows (hence flows with minimal time horizon, see Chapter 3) and therefore we start by identifying how to combine two temporally repeated (resp. uniform) flows more efficiently.

For two flows over time, we want to start the second flow as early as possible. Hence, we want to find the earliest point in time such that the capacity constraint remains fulfilled. Since we also want to combine multiple flows, it is convenient to specify an earliest starting time for the second flow and ensure that an eventually added third flow cannot overlap with the first flow. Last but not least, we actually want to combine the two flows one after the other, hence we specify that the second flow can start only during the last phase of the first flow (when units are not sent along all paths anymore). To that end, we define a function that calculates at what time the second flow should start (at latest after the end of the first flow, possibly a bit earlier).

Definition 2.22 (Delay)**.** *Let f, f' be temporally repeated resp. uniform flows over time over the same network (G, u, τ). Suppose that f has time horizon T, the underlying static flow has path decomposition $y_f : P_f \to \mathbb{R}_{\geq 0}$, and $t := \max_{p \in P_f} \tau(p)$ is the maximal path length*

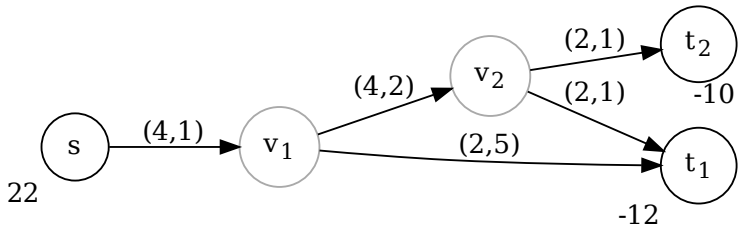

Figure 2.7: Again, we have the same flow over time network (G, u, τ) as in Figure 2.3 with a new balance function b. In contrast to the balance function in Figure 2.4, the demand is now greater than 0 for both nodes t_1 and t_2.

(compare with Definition 2.18). The flow f' has the time horizon T', the path decomposition $y'_f : P'_f \to \mathbb{R}_{\geq 0}$ and the maximal path length $t' := \max_{p \in P_{f'}} \tau(p)$.
Then, we define a function

$$\text{delay}(f, f') := \min\{\, \theta \in \{-t+1, \dots, 0\} \mid T' + \theta > t' \quad \text{and} \\ f_a(T+\rho) + f'_a(\rho - \theta) \leq u(a) \quad \forall a \in A, \theta \leq \rho \leq 0 \,\}.$$

A k-temporally repeated flow consists of k temporally repeated flows that are combined using the time delay as specified above. Take a look at Figure 2.8 for an example.

Definition 2.23 (k-Temporally Repeated Flow)**.** *Let $f^1, \dots, f^k$ be temporally repeated flows over the same network (G, u, τ) with time horizons $T_1, \dots, T_k$. Then, we define $2 \cdot k$ points in time:*

$$^1T := 0 \in \mathbb{N}_0, \qquad T^1 := T_1 \in \mathbb{N}_0,$$
$$^iT := T^{i-1} + \text{delay}(f^{i-1}, f^i) \qquad T^i := {}^iT + T_i \in \mathbb{N}_0, \quad \text{for } 1 < i \leq k.$$

Furthermore, we assume that $P^i_a(\theta)$ is the set $P_a(\theta)$ for the temporally repeated flow f^i (as in Definition 2.16). Then, we define a k-temporally repeated flow *F as a family of functions $f_a : \{0, \dots, T^k\} \to \mathbb{R}_{\geq 0}$, $a \in A$, with*

$$f_a(\theta) := \sum_{\substack{1 \leq i \leq k, \\ \text{if } \theta \in \{{}^iT, \dots, T^i\}}} f^i_a(\theta - {}^iT) \qquad \forall a \in A,\ \theta \in \{0, \dots, T^k\}.$$

Again, we call the flows $f^1, \dots, f^k$ *subflows* of the resulting k-temporally repeated flow f.

Example 2.24. In Figure 2.8, it holds for the first flow f that $t_f = 6$ and for the second flow f' that $t_{f'} = 4$. Furthermore, the time horizons are $T_f = 6$ and $T_{f'} = 8$. To avoid any overlapping with a potential third flow, the inequality $T_{f'} + \theta > t_{f'}$ must be satisfied, hence $\theta > t_{f'} - T_{f'} = 4 - 8 = -4$ and the earliest possible starting time is $T_f - \theta = 6 - 3 = 3$ for $\theta = -3$. Since the capacity constraints are valid for this starting time and $-t_f = -6 \leq -3 \leq 1$, the calculated starting time is indeed 3.

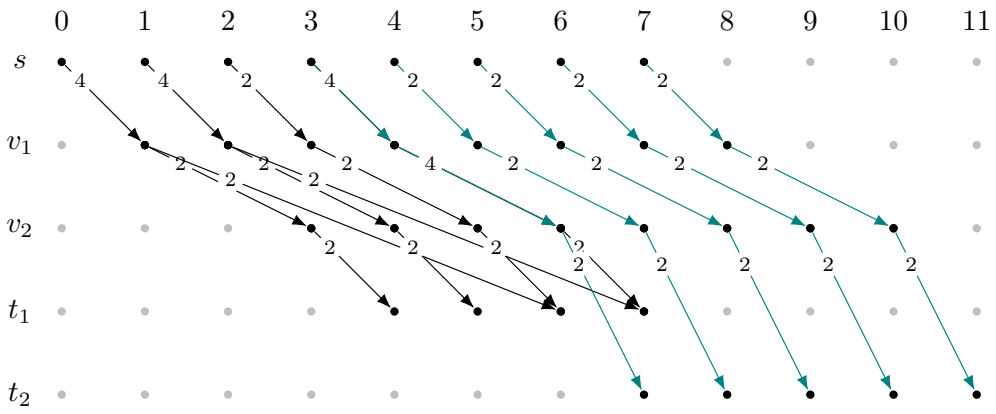

Figure 2.8: A 2-temporally repeated flow for the network and balances given in Figure 2.7 with time horizon 11. The first flow is the temporally repeated flow from Figure 2.5 and the second flow fulfills the additional demand of the sink t_2.

A k-uniform flow is defined similarly to a k-temporally repeated flow. Figure 2.9 contains an example, where we can also see the application of the function delay.

Definition 2.25 (k-Uniform Flow)**.** *Let $f^1, \dots, f^k$ be uniform flows over the same network (G, u, τ) with time horizons $T_1, \dots, T_k$. Then we define $2 \cdot k$ points in time:*

$$
\begin{aligned}
&{}^1T := 0 \in \mathbb{N}_0, && T^1 := T_0 \in \mathbb{N}_0,\\
&{}^iT := T^{i-1} + \operatorname{delay}(f^{i-1}, f^i) && T^i := {}^iT + T_i \in \mathbb{N}_0, \quad \text{for } 1 < i \le k.
\end{aligned}
$$

*Furthermore, we assume that $P_a^i(\theta)$ is the set $P_a(\theta)$ for the uniform flow f^i (as in Definition 2.18). Then, we define a k-*uniform flow *f as a family of functions $f_a : \{0, \dots, T^k\} \to \mathbb{R}_{\ge 0}$, $a \in A$, with*

$$
f_a(\theta) := \sum_{\substack{1 \le i \le k,\\ \text{if } \theta \in \{{}^iT, \dots, T^i\}}} f_a^i(\theta - {}^iT) \qquad \forall a \in A,\ \theta \in \{0, \dots, T^k\}.
$$

The definitions of k-temporally repeated and k-uniform flows enable us to talk about flows over time which have a structure that is more refined than the structure of temporally repeated and uniform flows. Again, the structure implies simplicity and e.g. makes the flows easier to remember by humans. On the other hand, the refined structure might yield better results that the temporally repeated and uniform flows considering the problems given in Chapter 3.

Lemma 2.26. *A k-temporally repeated resp. k-uniform flow f with time horizon T is a feasible flow over time.*

Proof. We show that the constructed k-temporally repeated flow f satisfies the constraints and consider the feasible, temporally repeated flows $f^1, \dots, f^k$ over the same network (G, u, τ).

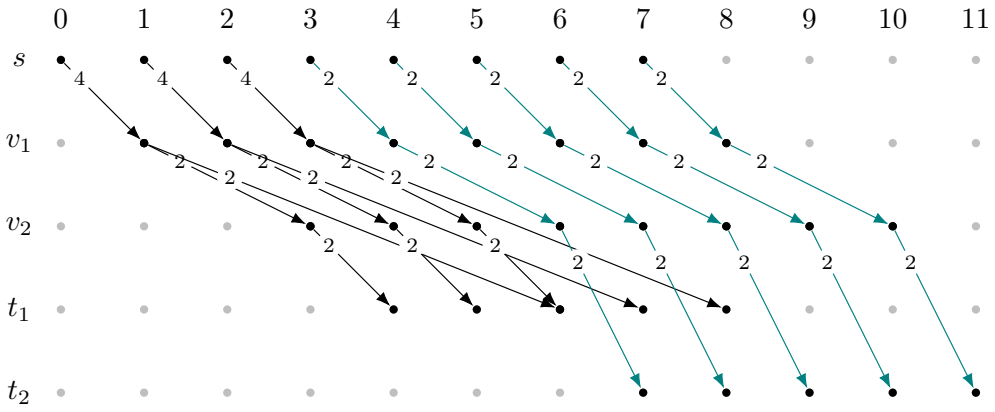

Figure 2.9: A 2-uniform flow for the network and balances given in Figure 2.7 with time horizon 11. The first flow is the uniform flow from Figure 2.6 and the second flow fulfills the additional demand of the sink t_2.

- Each of the flows $f^1, \dots, f^k$ is feasible and satisfies the capacity constraint. Hence, we only need to consider the points in time $\theta \in \{0, \dots, T^k\}$, where $\theta \in \{{}^{i-1}T, \dots, T^{i-1}\} \cap \{{}^{i}T, \dots, T^{i}\}$ for $1 < i \leq k$. Then, it follows that $\theta \in \{T^{i-1} + \text{delay}(f^{i-1}, \dots, f^i), T^{i-1}\}$ and the definition of delay guarantees that the capacity constraint is fulfilled.
- Since f^k is a feasible flow, it holds that $f_a(\theta) = 0$ for all $a \in A$ and $\theta > T - \tau_a$.
- The weak and strict flow conservation follow analogously since they hold for each subflow.

Hence, the k-temporally repeated flow is feasible. The feasibility of the k-uniform flow can by shown similarly. □

2.3 Equivalent Definitions of Flows over Time

In this section, we look at a definition for flows over time with integrals and show that the definition is loosely equivalent to our previous definition of flows over discrete time (see Definition 2.9). We start by defining continuous flows over time for networks with intermediate storage. The following definition is taken from [FS03] and translated such that it uses our notation.

Definition 2.27 (Continuous Flow Over Time)**.** *For a flow over time network (G, u, τ), a time horizon $T \in \mathbb{N}_0$, and a balance function b, a feasible* continuous b-flow over time *is a family of Lebesgue-measurable functions $f_a : \mathbb{R} \to \mathbb{R}_{\geq 0}$, $a \in A$, which satisfies*

1. *the capacity constraint*

$$0 \leq f_a(\theta) \leq u(a) \qquad \forall a \in A, \theta \in [0, T),$$

2. *the flow completion $f_a(\theta) = 0$ for all $a \in A$ and $\theta > T - \tau_a$,*

3. *the weak flow conservation for all* $v \in V \setminus \{s\}$ *and* $\theta \in [0, T)$

$$\int_0^\theta \left(\sum_{a \in \delta^-(v)} f_a(\xi - \tau_a) - \sum_{a \in \delta^+(v)} f_a(\xi) \right) \mathrm{d}\xi \geq 0,$$

4. *and the strict flow conservation for all* $\forall v \in V$

$$\int_0^T \left(\sum_{a \in \delta^-(v)} f_a(\theta - \tau_a) - \sum_{a \in \delta^+(v)} f_a(\theta) \right) \mathrm{d}\theta = -b(v).$$

Furthermore, $f_a(\theta) = 0$ *for* $\theta \notin [0, T)$.

For a flow over time network (G, u, τ)*, a balance function* b *and a time horizon* T*, we define* $\mathcal{F}_{cont}(G, u, \tau, b, T)$ *as the set of all feasible continuous flows over time.*

For a given flow over time network, we can now map each feasible continuous flow over time for a time horizon $T + 1$ to a corresponding feasible discrete flow over time for a time horizon T (as defined in Definition 2.9).

Lemma 2.28. *Let* (G, u, τ) *be a flow over time network,* b *a balance function and* T *a time horizon. Each feasible* continuous *b-flow over time* f *with time horizon* $T + 1$ *can be mapped to a feasible* discrete *b-flow over time* f' *with time horizon* T *via the surjective function*

$$\Theta : \mathcal{F}_{cont}(G, u, \tau, b, T + 1) \to \mathcal{F}(G, u, \tau, b, T),\ f \mapsto f',$$
$$\text{where} \qquad f'_a(\theta) := \int_\theta^{\theta+1} f_a(\xi)\, \mathrm{d}\xi \quad \forall a \in A,\ \theta \in \mathbb{Z}.$$

Proof. We show that the mapping is well-defined by proving that $f' := \Theta(f)$ is a feasible discrete flow over time with time horizon T for any feasible continuous flow over time $f \in \mathcal{F}_{cont}(G, u, \tau, b, T + 1)$. We check the constraints on discrete flows over time.

1. Since f is feasible, the capacity constraint holds and $0 \leq f_a(\rho) \leq u(a)$ for all $a \in A$ and $\rho \in [0, T + 1)$. Hence, for all $a \in A$ and $\theta \in \{0, \ldots, T\}$

$$0 \leq f'_a(\theta) = \int_\theta^{\theta+1} f_a(\xi)\, \mathrm{d}\xi \leq \int_\theta^{\theta+1} u(a)\, \mathrm{d}\xi = u(a)$$

   and the capacity constraint holds for f'.

2. Let $a \in A$ and $\theta \in \{0, \ldots, T\}$ with $\theta > T - \tau_a$. Then $f'_a(\theta) = \int_\theta^{\theta+1} f_a(\xi)\, \mathrm{d}\xi = 0$ since $f_a(\rho) = 0$ for $\rho \in [0, T + 1)$ with $\rho \geq \theta \geq T - \tau_a$.

3. The weak flow conservation for continuous flows over time implies for all $v \in V \setminus \{s\}$ and $\theta \in \{0, \dots, T\}$ that

$$\begin{aligned}
0 &\leq \int_0^{\theta+1} \left(\sum_{a \in \delta^-(v)} f_a(\xi - \tau_a) - \sum_{a \in \delta^+(v)} f_a(\xi) \right) \mathrm{d}\xi \\
&= \sum_{a \in \delta^-(v)} \int_0^{\theta+1} f_a(\xi - \tau_a) \, \mathrm{d}\xi - \sum_{a \in \delta^+(v)} \int_0^{\theta+1} f_a(\xi) \, \mathrm{d}\xi \\
&= \sum_{a \in \delta^-(v)} \sum_{i=0}^{\theta} \int_i^{i+1} f_a(\xi - \tau_a) \, \mathrm{d}\xi - \sum_{a \in \delta^+(v)} \sum_{i=0}^{\theta} \int_i^{i+1} f_a(\xi) \, \mathrm{d}\xi \\
&= \sum_{a \in \delta^-(v)} \sum_{i=0}^{\theta-\tau_a} \int_i^{i+1} f_a(\xi) \, \mathrm{d}\xi - \sum_{a \in \delta^+(v)} \sum_{i=0}^{\theta} \int_i^{i+1} f_a(\xi) \, \mathrm{d}\xi \\
&= \sum_{a \in \delta^-(v)} \sum_{i=0}^{\theta-\tau_a} f'_a(i) - \sum_{a \in \delta^+(v)} \sum_{i=0}^{\theta} f'_a(i).
\end{aligned}$$

   Thus, the weak flow conservation holds for the discrete flow over time f'.

4. Lastly, we show that the strict flow conservation for the continuous flow over time f implies the analogous constraint for the discrete flow over time f'. It holds that

$$\begin{aligned}
-b(v) &= \int_0^{T+1} \left(\sum_{a \in \delta^-(v)} f_a(\theta - \tau_a) - \sum_{a \in \delta^+(v)} f_a(\theta) \right) \mathrm{d}\theta \\
&= \sum_{a \in \delta^-(v)} \int_0^{T+1} f_a(\theta - \tau_a) \, \mathrm{d}\theta - \sum_{a \in \delta^+(v)} \int_0^{T+1} f_a(\theta) \, \mathrm{d}\theta \\
&= \sum_{a \in \delta^-(v)} \sum_{i=0}^{T} \int_i^{i+1} f_a(\theta - \tau_a) \, \mathrm{d}\theta - \sum_{a \in \delta^+(v)} \sum_{i=0}^{T} \int_i^{i+1} f_a(\theta) \, \mathrm{d}\theta \\
&= \sum_{a \in \delta^-(v)} \sum_{i=0}^{T-\tau_a} \int_i^{i+1} f_a(\theta) \, \mathrm{d}\theta - \sum_{a \in \delta^+(v)} \sum_{i=0}^{T} \int_i^{i+1} f_a(\theta) \, \mathrm{d}\theta \\
&= \sum_{a \in \delta^-(v)} \sum_{i=0}^{T-\tau_a} f'_a(i) - \sum_{a \in \delta^+(v)} \sum_{i=0}^{T} f'_a(i)
\end{aligned}$$

   for $v \in V$.

It holds that $f'_a(\theta) = 0$ for $\theta \in \mathbb{Z} \setminus \{0, \dots, T\}$. Hence, we have shown that f' is a feasible discrete flow over time for the time horizon T and the function Θ is well-defined.

Let $f' \in \mathcal{F}(G, u, \tau, b, T)$ be a discrete flow over time for the time horizon T. Then, we can

construct a continuous flow over time f for the time horizon $T+1$ as

$$f_a(\rho) := f'_a(\lfloor \rho \rfloor) \qquad \forall a \in A,\ \rho \in [0, T+1),$$

where $\lfloor \rho \rfloor \in \mathbb{N}_0$ is the largest integer value smaller than ρ. This flow over time is feasible since the Lebesgue integral is well-defined for piecewise linear functions [Fol99]. Furthermore, $\Theta(f) = f'$ and thus Θ is surjective. □

3 Quickest Transshipment and Related Problems

In this chapter, we state common problems on static flows and flows over time such as the Min Cost Flow Problem, the Max Flow over Time Problem and the Quickest Transshipment Problem. Then, we show the special relation between the Quickest Transshipment Problem and the Min Cost Flow over Time Problem and reduce the first problem onto the second one. Last but not least, we analyse which problems have integer optimal solutions and which do not. We observe that most problems are not integer anymore if the set of feasible solutions is restricted to temporally repeated or uniform flows.

3.1 Problem Formulations

On the next pages, we give an overview of the common questions and problems that arise while analyzing different kinds of flows.

Considering static flows, we may be interested in the maximal amount of flow that can be sent from one node to another node in the network.

Definition 3.1 (Max Flow Problem, [FJF62])**.** *Given*

- *a flow network* (G,u)*,*
- *a source* $s \in V$ *and a sink* $t \in V$*,*

find a balance function b *and a static* b*-flow* x *such that* $b(v) = 0$ *for all* $v \in V \setminus \{s,t\}$ *and* $\frac{1}{2}\sum_{v\in V} b(v)$ *is maximal.*

Furthermore, we might specify the supply and demand of all nodes via a balance function and search for a flow that fulfills the balance function with minimal costs.

Definition 3.2 (Min Cost Flow Problem, [FJF62])**.** *Given*

- *a flow network* (G,u,c)
- *and a balance function* b*,*

find a static b*-flow* x *with minimal costs* $c(x) := \sum_{a\in A} c(a)\cdot x(a)$*.*

On flow over time networks, we can consider the previous problems in an adapted form. We may be interested in the maximal amount of flow that can be sent from one node to another node in the network over a given time horizon.

Definition 3.3 (Max Flow over Time Problem, [FJF58])**.** *Given*

- *a flow over time network* (G, u, τ)*,*
- *a source* $s \in V$ *and a sink* $t \in V$*,*
- *and a time horizon* $T \in \mathbb{N}_0$*,*

find a balance function b *and a* b*-flow over time* f *with time horizon* T *such that* $b(v) = 0$ *for all* $v \in V \setminus \{s, t\}$ *and* $\frac{1}{2} \sum_{v \in V} b(v)$ *is maximal.*

Example 3.4. If a flow over time network models the pathways in a hospital, then the *Max Flow over Time Problem* can be used to analyze the maximal amount of beds that may be transported from the main depot to a station in a given time horizon.

Also, we might specify the supply and demand of all nodes via a balance function and search for a flow that fulfills the balance function with minimal costs over a given time horizon.

Definition 3.5 (Min Cost Flow over Time Problem, [KW04])**.** *Given*

- *a flow over time network with costs* (G, u, τ, c)*,*
- *a balance function* b*,*
- *and a time horizon* $T \in \mathbb{N}_0$*,*

find a b*-flow over time* f *with time horizon* T *and minimal costs*

$$c(f) := \sum_{a \in A} \sum_{\theta=0}^{T} c(a) \cdot f_a(\theta).$$

Example 3.6. Given a flow over time network as in the previous example, we define the cost function c such that it equals the transit times τ for all arcs. Then, the result of the *Min Cost Flow over Time Problem* for a balance function and a time horizon minimizes the total transit times for all beds. If each bed has to be pushed by the same number of persons, then this minimizes the total working time.

An earliest arrival flow satisfies the maximum possible demand at every point in time. This implies that flow arrives as early as possible at the sinks in the network. The problem is inspired by evacuation problems, where as many people as possible should reach exit points (e.g. the exits of a building) as early as possible.

Definition 3.7 (Earliest Arrival Flow over Time Problem, [SS14])**.** *Given*

- *a flow over time network* (G, u, τ)*,*
- *a balance function* b*,*
- *and a time horizon* $T \in \mathbb{N}_0$*,*

find a b*-flow over time* f *with time horizon* T *such that* $\text{value}(f(\theta))$ *is maximal for every point in time* $\theta \in \{0, \dots, T\}$*.*

A related problem is the *Quickest Transshipment Problem*, which aims to find a flow over time that satisfies a given balance function as fast as possible. Hence, the overall timespan is minimized.

Definition 3.8 (Quickest Transshipment Problem, [Fle01])**.** *Given*

- *a flow over time network* (G, u, τ)*,*
- *a balance function* b*,*
- *and a time horizon* $T \in \mathbb{N}_0$*,*

find a b*-flow over time* f *with time horizon* T *such that* $\theta \in \{0, \dots, T\}$ *with* $\text{value}(f(\theta)) = \text{value}(f)$ *is minimal.*

A related problem is the *Quickest Flow Problem*, which resembles the *Quickest Transshipment Problem* but considers only one source and one sink, see [Fle01].

Example 3.9. Again, we consider a flow over time network which models a hospital. We define a balance function such that it models the demand and supply of the main depot and the stations. Then, the result of the *Quickest Transshipment Problem* minimizes the overall time which is needed to transport all beds from the main depot to the station. This could be helpful to minimize the time that the pathways need to be blocked for other processes.

All of the problems that we consider in the rest of this work resemble the problems that we stated in this section. We might further specify the type of the solution (e.g. consider only temporally repeated flows) or consider combinations of the problems (e.g. finding a maximum flow with minimal costs), but the problems are always related to already defined problems.

3.2 Reduction of the Quickest Transshipment to Min Cost Flows over Time

In this section, we show that the Quickest Transshipment Problem can be reduced to the Min Cost Flow Over Time Problem. This implies that finding an efficient algorithm for solving the Min Cost Flow Over Time Problem enables us to efficiently calculate solutions of the Quickest Transshipment Problem.

Theorem 3.10. *There exists a Turing reduction from the Quickest Transshipment Problem to the Min Cost Flow Over Time Problem.*

Proof. Let N be an instance of the Quickest Transshipment Problem consisting of a flow over time network (G, u, τ), a balance function b, and a time horizon $T \in \mathbb{N}_0$. We aim to find a b-flow over time f such that all demands are fulfilled as early as possible.

Calculate a solution. We define an instance $N_{T'}$ of the Min Cost Flow Over Time Problem by specifying a cost function $c : V \to \mathbb{N}_0, v \mapsto 0$ and a flow over time network with costs (G, u, τ, c). Additionally, the same balance function b is used and the time horizon is indicated by $T' \in \mathbb{N}_0$.

We calculate an upper bound $T_{\max} \in \mathbb{N}_0$ for the time horizon of the optimal solution of the instance N. Therefore, we compute the shortest paths $p_1, \dots, p_h$ (concerning the transit time) from the source $s \in V$ to each sink $t_i \in V$, $1 \le i \le h$. The sum of the transit times for all those paths $\sum_{1 \le i \le h} \tau(p_i)$ multiplied by the number of demand $\frac{1}{2} \sum_{v \in V} |b(v)|$ yields such an upper bound $T_{\max}$.

Now, we can check for every time horizon $T' \in \{0, \dots, T_{\max}\}$ whether a solution for the instance $N_{T'}$ of the Min Cost Flow Over Time Problem exists. Let $T^* \in \{0, \dots, T_{\max}\}$ be minimal such that there exists a solution for the instance N_{T^*}. Then, the solution of N_{T^*} is an optimal solution for the instance N of the Quickest Transshipment Problem.

The solution is optimal. Any solution of the instance N_{T^*} is a b-flow over time f^* over time horizon T^*. Hence, it is also a feasible solution of the instance N.

Assume that the solution is not optimal. Then, there exists a b-flow over time f with a smaller time horizon $T < T^*$. Hence, the flow f is also a solution of the instance N_T and we have a contradiction. □

If there exists a solution for the instance N_T, $T \in \{0, \dots, T_{\max}\}$, then every instance $N_{T'}$ with $T' \ge T$ also has a solution. Hence, we can find the smallest time horizon $T^* \in \mathbb{N}_0$ via a binary search through the interval $[0, T_{\max}]$ by checking whether there exist solutions for the respective instances.

A different perspective on the reduction The Min Cost Flow Over Time Problem can be written as the linear program $LP_{\text{Min Cost}}$

$$
\begin{aligned}
\min \quad & \sum_{a \in A} \sum_{\theta=0}^{T} c_a f_a^\theta & & \\
\text{s.t.} \quad & 0 \le f_a^\theta \le u_a & & a \in A, \theta \in \{0, \dots, T\}, \\
& f_a^\theta = 0 & & a \in a, \theta \in \{T - \tau_a + 1, \dots, T\}, \\
& \sum_{a \in \delta^-(v)} \sum_{\xi=0}^{\theta-\tau_a} f_a^\xi - \sum_{a \in \delta^+(v)} \sum_{\xi=0}^{\theta} f_a^\xi \ge 0 & & v \in V \setminus \{s\}, \theta \in \{0, \dots, T\}, \\
& \sum_{a \in \delta^-(v)} \sum_{\xi=0}^{T-\tau_a} f_a^\xi - \sum_{a \in \delta^+(v)} \sum_{\xi=0}^{T} f_a^\xi = -b(v) & & v \in V, \\
& f_a^\theta \in \mathbb{Z}_{\ge 0} & & a \in A, \theta \in \{0, \dots, T\},
\end{aligned}
$$

for a given time horizon $T \in \mathbb{N}_0$. The flow over time f which solves the problem is fully specified via the decision variables f_a^θ for $a \in A$ and $\theta \in \{0, \dots, T\}$. The constraints guarantee that the solution is a feasible b-flow over time.

Furthermore, the following linear program LP_{Quickest} represents the Quickest Transshipment Problem:

$$
\begin{aligned}
\min \quad & \sum_{\theta=0}^{T} y_\theta \\
\text{s.t.} \quad & 0 \le f_a^\theta \le u_a && a \in A, \theta \in \{0, \dots, T\}, \\
& f_a^\theta = 0 && a \in A, \theta \in \{T - \tau_a + 1, \dots, T\}, \\
& \sum_{a \in \delta^-(v)} \sum_{\xi=0}^{\theta-\tau_a} f_a^\xi - \sum_{a \in \delta^+(v)} \sum_{\xi=0}^{\theta} f_a^\xi \ge 0 && v \in V \setminus \{s\}, \theta \in \{0, \dots, T\}, \\
& \sum_{a \in \delta^-(v)} \sum_{\xi=0}^{T-\tau_a} f_a^\xi - \sum_{a \in \delta^+(v)} \sum_{\xi=0}^{T} f_a^\xi = -b(v) && v \in V, \\
& y_\theta \ge y_{\theta+1} && \theta \in \{0, \dots, T\}, \\
& f_a^\theta \le M \cdot y_\theta && a \in A, \theta \in \{0, \dots, T\}, \\
& f_a^\theta \in \mathbb{Z}_{\ge 0} && a \in A, \theta \in \{0, \dots, T\}, \\
& y_\theta \in \{0, 1\} && \theta \in \{0, \dots, T\}.
\end{aligned}
$$

Here, $M \in \mathbb{N}_0$ is a very large number and might e.g. be set to $\frac{1}{2} \sum_{v \in V} |b(v)|$. Via the decision variables y_θ, $\theta \in [0, T]$, we ensure that there is no flow after a certain point in time. This point is as early as possible due to the objective function.

A different way to prove Theorem 3.10 is by looking at the LP formulations of the two problems. We make the following observation: Suppose that we fix the decision variables y_θ, $\theta \in [0, T]$, such that there exists a point in time $T^* \in \mathbb{N}_0$ with $y_\theta = 1$ for $\theta \le T^*$ and $y_\theta = 0$

for $\theta > T^*$. Then, the linear program LP_{Quickest} is reduced to

$$
\begin{aligned}
\min \quad & T^* \\
\text{s.t.} \quad & 0 \le f_a^\theta \le u_a && a \in A, \theta \in \{0,\dots,T\}, \\
& f_a^\theta = 0 && a \in A, \theta \in \{T-\tau_a+1,\dots,T\} \\
& \sum_{a\in\delta^-(v)} \sum_{\xi=0}^{\theta-\tau_a} f_a^\xi - \sum_{a\in\delta^+(v)} \sum_{\xi=0}^{\theta} f_a^\xi \ge 0 && v \in V\setminus\{s\}, \theta \in \{0,\dots,T\}, \\
& \sum_{a\in\delta^-(v)} \sum_{\xi=0}^{T-\tau_a} f_a^\xi - \sum_{a\in\delta^+(v)} \sum_{\xi=0}^{T} f_a^\xi = -b(v) && v \in V, \\
& f_a^\theta = 0 && \theta \in \{T^*+1,\dots,T\}, \\
& f_a^\theta \in \mathbb{Z}_{\ge 0} && a \in A, \theta \in \{0,\dots,T\}.
\end{aligned}
$$

Since the variable T^* is fixed, this new formulation resembles the $LP_{\text{Min Cost}}$ with the new time horizon T^* and costs set to 0 (where an arbitrary but fixed number is added to the objective function). Hence, a solution for the linear program LP_{Quickest} with timespan T^* exists, if a feasible solution for the corresponding $LP_{\text{Min Cost}}$ with time horizon T^* exists. This can be checked for any $T^* \in \{0,\dots,T\}$.

3.3 Integrality

In this section, we want to analyze the solution sets of the proposed problems and establish whether there exist integer optimal solutions. For now, we consider networks with a *single source and a single sink*. We give the following definition to explain the underlying theory.

Definition 3.11 (Totally Unimodular Matix)**.** *A matrix $A \in \mathbb{Z}^{n\times m}$ is* totally unimodular *if every square submatrix of A has determinant $-1, 0,$ or 1.*

We state the following fundamental result:

Theorem 3.12 ([Sch98])**.** *If $A \in \mathbb{Z}^{n\times m}$ is a totally unimodular matrix and $b \in \mathbb{Z}^m$ an integer vector then the polyhedron $P := \{x \mid Ax \le b\}$ is integer.*

A polyhedron is integer if all corner points are integer, meaning that for any objective function represented by the vector $b \in \mathbb{Z}^m$, there exists an integer optimal solution.

It is well-known that static flow networks with integer capacities and costs have integer optimal solutions for objective functions represented by integer vectors (since the matrices of the corresponding linear programs are totally unimodular, [Sch98]).

Theorem 3.13 ([Sch98])**.** *The Max Flow Problem and the Min Cost Flow Problem are integer.*

Hence, for both problems there always exists an integer optimal solution.

Suppose that we are given a flow over time network (G, u, τ, c). Due to the definitions in Chapter 2, the capacities, transit times, and costs are always integer. We can construct a time-expanded flow network and solve the given problem on this larger network. A time-expanded flow network contains $T + 1$ copies of the original network into layers. The time progress is represented via moving from a lower layer to an upper layer.

Definition 3.14 (Time-Expanded Flow Network, [Sku09])**.** *Given a flow over time network (G, u, τ) and a time horizon T, we define a so-called* time-expanded flow network (G^T, u^T) *in the following way:*

- *We specify the graph $G^T := (V^T, E^T)$ as*

$$V^T := \{v_\theta \mid v \in V,\ \theta = \{0, \dots, T\}\}.$$

  Now we have a copy of every node for each discrete point in time. Next, we define hold-over arcs (allowing the flow to stay at a node for several points in time) and transition arcs (representing the transition from one node to another node with the corresponding transit time) as

$$A^T := \{(v_\theta, v_{\theta+1}) \mid \theta \in \{0, \dots, T-1\}\} \cup \{(v_\theta, w_{\theta+\tau(v,w)}) \mid (v, w) \in A\}.$$

- *We define a capacity function $u^T : A^T \to \mathbb{N}_0 \cup \infty$ with*

$$(v_\theta, v_{\theta+1}) \mapsto \infty \text{ for } v \in V,\ \theta \in \{0, \dots, T-1\} \quad \text{and} \quad (v_\theta, w_{\theta+\tau(v,w)}) \mapsto u(v, w) \text{ for } (v, w) \in A.$$

If a cost function $c : A \to \mathbb{N}_0$ is given, then we define a corresponding cost function $c^T : A^T \to \mathbb{N}_0$ with

$$(v_\theta, v_{\theta+1}) \mapsto 0 \text{ for } v \in V,\ \theta \in \{0, \dots, T-1\} \quad \text{and} \quad (v_\theta, w_{\theta+\tau(v,w)}) \mapsto c(v, w) \text{ for } (v, w) \in A.$$

Time-expanded flow networks are static networks and hence the Max Flow and Min Cost Flow Problems on those networks are integer for integer capacities and costs.

Theorem 3.15 ([Sku09])**.** *Let (G, u, τ) be a flow over time network, T a time horizon and (G^T, u^T) the corresponding time-expanded network. For every flow over time f in (G, u, τ), there exists a corresponding static flow x in (G^T, u^T) and vice versa. It holds that $c(f) = c(x)$ and* value(f) = value(x).

The transformation from a flow over time f into a static flow x in the time-expanded network is very straightforward: If a unit of flow transitions via an arc $(v, w) \in A$ in the flow over time network, then it transitions via the associated transition arc in the time-expanded network. If a unit of flow stays at a node $v \in V$ in the flow over time network for a time unit, then it transitions via a hold-over arc in the time-expanded network. The formal proof of the previous theorem can be found in [Sku09].

Corollary 3.16. *A flow f in (G,u,τ) is maximal if and only if the corresponding flow x in (G^T,u^T) is maximal.*

Corollary 3.17. *Let (G,u,τ) be a network, $s \in V$ a source, $t \in V$ a sink, and b a balance function such that $b(v) = 0$ for all $v \in V \setminus \{s,t\}$. Furthermore, we have (G^T,u^T), a source $s_0 \in V$, a sink $t_T \in V$, and a balance function b^T with $b^T(v) = 0$ for all $v \in V^T \setminus \{s_0,t_T\}$.*

Then, a b-flow f in (G,u,τ) has minimal costs if and only if the corresponding b^T-flow x in (G^T,u^T) has minimal costs.

The statements in the previous both corollaries follow directly via Theorem 3.15.

Now, we show that the dynamic versions of the maximal flow and minimal cost flow problems are also integer.

Theorem 3.18. *Let (G,u,τ) be a flow over time network with a time horizon T. Then, the Max Flow over Time Problem is integer.*

Proof. We can find a solution of the maximal flow problem on the associated time-expanded network (G^T,u^T). Since the maximal flow problem is integer for static networks, there exists an integer optimal solution in the (G^T,u^T) and hence also an integer optimal flow over time. □

Theorem 3.19. *Let (G,u,τ) be a flow over time network with a time horizon T. Then, the Min Cost Flow over Time Problem is integer.*

Proof. The proof is similar to the proof of Theorem 3.18. □

L.R. Ford and D.R. Fulkerson have shown that the Max Flow over Time Problem over any network with a single source and a single sink has a temporally repeated optimal solution [FJF58]. We derive the following statement.

Theorem 3.20. *Let (G,u,τ) be a flow over time network with a time horizon T, a source $s \in V$, and a sink $t \in V$. Then, the problem of finding a temporally repeated flow with maximal value is integer.*

Proof. Let (G,u,τ) be a flow over time network, T a time horizon, and $s,t \in V$ the source and the sink. Then, we can calculate an optimal solution of the Max Flow over Time Problem via the algorithm stated in [FJF58]. It calculates a temporally repeated flow with maximal value and hence, the theorem is correct. □

If we consider the general problem of finding a maximal temporally repeated flow with multiple sinks, then this problem is not integer.

Theorem 3.21. *Let (G,u,τ) be a flow over time network with a time horizon T, a source $s \in V$, and a list of sinks $t_1,\ldots,t_h \in V$. Then, the problem of finding a maximal temporally repeated flow, which sends the flow to more than one sink, is* not *integer.*

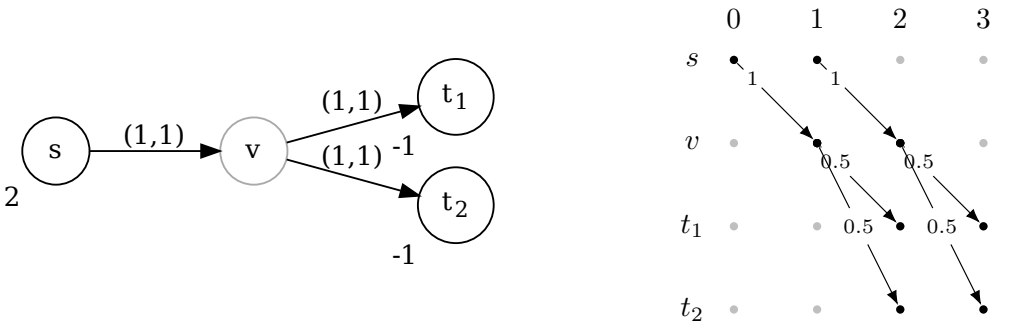

Figure 3.1: A flow network (G, u, τ) with labels (u, τ) and balances b, and the only maximal temporally repeated flow for time horizon $T = 3$. The flow is not integer.

Proof. Assuming that the problem is integer, there would exist an integer temporally repeated flow for the flow over time network (G, u, τ) described in Figure 3.1 which is also a maximal temporally repeated flow. But for the given time horizon 3, there exists exactly one temporally repeated solution. The solution is not integer and hence the problem is also not integer. □

Furthermore, the following problems are also not integer.

Corollary 3.22. *For a flow over time network (G, u, τ) with a time horizon T, the following problems are also* not *integer:*

- *The Min Cost Flow Problem for temporally repeated flows.*
- *The Quickest Transshipment Problem for temporally repeated flows.*
- *The Max Flow Problem, the Min Cost Problem, and the Quickest Transshipment Problem for uniform flows.*

Proof.

- Assume that the network in Figure 3.1 is extended by a cost function c and the costs resemble the transit times. For the given demand and time horizon $T = 3$, the only temporally repeated solution is the flow in Figure 3.1. Hence, the problem of finding a minimal cost temporally repeated flow is not integer.
- The quickest transshipment problem can be reduced to finding the smallest time horizon T such that the maximal flow satisfies the balances. For the network in Figure 3.1, the smallest time horizon is $T = 3$, which has only one solution. The solution is not integer and hence this is a counter-example.
- In all of the above cases, we can replace the notion of temporally repeated flows by uniform flows and the equivalent results follow.

Thus, the common problems are not integer if we restrict the set of feasible solutions to temporally repeated or uniform flows. □

4 Quickest Transshipment on Trees

In this chapter, we discuss flows over time on networks (G, u, τ), where the underlying graph is a tree. The root node $s \in V$ is the only source and the leaves $t_1, \ldots, t_h \in V$ are the only sinks. Figure 4.1 contains an example of such a *tree network*.

In the following, we analyze different approaches to solve the following problem:

Definition 4.1 (Quickest Transshipment On Trees)**.** *Given*

- *a tree network* (G, u, τ)*,*
- *and a balance function b,*

find an integer k-uniform flow with arbitrary $k \leq h$ *which satisfies the balances and has minimal overall time horizon* $T \in N$*.*

In the first section of this chapter, we look at an algorithm that creates an h-uniform flow where every subflow fills exactly one sink. We enhance the algorithm such that it finds better solutions containing subflows that might fill more than one sink.

Then, we define *almost-binary tree networks* as networks where the underlying tree graph has a certain structure. We show that each tree network can be transformed into an equivalent almost-binary tree network. This justifies the focus on almost-binary tree networks in subsequent sections.

Next, we analyze small almost-binary tree networks and calculate optimal solutions for the *Quickest Transshipment on Trees Problem* on those networks. Additionally, we give an intuition on why the calculation of optimal solutions for larger tree networks is hard.

Last but not least, we give an integer linear program for finding an optimal uniform flow and consider its linear relaxation. Then, we state an algorithm that computes an integer k-uniform solution of the problem defined above from the solution of the relaxed linear program.

4.1 Naive Algorithm

Since the network is a tree, a feasible, but usually non-optimal solution would be an h-uniform subflow, where each uniform flow serves the demand of exactly one sink. For the network in Figure 4.1, we could start by filling sink t_3, then fill sink t_2 and end with sink t_1. A better solution would be to start filling the sink with the longest path first, then the sink with the second-longest path, etc. This way, the path length of the long paths affects the

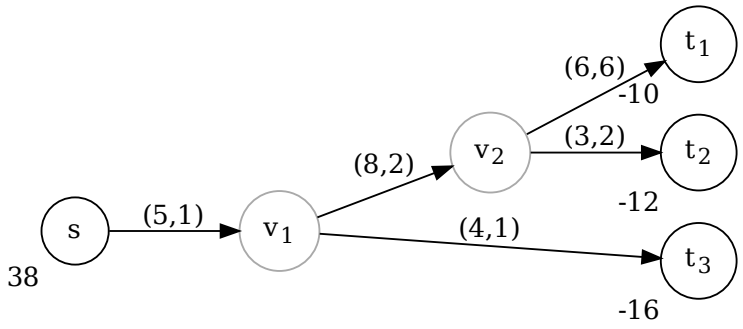

Figure 4.1: A tree network (G,u,τ) where the labels represent (u,τ), and a balance function b. Here, $p_1 = (s,v_1,v_2,t_1)$ is the path from the source s to the sink t_1. It has length $\tau(p_1) = 9$ and $\mathbf{u}(1) = 5$.

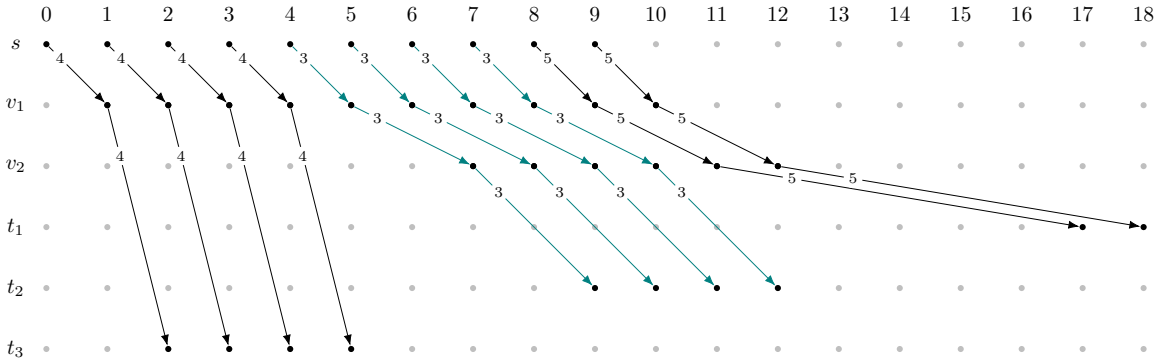

Figure 4.2: A feasible solution for the Quickest Transshipment Problem on the network in Figure 4.1. The solution additionally satisfies the constraint that each subflow fills exactly one sink. It starts by filling the sink with the shortest path.

overall time horizon as little as possible. Both possible solutions are depicted in Figure 4.2 and Figure 4.3.

Listing 4.1 contains a formalization of the described algorithm. There, p_i represents the unambiguous path from the source s to the sink t_i, $i \in \{1,\dots,h\}$. The algorithm computes the path length to each sink, orders them in decreasing order and fills them in the obtained order using uniform flows. We show that – under very limiting restrictions to the set of all feasible solutions – the algorithm computes an optimal solution.

Note that in the following, we treat t_i and i as synonyms and often write $b(i)$ instead of $b(t_i)$ and $\tau(i)$ instead of $\tau(p_i)$. Furthermore, we define $\mathbf{u}(i) := \min_{a \in p_i} u(a)$ as the limiting capacity on the path from the source to the sink t_i.

Lemma 4.2. *Let (G,u,τ) be a tree network and b a balance function. Under all h-uniform flows which fulfill the demand of exactly one sink in each subflow, the flow computed by the algorithm in Listing 4.1 has minimal time horizon.*

Proof. We show that the optimal solution is a combination of the quickest uniform flows

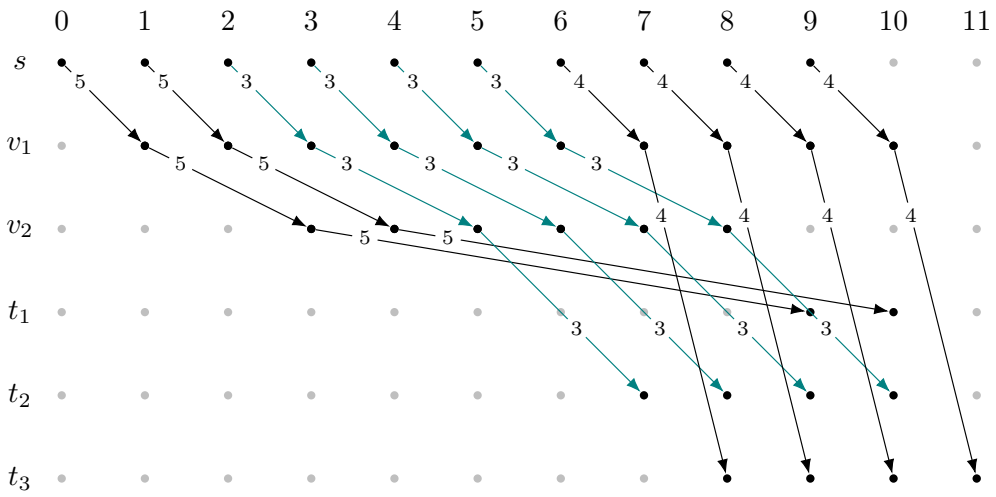

Figure 4.3: Another feasible solution for the Quickest Transshipment Problem on the network in Figure 4.1. This solution also satisfies the constraint that each subflow fills exactly one sink, but it fills the sinks in descending order regarding their path length. This solution has a significantly smaller time horizon than the solution depicted in Figure 4.2.

Listing 4.1: One Sink per Uniform Flow

```
1  Input: Tree network (G,u,τ) and balance function b
2  Output: h-uniform flow f which fulfills
3    exactly one sink per subflow
4
5  Calculate path length τ(p_i) with p_i = (s,...,t_i) for all i ∈ {1,...,h}
6  Sort sinks such that τ(p_1) ≥ τ(p_2) ≥ ... ≥ τ(p_h)
7
8  Construct uniform  flow f_i for all i ∈ {1,...,h}
9    which completely satisfies demand of sink t_i
10   with minimal time horizon
11 Construct flow f by combining f_1,...,f_h
12
13 Return f
```

that fill one sink each. Then, we express the time horizon of such a combination via a formula and show that the time horizon is smallest for the combination calculated in the algorithm.

Let f_i be the uniform flow that fulfills the demand of sink t_i in a minimal time horizon, $i \in \{1,\dots,h\}$. Each possible combination of the uniform flows, where each flow appears exactly once, is a solution of the problem. Since we combine the subflows according to Definition 2.22, the resulting h-uniform flow is feasible. Furthermore, it fulfills the demand given in the balance function b. Since the network has a tree structure, the flow cannot outrun the other flow (there are no cycles in the network) and hence the delay is always exactly one time unit.

Any other solution of the problem has a longer time horizon than the optimal combination of the flows $f_1,\dots,f_h$, since at least one of the subflows must have a non-optimal time horizon and hence the time horizon of the combination is also non-optimal.

Suppose that the sinks $t_1,\dots,t_h$ are ordered such that $\tau(p_1) \geq \tau(p_2) \geq \dots \geq \tau(p_h)$. We indicate the combination of the subflows by the order of the sinks. Let $(i_1,\dots,i_h) \subseteq \{1,\dots,h\}$ be such an ordering which represents a flow. Then, we can calculate the time horizon of this flow by the following formula:

$$T(i_1,\dots,i_h) := \max_{\ell\in\{1,\dots,h\}} \left\{ \sum_{k=1}^{\ell} \frac{-b(i_k)}{\mathbf{u}(i_k)} + \tau(i_\ell) - 1 \right\}.$$

For each subflow f_{i_ℓ}, we calculate the number of iterations for the previously executed subflows $f_{i_1},\dots,f_{i_{\ell-1}}$ and for the current subflow f_{i_ℓ} and add the path length to the current sink $\tau(i_\ell)$ (minus one, since we started at time zero). The maximum is the total time horizon of this combination.

The algorithm in Listing 4.1 calculates the flow represented by the tuple $(1,\dots,h)$.

Claim: The time horizon of the flow $(1,\dots,h)$ *is smaller than the time horizon* $T(i_1,\dots,i_h)$ *for any other combination* $(i_1,\dots,i_h)$.

Suppose that $T(i_1,\dots,i_h) = \sum_{k=1}^{\ell'} \frac{-b(i_k)}{\mathbf{u}(i_k)} + \tau(i_{\ell'}) - 1$ for $\ell' \in \{1,\dots,h\}$. We show that $T(1,\dots,h) \leq T(i_1,\dots,i_h)$ by proving that

$$\sum_{k=1}^{n} \frac{-b(k)}{\mathbf{u}(k)} + \tau(n) - 1 \leq \sum_{k=1}^{\ell'} \frac{-b(i_k)}{\mathbf{u}(i_k)} + \tau(i_{\ell'}) - 1 \quad \text{for all } n \in \{1,\dots,h\}. \tag{4.1}$$

Let $n \in \{1,\dots,h\}$ and $(1,\dots,n) \subseteq (1,\dots,h)$ is a tuple which contains the first n sinks regarding the path length. Furthermore, let $(i_1,\dots,i_\ell) \subseteq (i_1,\dots,i_h)$ be the smallest tuple that contains every element in the tuple $(1,\dots,n)$ (such that $(1,\dots,n) \subseteq (i_1,\dots,i_\ell)$).

> *Small Claim: It holds that* $\tau(n) \leq \tau(i_\ell)$.
>
> Suppose that $\tau(n) > \tau(i_\ell)$. Then, $i_\ell \notin (1,\dots,n)$ is not an element in the tuple and thus the tuple $(i_1,\dots,i_{\ell-1})$ is a smaller tuple which also satisfies that $(1,\dots,n) \subseteq (i_1,\dots,i_{\ell-1})$. Hence, we have a contradiction and the small claim holds.

Now we can show that the time horizon of the flow represented by $(1,\dots,h)$ is indeed the smallest.

- Suppose that $\ell < \ell'$. Then, we derive that

$$\begin{aligned}\sum_{k=1}^{n} \frac{-b(k)}{\mathbf{u}(k)} + \tau(n) - 1 &\leq \sum_{k=1}^{\ell} \frac{-b(i_k)}{\mathbf{u}(i_k)} + \tau(n) - 1 \\ &\leq \sum_{k=1}^{\ell} \frac{-b(i_k)}{\mathbf{u}(i_k)} + \tau(i_\ell) - 1 \\ &\leq \sum_{k=1}^{\ell'} \frac{-b(i_k)}{\mathbf{u}(i_k)} + \tau(i'_\ell) - 1 = T(i_1,\dots,i_h).\end{aligned}$$

  The first inequality holds since $(1,\dots,n)$ is contained in $(i_1,\dots,i_\ell)$ and the fraction is non-negative, the second inequality follows via the small claim, and the third inequality is valid since ℓ' maximizes the term.

- Suppose that $\ell = \ell'$ or $\ell > \ell'$. Then, the proof is analogous to the previous one.

We obtain that the statement in Equation (4.1) is valid for any $n \in \{1,\dots,h\}$ and hence $T(1,\dots,h) \leq T(i_1,\dots,i_h)$. Thus, the algorithm computes a flow following the requirements which has minimal time horizon. □

If we loosen the requirements and specify that we may fulfill the demand of more than one sink per subflow, we can enhance the previous algorithm. Again, we start by filling the sink with the longest path. But now, we additionally send as much flow as possible to the other sinks with long paths. Then, we update the balance function for those sinks and restart by filling the sink with the second longest path (if the balances were not updated to 0). Additionally, we send as much flow as possible to other sinks and resume analogously. Figure 4.4 depicts the solution of this algorithm for the network in Figure 4.1.

The formalization of the enhanced algorithm can be found in Listing 4.2. It uses residual networks for uniform flows.

Definition 4.3 (Residual Network for Uniform Flows)**.** *Let f be a uniform flow on a network (G,u,τ). Then, there exists a static flow x that the flow f is associated to (see Definition 2.18). The residual network $(G,u,\tau)_f$ is the residual network $(G,u)_x$ of the static flow x together with the transit times τ.*

We show that the algorithm computes a solution of the Quickest Transshipment on Trees Problem.

Lemma 4.4. *The algorithm in Listing 4.1 yields a feasible k-uniform flow with $k \leq h$, which fulfills the demand given via the balances b.*

Proof. The computed flow consists of $k \leq h$ distinct uniform subflows, where each is constructed in one iteration of the outer `For` loop. Let us consider one such iteration.

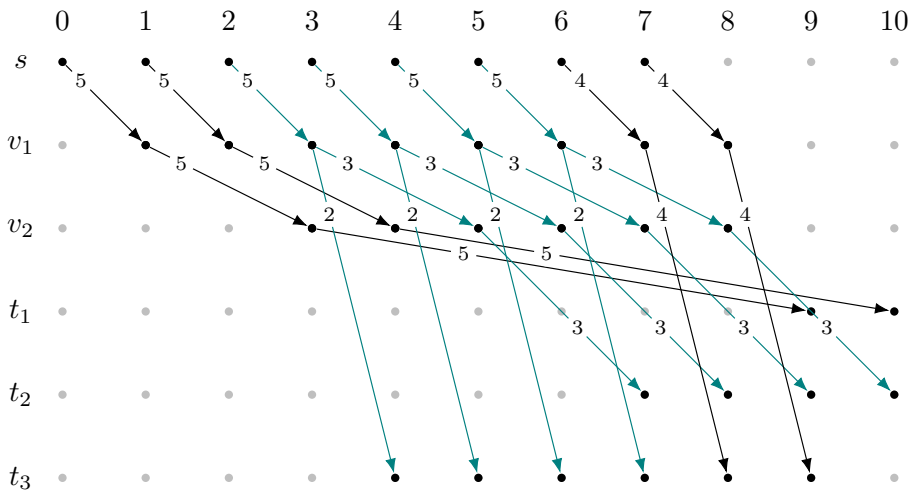

Figure 4.4: Another feasible solution for the Quickest Transshipment Problem given in Figure 4.1. It has a smaller time horizon than the solution given in Figure 4.3 and is a solution of the algorithm in Listing 4.2.

For $i \in \{1,\dots,h\}$ with $b(t_i) \neq 0$, the algorithm constructs a feasible uniform flow f_i which fulfills the demand of sink t_i. This flow is iteratively merged with uniform flows with corresponding time horizons on the (also updated) residual network. This results in a feasible uniform flow, since all flows are uniform, have the same time horizon, and satisfy the capacity constraints (see Definition 2.20).

The combination of the distinct uniform flows yields a feasible k-uniform flow f and hence the solution of the algorithm is feasible. Since there are exactly h iterations of the outer `For` look, there are at most h possible subflows of f, therefore $k \leq h$.

Suppose that there are ℓ subflows $f_1,\dots,f_\ell$ that fill sink t_i. Since the demand $b(t_i)$ is updated at each step, the subflows send exactly the original amount $b(t_i)$ of flow to sink t_i. □

We prove that any solution of the enhanced algorithm is indeed a better solution than the corresponding solution of the simple algorithm.

Lemma 4.5. *For a given network (G,u,τ) and a balance function b, the algorithm in Listing 4.2 yields a solution with time horizon smaller or equal to the solution of the algorithm in Listing 4.1.*

Proof. The proof of Lemma 4.2 states that the algorithm in Listing 4.1 computes a solution with time horizon

$$T_{4.1}(1,\dots,h) = \max_{\ell\in\{1,\dots,h\}} \left\{ \sum_{k=1}^{\ell} \frac{-b(k)}{\mathbf{u}(k)} + \tau(\ell) - 1 \right\}$$

if the sinks are ordered in descending order regarding their path length.

Listing 4.2: Enhanced Algorithm

```
1  Input: Tree network (G,u,τ) and balance function b
2  Output: k-uniform flow f with k ≤ h
3
4  Calculate path length τ(p_i) with p_i = (s,...,t_i) for all i ∈ {1,...,h}
5  Sort sinks such that τ(p_1) ≥ τ(p_2) ≥ ... ≥ τ(p_h)
6
7  Initiate f as empty flow
8
9  For i := 1 to h do:
10   If b(t_i) = 0 then:
11     Set T_i := 0
12   Else:
13     Construct uniform flow f_i
14       which completely fulfills demand of sink t_i
15       with minimal time horizon T_i
16     Create residual network (G,u,τ)_{f_i}
17     Set b(t_i) := 0
18
19     For j := i+1 to h do:
20       If b(t_j) ≠ 0 and p_j = (s,...,t_j) exists in (G,u,τ)_{f_i} then:
21           Construct uniform flow f_i^j
22             which fulfills as much demand of sink t_j as possible
23             but less or equal than b(t_j) (otherwise continue with j+1)
24             in the time horizon T_i
25           Merge f_i := f_i + f_i^j
26           Update residual network (G,u,τ)_{f_i}
27           Update b(t_j)
28
29     Update f by combining f and f_i
30
31 Return f
```

The solution of the algorithm in Listing 4.2 has time horizon

$$T_{4.2}(1,\dots,h) := \max_{\ell\in\{1,\dots,h\}} \left\{ \sum_{k=1}^{\ell} T_k + \tau(\ell) - 1 \right\},$$

since the time is determined by the construction of the uniform subflows in the outer `For` loop (see lines 11 and 15).

Throughout the execution of the algorithm, the balances may get updated. We denote the updated balances as $b'(t_k)$ and know that $0 \geq b'(t_k) \geq b(t_k)$. Hence, $-b'(t_k) \leq -b(t_k)$ and T_k is either 0 (line 11) or $T = \frac{-b'(k)}{\mathbf{u}(k)} \leq \frac{-b(k)}{\mathbf{u}(k)}$. Therefore, we derive that

$$T_{4.2}(1,\dots,h) \leq T_{4.1}(1,\dots,h)$$

and the updated algorithm computes a better (or equally good) solution. □

Figure 4.5 contains an example that shows that the enhanced algorithm does not compute an optimal solution for the problem given in Definition 4.1. But the algorithm helps us to obtain an upper bound for the Quickest Transshipment Problem on Trees. The correlating lower bound is easy to compute.

Lemma 4.6. *For a tree network (G,u,τ) and a balance function b, an upper bound for the Quickest Transshipment Problem on Trees is given by*

$$T_{4.1}(1,\dots,h) := \max_{\ell\in\{1,\dots,h\}} \left\{ \sum_{k=1}^{\ell} \frac{-b(k)}{\mathbf{u}(k)} + \tau(\ell) - 1 \right\}. \tag{4.2}$$

A lower bound is

$$T_{lower}(1,\dots,h) := \max_{\ell\in\{1,\dots,h\}} \left\{ \frac{-b(\ell)}{\mathbf{u}(\ell)} + \tau(\ell) - 1 \right\}. \tag{4.3}$$

Proof. The term in Equation (4.2) is a valid upper bound, since the algorithm in Listing 4.1 computes a solution with such a time horizon.

Suppose that $\ell \in \{1,\dots,h\}$ is a sink in the network. If we consider the network reduced to the source, the sink ℓ, and the path in between, then the optimal solution for the Quickest Transshipment Problem on Trees has time horizon $\frac{-b(\ell)}{\mathbf{u}(\ell)} + \tau(\ell) - 1$. Solving the Quickest Transshipment Problem on the original network, needs at least as much time since the network is a tree and does not include any shortcuts. Hence, the term in Equation (4.3) is a lower bound. □

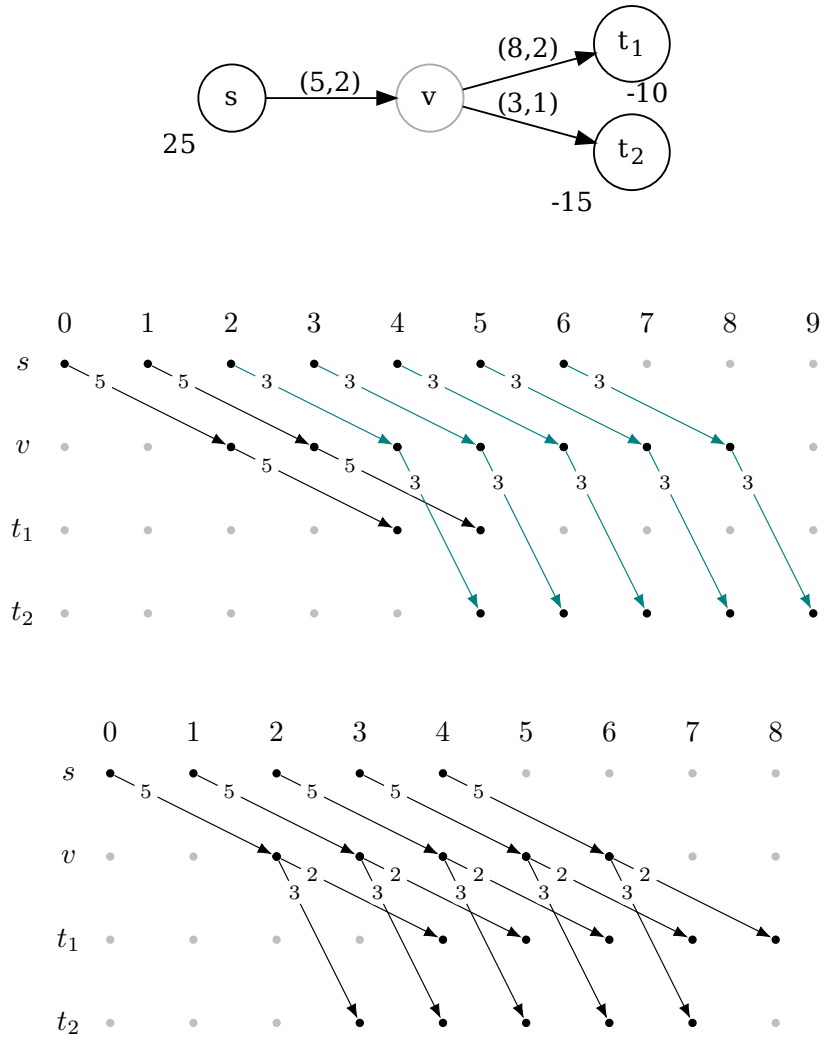

Figure 4.5: A tree network (G, u, τ) where the labels represent (u, τ) with a balance function b. The flow with the longer time horizon is computed by the enhanced algorithm in Listing 4.2 and is not an optimal solution (compare with the quicker flow below).

4.2 Almost-binary Trees

In this section, we analyze tree networks and consider equivalence relations on those networks. This allows us to consider only tree networks with a special structure in subsequent sections.

For that purpose, we define several operations which transform a tree network into an equivalent network. Those operations either delete nodes or add auxiliary nodes. Using those operations, we prove that we can transform every tree network into an equivalent network where the underlying graph is a binary tree. At the end of this section, we define *almost-binary trees* that will be used in the next section.

For the sake of completeness, we consider networks with cost functions. If there does not exist a cost function, the costs can simply be ignored throughout this section.

Given two networks and one flow on each network, we call those flows *equivalent* if the conditions given in Definition 4.7 hold. The equivalence of flows will help us to define the equivalence of tree networks.

Definition 4.7. *Let f be a flow on (G,u,τ,c), f' a flow on (G',u',τ',c') and both networks are tree networks. The two flows f and f' are* equivalent*, denoted by $f \equiv f'$, if*

- *the number of sinks in both networks is equal and we can pair each sink in one network with a sink in the other one, and*
- *for each sink at each point in time the same number of units arrives with the same aggregated costs.*

The aggregated costs are the costs of the path from the source to the sink.

Equivalent flows satisfy the same balance function (maybe with adapted sinks), have the same costs, and have the same overall time horizon. Additionally, the sum of the transit times for all units is also the same. Now, two networks are *equivalent* if for each flow on one network there exists an equivalent flow on the other network.

Definition 4.8. *Two tree networks $(G,u,\tau,c),(G',u',\tau',c')$ are* equivalent*,*

$$(G,u,\tau,c) \equiv (G',u',\tau',c'),$$

if for each flow f on the network (G,u,τ,c) there exists an equivalent flow f' on the other network (G',u',τ',c') and vice versa.

We define several operations that transform a tree network into an equivalent tree network. The first operation allows us to merge a node with its child node if there exists exactly one child. Otherwise, the tree network remains unchanged. The operation takes the network and a particular node in the network as arguments. Compare the definition with Figure 4.6 which depicts how the network is changed.

Definition 4.9 (Merge Nodes)**.** *Let (G,u,τ,c) be a tree network and $v \in G$ a node. Then, we define an operation $\rho_1((G,u,\tau,c),v)$ in the following way:*

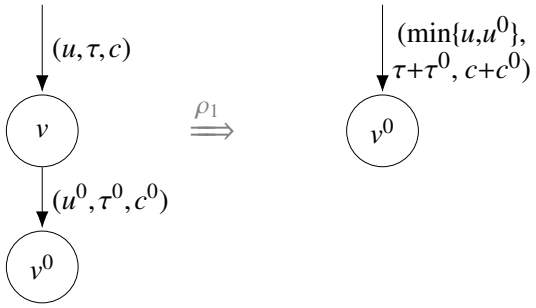

Figure 4.6: Transformation via the operation ρ_1, which merges a node v with its single child and updates the capacities, transit times, and costs on the arcs.

- *If v is a leaf or has at least two children or is the root, then it maps to the network (G,u,τ,c).*
- *Otherwise, it maps to the network (G',u',τ',c') which resembles the network (G,u,τ,c) except that the node v and the arcs (r,v) and (v,v^0) are deleted (where $r \in V$ is the parent of v and v^0 is the child of v) and an arc (r,v^0) is added with*

$$u(r,v^0) := \min\{u(r,v),u(v,v^0)\}, \quad \tau(r,v^0) := \tau(r,v)+\tau(v,v^0),$$
$$\text{and } c(r,v^0) := c(r,v)+c(v,v^0).$$

We show that the previously defined operation transforms a network into an equivalent network.

Lemma 4.10. *For a tree network (G,u,τ,c) and any node $v \in G$, the network $\rho_1((G,u,\tau,c),v)$ is equivalent to the network (G,u,τ,c).*

Proof. If v is a leaf or has more than one child or is the root, then the two networks are equal and hence also equivalent. We assume that this is not the case.

This operation does not change the number of leaves (or sinks) in the network since v has a child node. Hence, we can associate each leaf with the same leaf in the transformed network. We show that the networks (G,u,τ,c) and $(G',u',\tau',c') := \rho_1((G,u,\tau,c),v)$ are equivalent by showing that a flow f on (G,u,τ,c) has an equivalent flow f' on (G',u',τ',c') and vice versa.

Otherwise, we suppose that f is a flow on the tree network (G,u,τ,c). Then, we create a flow f' on the network (G',u',τ',c') which differs from f only on the arcs between r, v, and v^0, where $r \in G$ is the parent and $v^0 \in G$ the child of v. Each flow is sent along arc (r,v^0) instead of along the path (r,v,v^0). This yields a feasible flow since the capacity of the updated arc (r,v^0) is sufficient. Furthermore, the flow f' is equivalent, since the transit time and costs of the updated arc are equal to the transit time and costs of the original path.

For any flow f' on the network (G',u',τ',c'), there also exists an equivalent flow f on (G,u,τ,c) which can be constructed and proven in a similar way. □

The second operation adds two auxiliary nodes if a node has more than two children. The

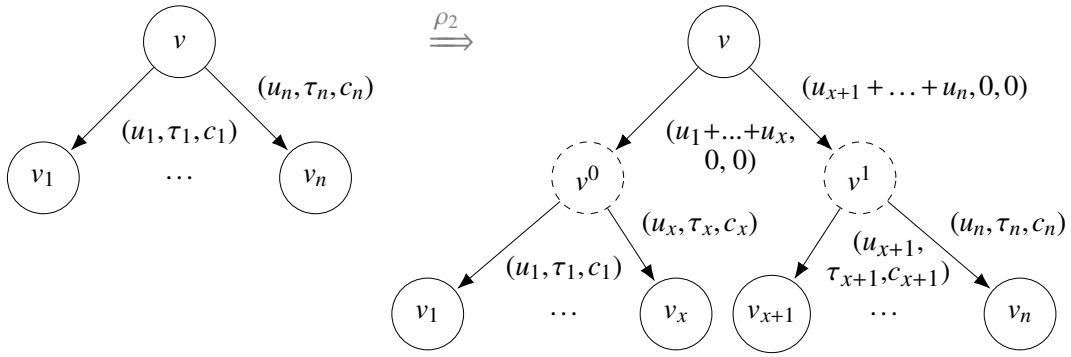

Figure 4.7: Transformation via the operation ρ_2, which reorganizes the $n > 2$ children of a node v via two auxiliary nodes. The operation ensures that the node v has at most two children and adapts the capacities, transit times, and costs. If the node v has at most two children, then the operation does not alter the network.

two auxiliary nodes become the new children of the node and the original children become children of the auxiliary nodes. This increases the height of the subtree and ensures that the particular node has at most two children. Compare the definition with Figure 4.7 which depicts the transformation of a node with more than two children.

Definition 4.11 (Split Children)**.** *Let (G, u, τ, c) be a tree network and $v \in G$ a node. Then, we define an operation $\rho_2((G, u, \tau, c), v)$ which maps to the network (G, u, τ, c) if v has at most two children, otherwise it maps to a network (G', u', τ', c') which resembles (G, u, τ, c) except that*

- *two new auxiliary nodes v^0, v^1 are added, and*
- *two new arcs (v, v^0), (v, v^1) are added with*

$$u(v, v^0) := \sum_{i=1}^{x} u(v, v_i), \quad u(v, v^1) := \sum_{i=x+1}^{n} u(v, v_i),$$
$$\tau(v, v^0) := 0, \tau(v, v^1) := 0, \text{ and } c(v, v^0) := 0, c(v, v^1) := 0,$$

  where $x := \lceil \frac{n}{2} \rceil$ and n is the number of children of v.

- *For each original child v_i of v the arc (v, v_i) is replaced by the arc (v^0, v_i) (if $i \leq x$) or (v^1, v_i) (if $i > x$) with the same capacity, transit time, and costs.*

Let us take a look at how the operations ρ_1 and ρ_2 alter the capacities of the arcs. The first operation sets the capacity to the minimum of both involved original capacities. This is crucial since otherwise, the transformed network might allow more units to flow over an arc than in the original network.

The split operation ρ_2 restricts the capacity of the arcs $(v, v^0), (v, v^1)$ by the sum of the arcs to the children of the auxiliary nodes. However, the capacity could have been an arbitrary larger value, since the flow in the transformed network is already restricted by the arcs from

the auxiliary nodes to their children. For the computation of algorithms on a tree network, it is generally useful to choose values as small as possible, hence we choose the smallest possible capacity such that the network is equivalent to the original network.

Now, we prove that the second operation also preserves the equivalence of tree networks.

Lemma 4.12. *For a tree network (G,u,τ,c) and any node $v \in G$, the network $\rho_2((G,u,\tau,c),v)$ is equivalent to the network (G,u,τ,c).*

Proof. Again, this operation does not change the number of leaves (or sinks) in the network. Hence, we can associate each leaf with the same leaf in the transformed network.

Again, we can show the equivalence of the networks by constructing equivalent flows for both directions. The proof is similar to the proof of Lemma 4.10. □

We can use the previously defined operations to transform any tree network into an equivalent tree network where the underlying graph is a binary tree. A tree is *binary* if each node has at most two children and it is *complete binary* if all nodes except the leaves have exactly two children. We will need complete binary trees in Corollary 4.14 and later.

Theorem 4.13. *Given a tree network (G,u,τ,c), we can construct another tree network (G',u',τ',c'), where G' is a binary tree, by a series of operations ρ_1, ρ_2.*

To prove this theorem, we introduce the recursive algorithm Transformation given in Listing 4.3. For any network (G,u,τ,c) and any node $v \in G$, the execution of the algorithm Transformation$((G,u,\tau,c),v)$ transforms the subgraph belonging to v into a binary tree. Therefore, it analyzes whether the node v has one child, two children or more children. If it has one child and is not the root of the tree G, then the node gets merged with its child. If it has two children, then the algorithm is executed recursively on the children. If it has more than two children, then it creates auxiliary nodes such that v now has two children. Then it executes the algorithm recursively on the newly added auxiliary nodes.

The algorithm uses only the operations ρ_1 and ρ_2. The subsequent proof shows that the resulting network is indeed a binary tree network.

Listing 4.3: Transformation into Binary Tree Network

```
1  Input: Tree network (G,u,τ,c), node v ∈ G
2  Output: An equivalent tree network (G',u',τ',c'),
3    where the subtree belonging to v is a binary tree
4
5  If v has exactly one child v_1 and is not the root:
6    Update (G,u,τ,c) := ρ_1((G,u,τ,c),v)
7    Set (G,u,τ,c) := Transformation((G,u,τ,c),v_1)
8  Else if v has two children v_1, v_2:
9    Set (G,u,τ,c) := Transformation((G,u,τ,c),v_1)
10   Set (G,u,τ,c) := Transformation((G,u,τ,c),v_2)
11 Else if v has n children:
```

```
    Update (G,u,τ,c) := ρ2((G,u,τ,c),v)
    Let v1,v2 be the new children
    Set (G,u,τ,c) := Transformation((G,u,τ,c),v1)
    Set (G,u,τ,c) := Transformation((G,u,τ,c),v2)

Return (G,u,τ,c)
```

Proof of Theorem 4.13. Let (G,u,τ,c) be a tree network and $v \in G$ a node. We show that the execution of the algorithm in Listing 4.3 on (G,u,τ,c) and v computes an equivalent network where the subgraph belonging to v is a binary tree via structural induction:

Base Case: Suppose that v is a leaf, then the algorithm returns the network (G,u,τ,c) and the subgraph belonging to v is already a binary tree.

Induction Hypothesis: We assume that the statement is true for all tree networks (G,u,τ,c) and all nodes $v \in G$ such that the subtree belonging to v has at most n nodes, where $n \in \mathbb{N}_0$ is arbitrary but fixed.

Inductive Step: Suppose that v is a node such that all the subgraphs belonging to the children of v have at most n nodes. Then, there are three cases:

- If v is not the root and has exactly one child v_1, then the operation ρ_1 deletes v and replaces it with v_1. Furthermore, the reductive execution of the algorithm on the updated network (G,u,τ,c) and the node v_1 ensures that the subgraph belonging to v_1 is binary due to the induction hypothesis.
- If v has two children v_1, v_2, then the execution of the algorithm on both nodes yields binary subgraphs (again due to the induction hypothesis) and the resulting subgraph belonging to v is binary.
- If v has more children $v_1, \ldots, v_n$, then the operation ρ_2 adds two new auxiliary nodes which become the only children of v. Since the algorithm is executed on both nodes, the belonging subgraphs are binary trees (again due to the induction hypothesis) and hence also the subgraph belonging to v.

Hence, the assumption holds. □

From the previous proof, we can derive that the structure of the transformed network is a special binary tree.

Corollary 4.14. *Given a tree network (G,u,τ,c) with root $r \in G$, the execution of the algorithm in Listing 4.3 on (G,u,τ,c) and r yields a network (G',u',τ',c'), where*

- *the graph G' is a binary tree, and*
- *if $r \in G'$ is the root, then for each child v of r the subgraph which belongs to v is a* complete binary tree.

Proof. It follows from Theorem 4.13 that the resulting network is a binary tree network. Furthermore, we can see in the proof that the subgraphs of the children of the root are

complete binary since every node which is not the root and has exactly one child is merged with this child. Hence, every node in the graph which is neither the root nor a leaf has exactly two children. □

Since the operations ρ_1, ρ_2 preserve the equivalence of networks, the concatenation of the operations also preserves the equivalence. The next corollary follows directly.

Corollary 4.15. *For every tree network, there exists an equivalent tree network, where the underlying graph is a binary tree.*

Proof. This follows directly from Theorem 4.13 since the equivalence of the networks is preserved under the operations ρ_1 and ρ_2 (see Lemma 4.10 and 4.12). □

For tree networks with bounded maximal degree of the nodes, the runtime of the algorithm is polynomial in the size of the network.

Corollary 4.16. *Let (G, u, τ, c) be a tree network, where the tree G has degree $k \in \mathbb{N}_0$ and $|G| = n$. Then, an equivalent binary tree network can be computed in $O(n \cdot k)$ steps.*

Proof. We obtain such an equivalent binary tree network by executing the algorithm in Listing 4.3 on the tree network (G, u, τ, c). If k is the maximal degree of any node in the tree G, then there are at most $n \cdot k$ recursive calls of the algorithm, where the factor k is caused by the split operation ρ_2. □

We define another operation on tree networks, which ensures that a particular node is either a leaf or has exactly one child. Therefore, it might add an auxiliary node which increases the height of the subtree. Compare the definition with Figure 4.8 which depicts the transformation if the node has at least two children.

Definition 4.17 (Single Child)**.** *Let (G, u, τ, c) be a tree network and $v \in G$ a node. Then, we define an operation $\rho_3((G, u, \tau, c), v)$ which maps to the network (G, u, τ, c) if v has at most one child, otherwise it maps to a network (G', u', τ', c') which resembles (G, u, τ, c) except that*

- *an auxiliary node v^0 and an arc (v, v^0) are added with*

$$u(v, v^0) := \sum_{i=1}^{n} u(v, v_i), \quad \tau(v, v^0) := 0, \text{ and } c(v, v^0) := 0,$$

  where $v_1, \ldots, v_n$ are the original children of v.
- *The arcs (v, v_i) are replaced by the arcs (v^0, v_i) with the same capacity, transit time, and costs.*

As with operation ρ_2, the capacity of the newly created arc (v, v^0) may also be larger than the sum of the capacities of the arcs (v, v_i), $i \in \{1, \ldots, n\}$. A network with a larger capacity

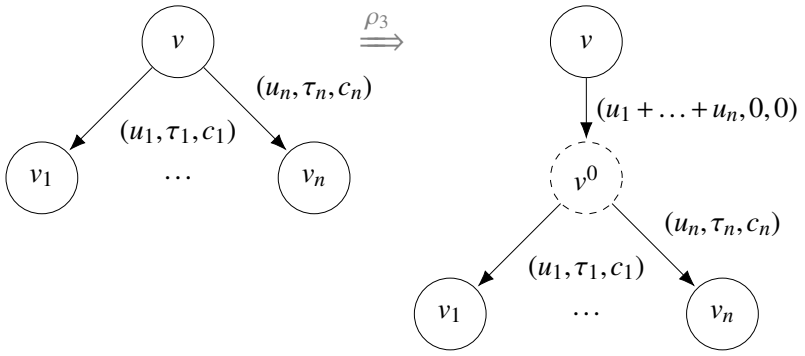

Figure 4.8: Transformation via the operation ρ_3, which creates an auxiliary node if the node v has at least two children. The original children become the children of the auxiliary node.

on the arc (v, v^0) would also be equivalent to the original network, but computations on the network might be slower due to the usage of larger numbers (see Figure 4.9 for an example).

Lemma 4.18. *For a tree network (G, u, τ, c) and any node $v \in G$, the network $\rho_3((G, u, \tau, c), v)$ is equivalent to the network (G, u, τ, c).*

Proof. Again, the number of leaves (or sinks) is not changed by this operation. Furthermore, we can show the equivalence of the networks by constructing equivalent flows for both directions. □

We define another type of trees that resemble binary trees except that the root has at most one child. For our problem, this is the simplest type of trees and will be used in the next section. We call those trees *almost-binary trees* and Figure 4.9 contains an example of an almost-binary tree network.

Definition 4.19 (Almost-binary Tree)**.** *An* almost-binary tree *is a tree G which satisfies that*

- *the root node $r \in G$ has at most one child $v \in G$, and*
- *the subtree belonging to the node v is a complete binary tree (if v exists).*

Theorem 4.20. *For each tree network (G, u, τ, c), there exists an equivalent tree network where the underlying graph is an almost-binary tree.*

Proof. Corollary 4.14 shows that there exists a binary tree network (G_0, u_0, τ_0, c_0) which is equivalent to (G, u, τ, c). It also states that for each child v of the root $r \in G_0$, the subgraph belonging to v is a complete binary tree.

If r has no or exactly one child, then the network is already almost-binary. Otherwise, r has two children and the tree G_0 is complete binary. Then, the result of the operation ρ_3 on (G_0, u_0, τ_0, c_0) and r yields a network

$$(G_1, u_1, \tau_1, c_1) := \rho_3((G_0, u_0, \tau_0, c_0), r) \equiv (G_0, u_0, \tau_0, c_0) \equiv (G, u, \tau, c),$$

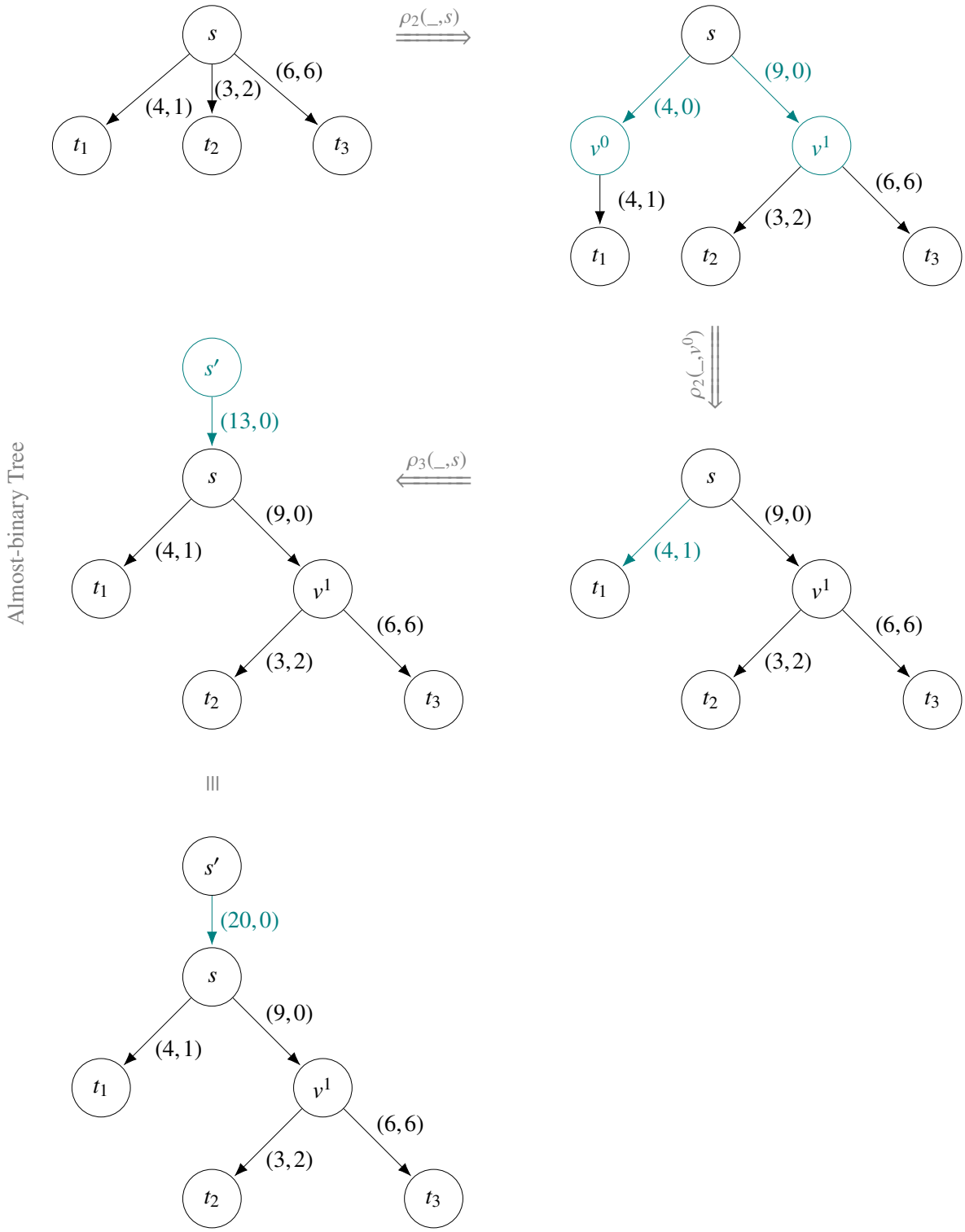

Figure 4.9: All depicted trees lie in one equivalence class. The top left tree is non-binary and can be transformed into the top right tree which is binary. This tree can be transformed into the middle right tree which is complete binary. There exists a transformation into the middle left tree which is an almost-binary tree.
The operation ρ_3 chooses the smallest possible capacity, but any larger capacity would also yield an equivalent tree (e.g. the tree at the bottom).

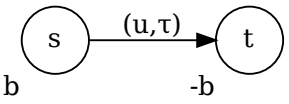

Figure 4.10: The unambiguous structure of an almost-binary tree (G, u, τ, c) with one sink and a balance function b.

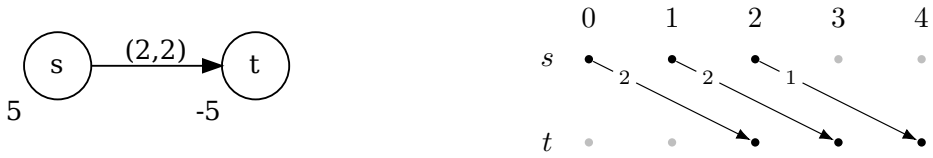

Figure 4.11: An almost-binary tree and the optimal 1-uniform flow.

where the root $r \in G_1$ has exactly one child and the subtree of the child is a complete binary tree. Hence, the statement is correct. □

We have shown that for each tree network (G, u, τ, c), there exist equivalent tree networks (G_0, u_0, τ_0, c_0) and (G_1, u_1, τ_1, c_1) such that the tree G_0 is a binary tree and the tree G_1 is an almost-binary tree.

The equivalence of tree networks (see Definition 4.8) partitions the set of all tree networks into equivalence classes. In one equivalence class, there exists for each flow on each network in the class an equivalent flow in every other network in the same class. Any problem that we considered in Chapter 3 has the same optimal value for every network in the same equivalence class. If we know the transformation of any network into an equivalent network with a special form (e.g. almost-binary), we can reduce the problem and solve it only for those special graphs.

In the next section, we do exactly this and consider only almost-binary graphs.

4.3 Optimal Solutions for Small Trees

In this section, we analyze the problem stated in Definition 4.1 on almost-binary tree networks. We start by calculating the optimal solutions on small almost-binary trees and observe whether the findings can be transferred efficiently onto larger trees.

Single Sink We consider almost-binary tree networks with one leaf (thus one sink). Then, the tree consists of exactly two nodes with one arc in between (see Figure 4.10).

For any balance function, we can compute the shortest time horizon and the corresponding flow easily. We calculate the number of iterations and add the length of the path from the source to the sink minus one (since we started the first iteration at point in time zero). Figure 4.11 contains an example.

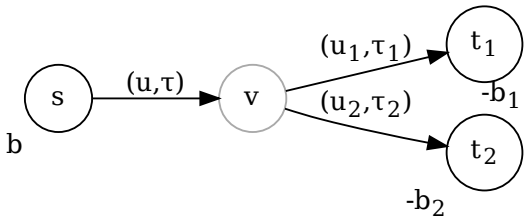

Figure 4.12: The unambiguous structure of an almost-binary tree (G, u, τ, c) with two sinks and a balance function b.

Lemma 4.21. *Let (G, u, τ, c) be an almost-binary tree network with one leaf and a balance function b and variables as implicitly defined in Figure 4.10. Then, the minimal time horizon for a 1-uniform flow satisfying b is*

$$T := \left\lceil \frac{b}{u} \right\rceil + \tau - 1.$$

Proof. The optimal uniform flow which satisfies the balances sends as much flow as possible at each point in time, hence u units. The rest of the formula follows directly. □

We have shown that an optimal solution for an almost-binary tree with one sink can be calculated by a very simple formula. Let us consider trees with two sinks now.

Two Sinks We continue with almost-binary tree networks with two leaves. From the definition of almost-binary trees, we can derive that the tree has exactly four nodes and the form depicted in Figure 4.12.

Now, we consider the problem of finding a 1- or 2-uniform flow with minimal time horizon which satisfies the balances. There are three different types of feasible solutions for almost-binary trees with two sinks:

- A 2-uniform flow that fills one sink first and the other sink afterward. Figure 4.13 contains an example.
- A 2-uniform flow whose first subflow completely fills one sink and partly fills the other sink. The second subflow fills the remainder of the other sink. If there is no remainder, then the flow is a 1-uniform flow, see Figure 4.14 for an example.
- A 2-uniform flow that fills both sinks in both subflows but with different rates, see Figure 4.15.

For each type of solution, we give a function that calculates the time horizon in the special case.

We can directly calculate the optimal solution of the first type (that fills the sinks one after another). It needs to fill the sink with the longer path first since this reduces the time at the end where no new flow is sent but flow inside the network has not arrived yet. The following function calculates its time horizon.

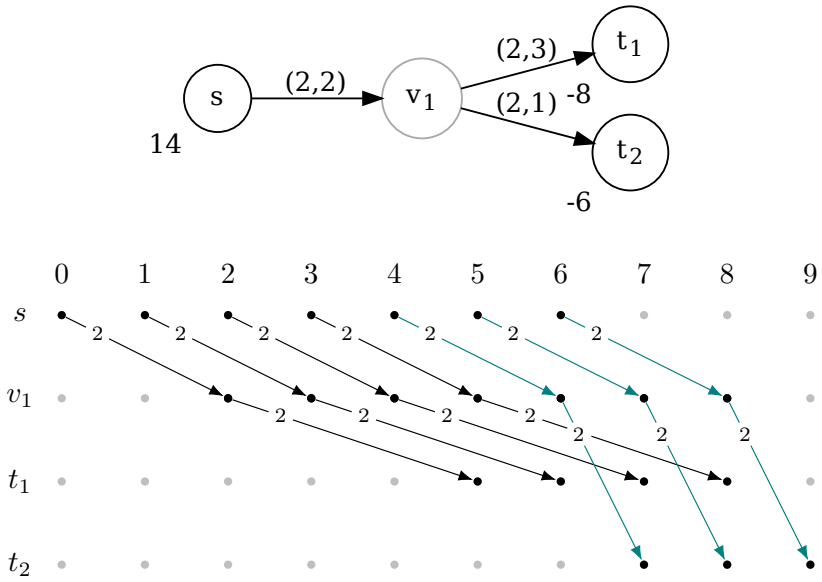

Figure 4.13: A network and the optimal solution on this network. It fills one sink after another and starts with the sink with the longer path. If the subflows were switched, the time horizon would be longer.

Remark 4.22 (First Type). For the network in Figure 4.12, we define $\mathbf{u}_1 := \min\{u, u_1\}$, $\mathbf{u}_2 := \min\{u, u_2\}$, $\boldsymbol{\tau}_1 := \tau + \tau_1$, and $\boldsymbol{\tau}_2 := \tau + \tau_2$. Then, $f(\mathbf{u}_1, \boldsymbol{\tau}_1, b_1, \mathbf{u}_2, \boldsymbol{\tau}_2, b_2)$ calculates the time horizon of the optimal solution of the first type as

$$f : \mathbb{N}_0^6 \to \mathbb{N}_0, (\mathbf{u}_1, \boldsymbol{\tau}_1, b_1, \mathbf{u}_2, \boldsymbol{\tau}_2, b_2) \mapsto \begin{cases} \left\lceil \frac{b_1}{\mathbf{u}_1} \right\rceil + \left\lceil \frac{b_2}{\mathbf{u}_2} \right\rceil + \max\left\{ \boldsymbol{\tau}_2, \boldsymbol{\tau}_1 - \left\lceil \frac{b_2}{\mathbf{u}_2} \right\rceil \right\} - 1, & \boldsymbol{\tau}_1 \geq \boldsymbol{\tau}_2, \\ f(\mathbf{u}_2, \boldsymbol{\tau}_2, b_2, \mathbf{u}_1, \boldsymbol{\tau}_1, b_1), & \text{otherwise.} \end{cases}$$

For the example in Figure 4.13, this yields $\mathbf{u}_1 := 2$, $\mathbf{u}_2 := 2$, $\boldsymbol{\tau}_1 := 5$, and $\boldsymbol{\tau}_2 := 3$ and the time horizon is $f(\mathbf{u}_1, \boldsymbol{\tau}_1, b_1, \mathbf{u}_2, \boldsymbol{\tau}_2, b_2) = \left\lceil \frac{8}{2} \right\rceil + \left\lceil \frac{6}{2} \right\rceil + 3 - 1 = 9$.

For the second type of solutions (which completely fill one sink and partly fill the other sink, first), we can compute the time horizon of a solution if two flow rates $x_1, x_2 \in \mathbb{N}_0$ are given. Afterward, the optimal solution of the second type with the given flow rates, fills both sinks via the flow rates until one sink is satisfied. Then, it fills the other sink as fast as possible. The time horizon is always smaller if the subflow filling both sinks comes first, since then the time after the iterations is minimized.

Remark 4.23 (Second Type). For the network in Figure 4.12, we define $\mathbf{u}_1 := \min\{u, u_1\}$, $\mathbf{u}_2 := \min\{u, u_2\}$, $\boldsymbol{\tau}_1 := \tau + \tau_1$, and $\boldsymbol{\tau}_2 := \tau + \tau_2$. Then, $g(\mathbf{u}_1, \boldsymbol{\tau}_1, b_1, \mathbf{u}_2, \boldsymbol{\tau}_2, b_2, x_1, x_2)$ calculates the time horizon for a solution of the second type which fills sink t_1 at rate $1 \leq x_1 \leq \mathbf{u}_1$ and

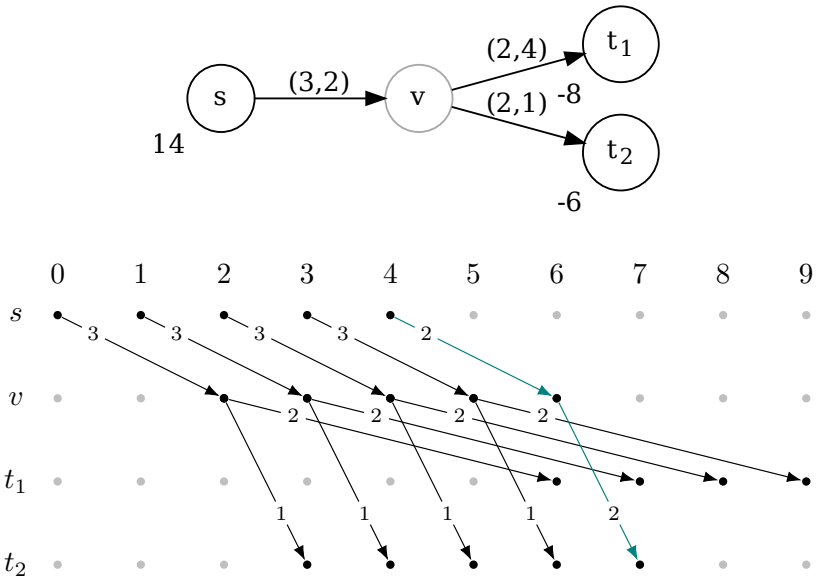

Figure 4.14: A network and the optimal solution on this network. The first subflow completely fills the sink t_1 and partly fills the sink t_2. The second subflow fills the remainder of sink t_2.

sink t_2 at rate $1 \leq x_2 \leq \mathbf{u}_2$:

$$g : \mathbb{N}_0^8 \to \mathbb{N}_0,$$

$$(\mathbf{u}_1, \boldsymbol{\tau}_1, b_1, \mathbf{u}_2, \boldsymbol{\tau}_2, b_2, x_1, x_2) \mapsto \begin{cases} d + \left\lceil \frac{b_2 - d \cdot x_2}{\mathbf{u}_2} \right\rceil + \max\left\{ \boldsymbol{\tau}_2, \boldsymbol{\tau}_1 - \left\lceil \frac{b_2 - d \cdot x_2}{\mathbf{u}_2} \right\rceil \right\} - 1, & d := \left\lceil \frac{b_1}{x_1} \right\rceil \leq \left\lceil \frac{b_2}{x_2} \right\rceil, \\ g(\mathbf{u}_2, \boldsymbol{\tau}_2, b_2, \mathbf{u}_1, \boldsymbol{\tau}_1, b_1, x_2, x_1), & \text{otherwise.} \end{cases}$$

For the example in Figure 4.14, this yields $\mathbf{u}_1 := 2$, $\mathbf{u}_2 := 2$, $\boldsymbol{\tau}_1 := 6$, and $\boldsymbol{\tau}_2 := 3$. For $x_1 := 2$, $x_2 := 1$, the time horizon is $g(\mathbf{u}_1, \boldsymbol{\tau}_1, b_1, \mathbf{u}_2, \boldsymbol{\tau}_2, b_2, x_1, x_2) = 4 + \left\lceil \frac{6-4 \cdot 1}{2} \right\rceil + 5 - 1 = 9$.

For the third type of solutions (where both subflows fill both sinks at different rates), we can compute the time horizon of a solution if four flow rates $x_1, x_2, y_1, y_2 \in \mathbb{N}_0$ and a number of iterations $d \in \mathbb{N}_0$ for the first subflow is given. Then, the first subflow fills sink t_1 at rate x_1 and sink t_2 at rate x_2 d times. The second subflow fills both sinks with the rates y_1 and y_2 until the balances are satisfied.

Not all combinations of rates $x_1, x_2, y_1, y_2 \in \mathbb{N}_0$ and iterations $d \in \mathbb{N}_0$ are feasible, hence the following function is only defined if the rates and iterations allow a feasible 2-uniform flow.

Remark 4.24 (Third Type). For the network in Figure 4.12, we define $\mathbf{u}_1 := \min\{u, u_1\}$, $\mathbf{u}_2 := \min\{u, u_2\}$, $\boldsymbol{\tau}_1 := \tau + \tau_1$, and $\boldsymbol{\tau}_2 := \tau + \tau_2$. Furthermore, we have $1 \leq x_1, y_1 \leq \mathbf{u}_1$, $1 \leq x_2, y_2 \leq \mathbf{u}_2$, and $d \in N$. Then, $h(\boldsymbol{\tau}_1, b_1, \boldsymbol{\tau}_2, b_2, x_1, x_2, y_1, y_2, d)$ calculates the time horizon for

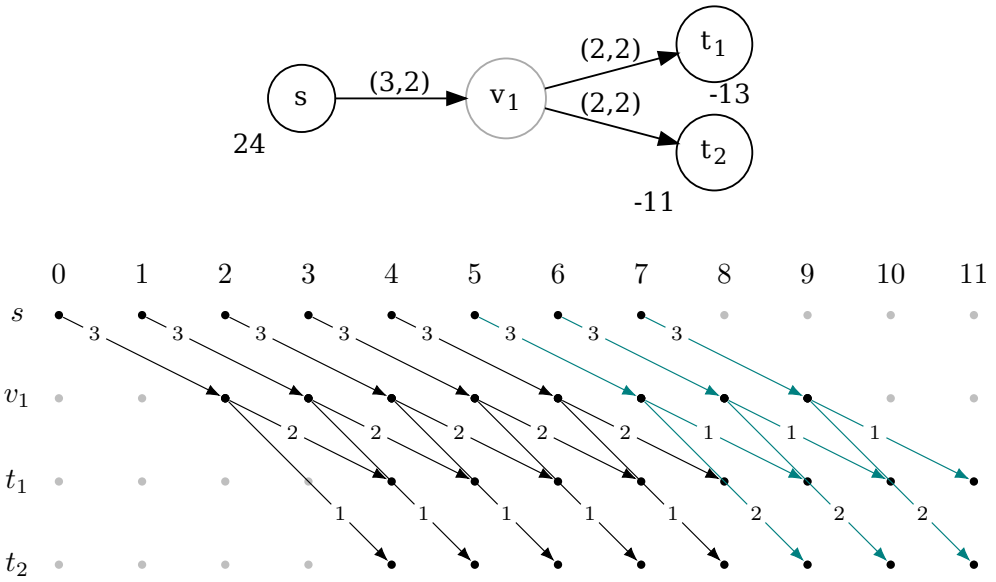

Figure 4.15: A network and the optimal solution on this network. The first subflow fills t_1 five times with rate 2 and t_2 with rate 1. The second subflow fills t_1 with rate 1 and t_2 with rate 2.

a solution of the third type (if the rates and number of iterations is feasible) as

$$h : \left\{ (\boldsymbol{\tau}_1, b_1, \boldsymbol{\tau}_2, b_2, x_1, x_2, y_1, y_2, d) \in \mathbb{N}_0 \;\middle|\; 0 < e := \left\lceil \frac{b_1 - d \cdot x_1}{y_1} \right\rceil = \left\lceil \frac{b_2 - d \cdot x_2}{y_2} \right\rceil \right\} \to \mathbb{N}_0,$$
$$(\boldsymbol{\tau}_1, b_1, \boldsymbol{\tau}_2, b_2, x_1, x_2, y_1, y_2, d) \mapsto d + e + \max\{\boldsymbol{\tau}_1, \boldsymbol{\tau}_2\} - 1.$$

For the example in Figure 4.15, this yields $\mathbf{u}_1 := 2$, $\mathbf{u}_2 := 2$, $\boldsymbol{\tau}_1 := 4$, and $\boldsymbol{\tau}_2 := 4$. For $x_1 := 2$, $x_2 := 1$, $y_1 := 1$, $y_2 := 2$, and $d := 5$, the time horizon is $h(\boldsymbol{\tau}_1, b_1, \boldsymbol{\tau}_2, b_2, x_1, x_2, y_1, y_2, d) = 5 + \left\lceil \frac{13-5\cdot 2}{1} \right\rceil + 4 - 1 = 11$.

Lemma 4.25. *Let (G, u, τ, c) be an almost-binary tree network with two leaves and a balance function b and variables as implicitly defined in Figure 4.12. We define $\mathbf{u}_1 := \min\{u, u_1\}$, $\mathbf{u}_2 := \min\{u, u_2\}$, $\boldsymbol{\tau}_1 := \tau + \tau_1$, and $\boldsymbol{\tau}_2 := \tau + \tau_2$. Then, the quickest 1- or 2-uniform flow has time horizon*

$$\begin{aligned} T := \min\big(&\{\, f(\mathbf{u}_1, \boldsymbol{\tau}_1, b_1, \mathbf{u}_2, \boldsymbol{\tau}_2, b_2) \,\} \\ &\cup \{\, g(\mathbf{u}_1, \boldsymbol{\tau}_1, b_1, \mathbf{u}_2, \boldsymbol{\tau}_2, b_2, x_1, x_2) \mid 1 \le x_1 \le \mathbf{u}_1,\ 1 \le x_2 \le \mathbf{u}_2,\ x_1 + x_2 \le u \,\} \\ &\cup \{\, h(\boldsymbol{\tau}_1, b_1, \boldsymbol{\tau}_2, b_2, x_1, x_2, y_1, y_2, d) \mid \\ &\quad 1 \le x_1 \le \mathbf{u}_1,\ 1 \le y_1 \le \mathbf{u}_1,\ 1 \le x_2 \le \mathbf{u}_2, \\ &\quad 1 \le y_2 \le \mathbf{u}_2,\ x_1 + x_2 \le u,\ y_1 + y_2 \le u, \\ &\quad d < \min\left\{ \left\lceil \tfrac{b_1}{x_1} \right\rceil, \left\lceil \tfrac{b_2}{x_2} \right\rceil \right\},\ \left\lceil \tfrac{b_1 - d\cdot x_1}{y_1} \right\rceil = \left\lceil \tfrac{b_2 - d\cdot x_2}{y_2} \right\rceil \,\} \big). \end{aligned}$$

Proof. First, we show that the arguments for the function h are always feasible. Since

$d < \min\left\{\left\lceil \frac{b_1}{x_1} \right\rceil, \left\lceil \frac{b_2}{x_2} \right\rceil\right\}$, it holds that $d \cdot x_1 < b_1$ and $d \cdot x_2 < b_2$, hence $\left\lceil \frac{b_1 - d \cdot x_1}{y_1} \right\rceil, \left\lceil \frac{b_2 - d \cdot x_2}{y_2} \right\rceil > 0$. Furthermore, we enforce the equality $\left\lceil \frac{b_1 - d \cdot x_1}{y_1} \right\rceil = \left\lceil \frac{b_2 - d \cdot x_2}{y_2} \right\rceil$, hence the combination is feasible and the function h calculates a time horizon.

Now, we show that the time horizon of the optimal solution of the observed problem is indeed T. It is obvious that every solution of the problem consists either of one subflow that fills both sinks (which is represented by g), two subflows that fill both sinks separately (which is represented by f), two subflows where one subflow fills both sinks and the other subflow fills only one sink (which is also represented by g), or two subflows which both fill both sinks (which is represented by h). For each possible solution, either the solution itself on an even quicker solution is calculated by the defined functions. Hence, the optimal time horizon is T. □

We show that if the minimal capacities $\mathbf{u}_1, \mathbf{u}_2$ on the paths to the sinks t_1, t_2 are divisors of the demand b_1, b_2, then the optimal solution has either the first or the second type.

Lemma 4.26. *Let (G, u, τ, c) be an almost-binary tree network with two leaves and a balance function b and variables as implicitly defined in Figure 4.12. We define $\mathbf{u}_1 := \min\{u, u_1\}$, $\mathbf{u}_2 := \min\{u, u_2\}$, $\boldsymbol{\tau}_1 := \tau + \tau_1$, and $\boldsymbol{\tau}_2 := \tau + \tau_2$. If*

$$\mathbf{u}_1 \mid b_1 \quad \textit{and} \quad \mathbf{u}_2 \mid b_2,$$

then the minimal time horizon for a 1- or 2-uniform flow satisfying b is

$$\begin{aligned} T := \min(\ &\{\ f(\mathbf{u}_1, \boldsymbol{\tau}_1, b_1, \mathbf{u}_2, \boldsymbol{\tau}_2, b_2)\ \} \\ &\cup \{\ g(\mathbf{u}_1, \boldsymbol{\tau}_1, b_1, \mathbf{u}_2, \boldsymbol{\tau}_2, b_2, x_1, x_2) \mid 1 \le x_1 \le \mathbf{u}_1,\ 1 \le x_2 \le \mathbf{u}_2,\ x_1 + x_2 \le u\ \}\ ). \end{aligned}$$

Proof. We show that for any solution of the third type, there exists a solution having the first or second type with smaller or equal time horizon.

Suppose that $\boldsymbol{\tau}_1 \ge \boldsymbol{\tau}_2$ and f is a flow of the third type.

- If $u = \mathbf{u}_1$, then we can construct a flow f' of the first type, which first fills the sink t_1 as fast as possible and then the sink t_2 afterward. The capacity u is completely utilized at every iteration of the first subflow. Furthermore, the transit time to t_2 is smaller than the time to t_1, hence the flow is actually minimal and has smaller or equal time horizon than f.
- If $u \neq \mathbf{u}_1$, but $d \cdot (u - \mathbf{u}_1) \le b_2$, where $d := \left\lceil \frac{b_1}{\mathbf{u}_1} \right\rceil$ is the number of iterations to fill t_1. Then, we construct a flow f' of the second type, where the first subflow completely fills t_1 and the remaining capacity $u - \mathbf{u}_1$ is used to fill t_2. The second subflow fills the remainder of t_2 as fast as possible. Again, this flow has minimal time horizon and is at least as quick as f.
- If $u \neq \mathbf{u}_1$ and $d \cdot (u - \mathbf{u}_1) > b_2$. Then, it follows that $\left\lceil \frac{b_2}{\min\{\mathbf{u}_2, u - \mathbf{u}_1\}} \right\rceil \cdot \mathbf{u}_1 \le b_1$, hence we can construct a flow f of the second type, where the first subflow completely fills the sink t_2 with capacity $\min\{\mathbf{u}_2, u - \mathbf{u}_1\}$ and partly fills the sink t_1 with capacity $\mathbf{u}_1$. Then, the second subflow fills the remainder of t_1. This flow fills the sink t_1 as fast as

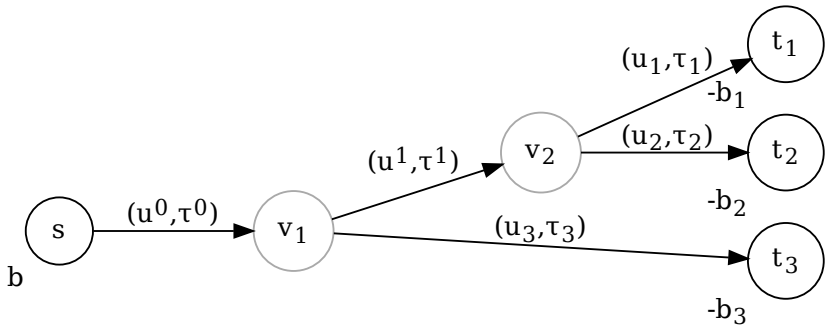

Figure 4.16: The unambiguous structure of an almost-binary tree (G, u, τ, c) with three sinks and a balance function b.

possible and meanwhile also fills the sink t_2. Hence, it is also minimal regarding its time horizon and thus at least as quick as f.

The assumption follows. □

Arbitrary Almost-binary Networks For an almost-binary tree with three leaves, the structure of the tree is still predefined (see Figure 4.16). If there are more than three leaves, then there exist differently structured almost-binary trees with the same number of leaves.

For small almost-binary tree networks and corresponding balance functions, it is relatively easy to calculate the minimal time horizon. But for larger tree networks, the calculation becomes hard. It would be useful to calculate partial solutions for subtrees and compose or extend them to solutions of the whole tree. Figure 4.17 contains an almost-binary tree network, a solution for a subtree, and an extended solution. We see that it is possible to calculate and then extend partial solutions in order to obtain a solution for the whole tree network.

Unfortunately, we cannot guarantee that the extension of an optimal solution for a subtree yields an optimal solution for the whole tree network. The flows in Figure 4.17 are an example; the first flow is optimal for the subtree, but the second flow is not optimal for the whole tree. Note that the extension is already the best possible extension. Figure 4.18 contains a non-optimal solution of the subtree and its extension to an optimal solution for the whole tree network.

The previous example decreases the hope for an efficient polynomial algorithm that solves the problem if we cannot restrict the set of relevant solutions of the subgraphs.

4.4 Linear Relaxation

In the following, we observe whether it is possible to solve the problem defined at the beginning of this chapter via a linear program. More concretely, we give a integer linear program for finding a 1-uniform flow and analyze its linear relaxation.

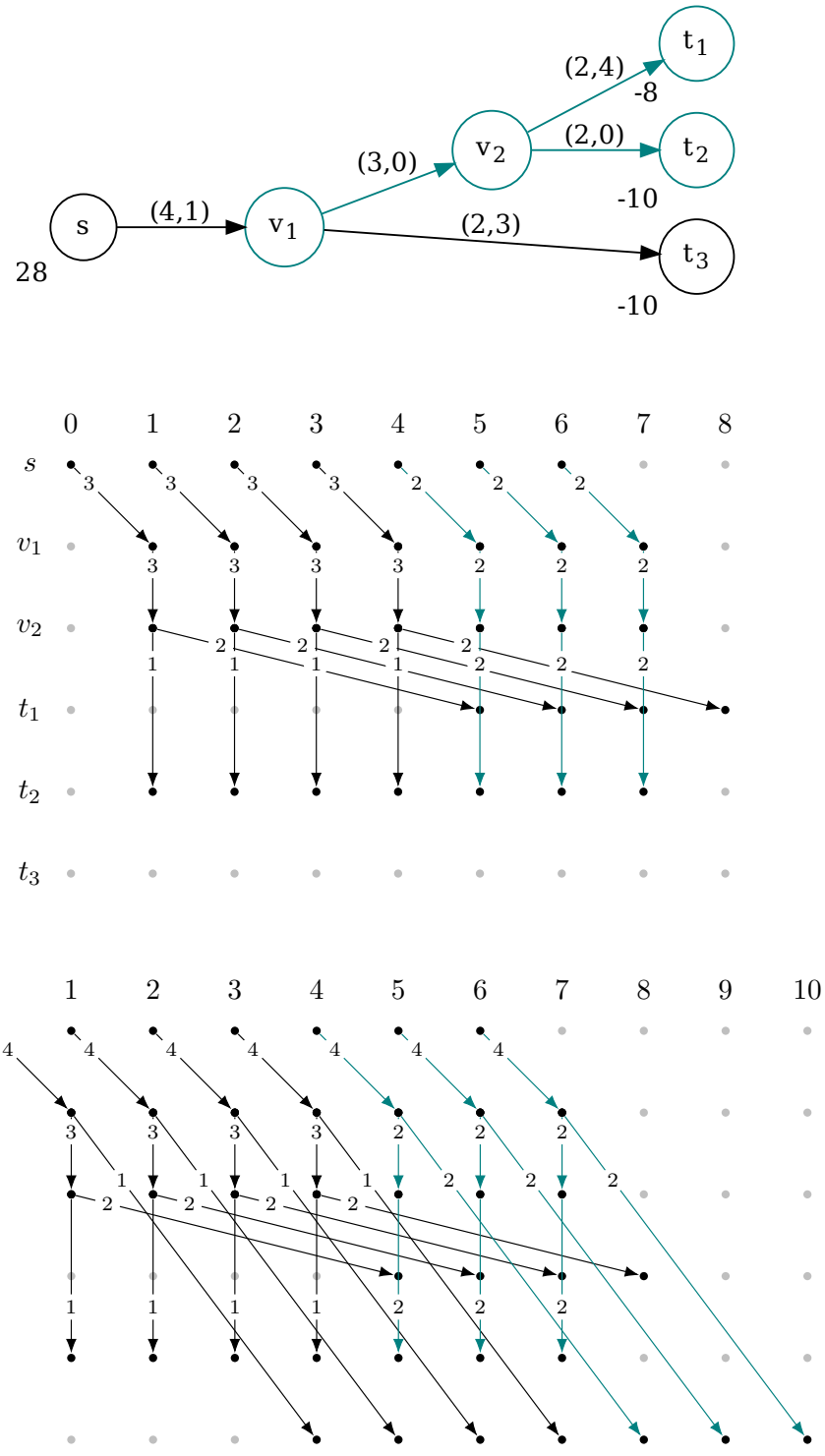

Figure 4.17: A network and two flows on this network. The first flow considers only the subgraph containing v_1, v_2, t_1, t_2, and the corresponding arcs and satisfies the restricted balance function. It is an optimal flow for this reduced network. The second flow extends the solution to a new solution for the whole network.

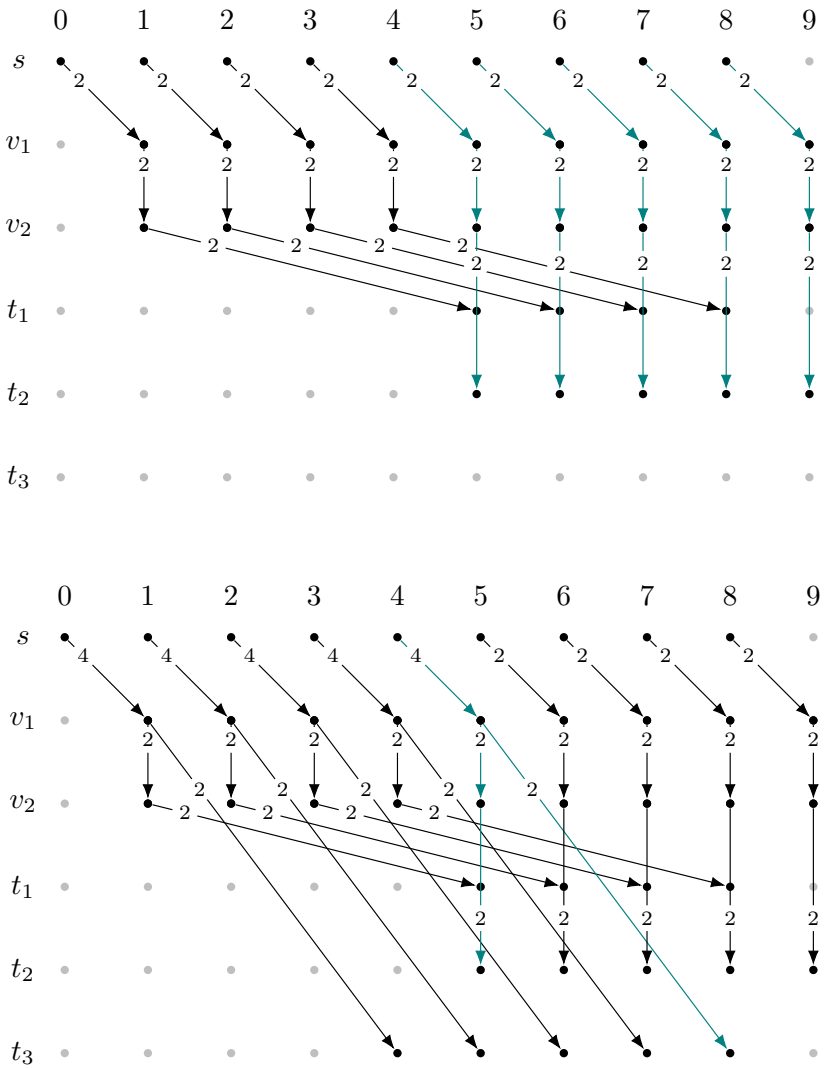

Figure 4.18: Two flows on the network given in Figure 4.17. Again, the first flow considers the reduced network and satisfies the restricted balance function. It is *not* an optimal solution, but the second flow is an extension that is better than any extension of the optimal flow in the previous figure.

Again, $p_i = (s, \dots, t_i)$, $i \in \{1, \dots, h\}$ is the path from the source to a sink.

The following linear program (IP) finds a 1-uniform flow with minimal time horizon for a tree network (G, u, τ, c):

$$\begin{aligned} \min \quad & d \\ \text{s.t.} \quad & \sum_{i \in I_a} -b_i \leq d \cdot u(a), && a \in A,\ I_a := \{i \in \{1, \dots, h\} \mid a \in p_i\}, \\ & d \in \mathbb{N}_0. \end{aligned}$$

Here, d is the number of iterations (the number of times that flow is sent from the source). For 1-uniform flows, minimizing the number of iterations leads to minimizing the total time horizon, since the overhead is exactly $\max_{i \in \{1,\dots,h\}}\{\tau(p_i)\}$ and cannot be reduced. The total time horizon is $d + \max_{i \in \{1,\dots,h\}}\{\tau(p_i)\} - 1$.

Let us consider the relaxation (LP) of the linear program (IP), where $d \in \mathbb{R}_{\geq 0}$. It is computable in polynomial time, but cannot easily be transformed into an optimal integer solution. In the rest of this section, we show how to transform the solution of the relaxed problem into an integer flow of the following type.

Definition 4.27 (Load-Consistent k-Uniform Flow)**.** *A k-uniform flow f is* load-consistent *if for all subflows $f_1, \dots, f_k$ and the corresponding path decompositions $y_i : P^i \to \mathbb{R}_{\geq 0}$, $i \in \{1, \dots, k\}$, it holds that*

$$P^i \supseteq P^{i+1} \text{ and } y^i(p) = y^{i+1}(p) \quad \text{for all } p \in P^{i+1},\ i \in \{1, \dots, k-1\}.$$

We call $y^i(p)$ for $p \in P^i$ the *load* of the path (or the corresponding sink). A load-consistent k-uniform flow is a special k-uniform flow where the load to each sink is consistent up to a certain point in time after which it is zero (if the load is not a divisor of the demand, then the very last iteration that sends flow to the sink may send less than the usual load). Figure 4.19 contains an example of a load-consistent flow.

We can transform a solution of the relaxed linear program (LP) into an optimal load-consistent k-uniform flow with $k \leq h$ via the algorithm in Listing 4.4. The algorithm performs the following steps:

- It calculates an optimal solution $d' \in \mathbb{R}_{\geq 0}$ of the relaxed linear program (LP).
- For any $i \in \{1, \dots, h\}$, if $\frac{b_i}{d'} < 1$, then we fix the load $x_i := 1$ and recalculate d' via (LP) for all remaining sinks $\{1, \dots, h'\}$.
- For all $i \in \{1, \dots, h'\}$, set load $x_i := \left\lfloor \frac{b_i}{d'} \right\rfloor$ to the closest smaller integer. The resulting flow is feasible, since the capacity constraints remain satisfied.
- The time horizon of the load-constraint k-uniform flow is induced by the sink t_i with maximal value $\frac{b_i}{x_i} + \tau(p_i)$. The time horizon can be reduced if the load x_i may be increased. If possible, it updates the load, otherwise, the solution is optimal and the algorithm breaks.

Listing 4.4: Relaxed Solution Into Integer Uniform Flow

```
1  Input: Tree network (G,u,τ) and balance function b
2  Output: Optimal integer load-consistent k-uniform flow satisfying b,
3    if it exists (k ≤ h)
4
5  Calculate optimal solution d' ∈ ℝ≥0 of (LP)
6  For i in 1,...,h do:
7    If -b_i/d' < 1: Fix x_i := 1, update capacities u(a) := u(a) - 1  ∀a ∈ p_i and set b_i := 0
8  Recalculate optimal solution d' ∈ ℝ≥0 of (LP) for remaining sinks t_1,...,t_h'
9  If no solution exists:
10   Return // No optimal integer solution exists
11 Reset network
12
13 Set x_i := ⌊-b_i/d'⌋,  i ∈ {1,...,h'}
14 Sort all sinks such that -b_1/x_1 + τ(p_1) ≥ -b_2/x_2 + τ(p_2) ≥ ... ≥ -b_h/x_h + τ(p_h)
15
16 For i in 1,...,h:
17   If possible to augment load to t_i:
18     Update x_i := x_i + 1
19     Resort sinks i+1,...,h
20   Else if exists t_j, i < j, such that max{ -b_i/(x_i+1) + τ(p_i), -b_j/(x_j-1) + τ(p_j) } < -b_i/x_i + τ(p_i)
21     and updated flow feasible:
22     Update x_i := x_i + 1, x_j := x_j - 1
23     Resort sinks i+1,...,h
24   Else:
25     Break, the solution is optimal
26
27 Return x_1,...,x_h
```

In the following, we prove a few properties.

We start by showing that the flow obtained by sending x_i to sink t_i until the demand is satisfied results in a feasible load-consistent flow.

Lemma 4.28. *The flow obtained after line 13 in Listing 4.4 is a feasible integer load-consistent k-uniform flow with $k \leq h$.*

Proof. Obviously, the loads $x_1, \ldots, x_h$ and the number of iterations d are integer. We consider the sinks $t_1, \ldots, t_{h'}$ that are not fixed to 1 in line 7.

The constraint $\sum_{i \in I} -b_i \leq d' \cdot u(a)$ in (LP) implies that $\sum_{i \in I} \left\lfloor \frac{-b_i}{d'} \right\rfloor \leq \sum_{i \in I} \frac{-b_i}{d'} \leq u(a)$, hence the flow is feasible.

Furthermore, the load to t_i is consistent until the balances b_i are satisfied. Furthermore, there are at most h different points in time where the demand of a sink is finally satisfied, hence

the flow is k-uniform with k at most h. □

We have shown that the first integer flow is feasible. Now, we prove that the updates also yield feasible k-uniform flows.

Lemma 4.29. *The flow obtained after lines 17-19 in Listing 4.4 is feasible.*

Proof. It is possible to augment load x_i to sink t_i if, for all arcs $a \in p_i$, the inequality $\sum_{j \in I} x_j \leq u(a)$ is strict. Hence, the inequality $x_i + 1 + \sum_{i \neq j \in I} x_j \leq u(a)$ is also valid and the resulting flow is feasible. □

In line 21 of the algorithm, it already says that the flow may only be updated if the resulting flow is feasible, meaning that the updated loads may not violate any capacity constraints. We show that we only need to check a small set of sinks whether we can decrease their load in order to increase the load to t_i.

Lemma 4.30. *Let $a_1, \ldots, a_k \in p_i$ be the arcs that restrict the load to t_i. In lines 20-23, if there exists such a sink t_j, then the arcs also lie on the path to t_j, hence $a_1, \ldots, a_k \in p_j$.*

Proof. The arcs $a_1, \ldots, a_m \in p_i$ restrict the load to t_i, hence $\sum_{i \in I} x_i = u(a_n)$ for $1 \leq n \leq m$.
If t_j is another sink with $a_\ell \notin p_j$ for an $\ell \in \{1, \ldots, m\}$ then the updated flow is not feasible, since

$$x_i + 1 + \sum_{i \neq k \in I} x_i > u(a_\ell).$$

Thus, we only need to consider sinks t_j with $a_1, \ldots, a_m \in p_j$. □

The next lemma justifies why the algorithm does not have to consider any sink t_i more than once (through the `for`-loop).

Lemma 4.31. *There exists a load-consistent k-uniform solution with minimal time horizon such that for each sink t_i, $i \in \{1, \ldots, h\}$, the load is at most $x_i = \left\lfloor \frac{-b_i}{d'} \right\rfloor + 1$.*

Initially, the load for each sink t_i, $i \in \{1, \ldots, h\}$, is set to $x_i := \left\lfloor \frac{-b_i}{d'} \right\rfloor$. For the sink with the longest time horizon $\left\lceil \frac{-b_i}{x_i} \right\rceil + \tau(p_i)$, the load may be increased once, but – following from this lemma – we need not consider increasing the load again for this sink if we want to find an optimal solution.

Proof. If $x_i = \left\lfloor \frac{-b_i}{d'} \right\rfloor + 1$, then the time to fill only sink t_i is smaller than $d' + \tau(p_i)$, hence smaller than the time horizon of the optimal relaxed solution, which is a lower bound for the total time horizon of the integer load-consistent solution.
Given a flow with $x_i > \left\lfloor \frac{-b_i}{d'} \right\rfloor + 1$ for a sink t_i, we can set $x_i := \left\lfloor \frac{-b_i}{d'} \right\rfloor + 1$ and obtain another feasible flow with the same time horizon. Hence, the assumption is valid. □

We show that the algorithm computes a solution with a minimal time horizon.

Theorem 4.32. *The result of the algorithm in Listing 4.4 is an optimal integer load-consistent k-uniform flow for $k \leq h$.*

Proof. It follows from Lemma 4.28, Lemma 4.29, and Lemma 4.30 that the resulting flow is feasible and load-consistent k-uniform. We need to show that the time horizon is minimal.

The algorithm returns a solution if the flow cannot be updated by the two actions considered in Lemma 4.29 (lines 17-19) and Lemma 4.30 (lines 20-23).

Suppose that the sinks $t_1, \ldots, t_h$ are sorted such that

$$\frac{-b_1}{x_1} + \tau(p_1) \geq \frac{-b_2}{x_2} + \tau(p_2) \geq \ldots \geq \frac{-b_h}{x_h} + \tau(p_h).$$

The time horizon of the flow is determined by $\frac{-b_1}{x_1} + \tau(p_1)$. Hence, the time horizon can only be reduced if the number of iterations to fill sink t_1 is reduced, respectively if the load x_1 is increased. There is no integer capacity on path p_1 left, otherwise, the algorithm would not have stopped. For the same reason, the load of no other sink t_j may be decreased, without increasing the total time horizon, in order to free capacity and increase the load x_1.

Thus, the time horizon is minimal for all load-consistent k-uniform flows with $k \leq h$. □

In the rest of this chapter, we look at an exemplary execution of the algorithm and analyze its runtime.

Example 4.33. We apply the algorithm on the network in Figure 4.17.

- Calculate the optimal solution $d' \in \mathbb{R}_{\geq 0}$ of (LP) as

$$d' := \max_{a \in A} \left\{ \frac{\sum_{i \in I_a} -b_i}{u(a)} \right\} = \max \left\{ \frac{8}{2}, \frac{10}{2}, \frac{18}{3}, \frac{10}{2}, \frac{28}{4} \right\} = \max\{4,5,6,5,7\} = 7.$$

- Set loads $x_1 := \left\lfloor \frac{-b_1}{d'} \right\rfloor = \left\lfloor \frac{8}{7} \right\rfloor = 1, x_2 := \left\lfloor \frac{-b_2}{d'} \right\rfloor = \left\lfloor \frac{10}{7} \right\rfloor = 1, x_3 := \left\lfloor \frac{-b_3}{d'} \right\rfloor = \left\lfloor \frac{10}{7} \right\rfloor = 1.$
- Sort sinks $\frac{-b_3}{x_3} + \tau(p_3) = \frac{10}{1} + 4 = 14 \geq \frac{-b_1}{x_1} + \tau(p_1) = \frac{8}{1} + 5 = 13 \geq \frac{-b_2}{x_2} + \tau(p_2) = \frac{10}{1} + 1 = 11.$
- Analyze sink t_3: There is free capacity on the path p_3 to sink t_3, hence augment load to $x_3 := 2$. Now, $\frac{b_3}{x_3} + \tau(p_3) = \frac{10}{2} + 4 = 9$.
- Analyze sink t_1: There is no free capacity on the path p_1.
  The restricting arc is (s, v_0) and we check sinks t_2 and t_3:
  - It is $\max\left\{ \frac{-b_1}{x_1+1} + \tau(p_1), \frac{-b_3}{x_3+1} + \tau(p_3) \right\} = 14 > 13$, hence updating the loads x_1, x_3 does not reduce the time horizon.
  - Since $x_2 = 1$, we cannot reduce the load to sink t_2 (the flow would not be feasible anymore).

  We cannot update the flow and the solution is an optimal load-consistent flow.

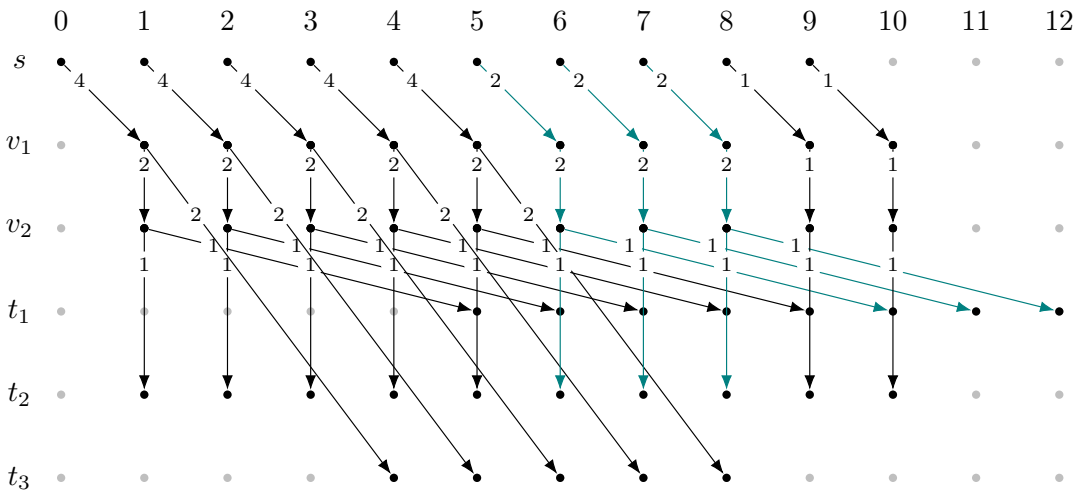

Figure 4.19: The optimal load-consistent k-uniform flow ($k \leq 3$) that satisfies the network in Figure 4.17.

The solution of the algorithm is depicted in Figure 4.19. The optimal k-uniform flow has smaller time horizon, but the obtained result is the load-consistent flow with minimal time horizon.

Lastly, we show that the algorithm has polynomial runtime.

Corollary 4.34. *An integer load-consistent k-uniform flow with minimal time horizon and $k \leq h$ can be computed in polynomial time.*

Proof. Finding a solution to the relaxed linear program (*LP*) has polynomial runtime [Sch98]. We may need to do this twice. The runtime of the `for`-loop in line 6 can be disregarded in comparison to the second `for`-loop.

Lemma 4.31 shows that the `for`-loop in line 16 is correctly chosen. In each iteration of the loop and in the worst case, we compare with all other sinks (even though this can be greatly reduced, see Lemma 4.30) and check all arcs (while checking if the updated flow is feasible). We re-sort the sinks in each iteration, which can also be done in polynomial time.

Hence, the overall time of the algorithm is polynomial. □

5 Lexicographic Costs

In this chapter, we consider flows over time with different kinds of lexicographic cost functions. In the first section, we prove an optimality criterion that was proposed in [Ham89] for minimal static flows regarding a lexicographic cost function.

In the second section, we recapitulate the algorithm by L.R. Ford and D.R. Fulkerson for computing maximal temporally repeated flows. Then, we update the algorithm such that it computes maximal temporally repeated flows where the maximal costs over all points in time are minimal. We show an example where such an optimized flow could be useful and analyze whether we can calculate maximal temporally repeated flows with minimal total costs in a similar way.

5.1 Optimality Criterion via Negative Cycles

In this section, we prove a theorem that was originally proposed – but not proven – by Horst W. Hamacher in [Ham89]. It states the following:

Theorem 5.1. *Let $(G,u,\mathbf{c})$ be a network with a lexicographic cost function $\mathbf{c}$ and a balance function b. A b-flow x is a lexicographic min cost flow if and only if there exists no x-augmenting cycle with lexicographic negative costs.*

This theorem resembles a theorem of M. Klein from 1967 (see [Kle67] and [BS67]), which states that a static b-flow has minimal costs if and only if there exists no augmenting negative cycle. Here, we examine the translation of this theorem to static networks with lexicographic costs. Let us start by looking at the definitions of the terms *lexicographic cost function*, *lexicographic min cost flow*, and *lexicographic negative cycle*.

A *lexicographic cost function* $\mathbf{c} : A \to \mathbb{N}_0^m$ maps each arc to a tuple of costs. Hence, the costs of a flow x are also given as a tuple. The residual graph is adapted analogously.

Definition 5.2 (Residual Lexicographic Costs)**.** *For a flow x over a network $(G,u,\mathbf{c})$, the residual network for a simple cost function $\mathbf{c} : A \to \mathbb{N}_0$ is defined in Definition 2.15. Given a lexicographic cost function $\mathbf{c} : A \to \mathbb{N}_0^m$, we define the residual network $(G,u,\mathbf{c})_x$ analogously and the residual cost function as*

$$\mathbf{c}_x : A_x \to \mathbb{Z}^m,\ a \mapsto \begin{cases} \mathbf{c}(a), & a \in A \\ -\mathbf{c}(\overleftarrow{a}), & a \in \overleftarrow{A}. \end{cases}$$

In order to compare the costs, we need to define an ordering on the set of all costs.

Definition 5.3 (Lexicographic Ordering)**.** *For a set of tuples $\mathbb{Z}^m$, we specify $\prec$ as a* lexicographic ordering *on the set with*

$$(v_1,\dots,v_m) \prec (w_1,\dots,w_m) \quad \textit{iff} \quad (v_1 < w_1) \quad \textit{or} \quad (v_1 = w_1 \textit{ and } (v_2,\dots,v_m) \prec (w_2,\dots,w_m))$$
$$\textit{for} \quad (v_1,\dots,v_m),(w_1,\dots,w_m) \in \mathbb{Z}^m.$$

For a restriction of $\mathbb{Z}^m$ to the set $\mathbb{N}_0^m$, the lexicographic ordering can be adapted.

We show that the previously defined ordering is total, hence that we can compare each pair of tuples of costs. This allows us to define minimal cost flows, afterward.

Lemma 5.4. *The lexicographic ordering $\prec$ on $\mathbb{Z}^m$ is total.*

Proof. The ordering is

- irreflexive, since $(v_1,\dots,v_m) \not\prec (v_1,\dots,v_m)$,
- transitive, since $(v_1,\dots,v_m) \prec (w_1,\dots,w_m) \prec (u_1,\dots,u_m)$ implies $(v_1,\dots,v_m) \prec (u_1,\dots,u_m)$, and
- connected, since either $(v_1,\dots,v_m) \prec (w_1,\dots,w_m)$, $(w_1,\dots,w_m) \prec (v_1,\dots,v_m)$, or $(v_1,\dots,v_m) = (w_1,\dots,w_m)$,

for all $(v_1,\dots,v_m),(w_1,\dots,w_m),(u_1,\dots,u_m) \in \mathbb{Z}^m$. Hence, the order is total. □

Definition 5.5 (Lexicographic Min Cost Flow)**.** *A* lexicographic min cost flow *is a lexicographic flow on a network $(G,u,\mathbf{c})$ which satisfies a balance function b with minimal costs regarding the lexicographic ordering $\prec$.*

Theorem 5.1 gives an optimality criterion for lexicographic min cost flows. It states that a flow has lexicographic minimal costs if there are no augmenting cycles with negative costs. We give the definition of such cycles next.

Definition 5.6 (Augmenting Cycle)**.** *For a network (G,u) and a flow x, a directed cycle C in the residual graph G_x is an* augmenting cycle*.*

An augmenting cycle is a cycle through the residual graph of a flow x along which we can augment x such that we obtain an updated flow x' which satisfies the same balances.

We may also come from the other side and observe two flows x, x' which satisfy the same balances and investigate their differences. The next definition is from [Bue22].

Definition 5.7 (Difference Between Flows)**.** *Let x, x' be two flows over the same network (G,u) satisfying the same balance function b. Then, we define the* difference flow $z := x' \bigtriangleup x$ *over the reverse graph as*

$$z : \overleftrightarrow{A} \to \mathbb{R}_{\geq 0},\ a \mapsto \begin{cases} \max\{0, x'(a) - x(a)\}, & a \in A \\ \max\{0, x(a) - x'(a)\}, & a \in \overleftarrow{A}. \end{cases}$$

We state a few properties that enable us to show Theorem 5.1, afterward. The first property in the following lemma states that a difference flow satisfies flow conservation. This implies by Lemma 2.8 that the flow can be decomposed into a set of cycles.

The second property states that the difference flow is 0 on all arcs that do not lie in the residual graph of the second flow. Hence, it suffices to consider the arcs of the residual graph (instead of observing all arcs in the reverse graph) in future proofs.

Lemma 5.8. *Let $z := x' \bigtriangleup x$ be a difference flow over a network (G, u). Then, the following properties hold:*

- *flow conservation in the reverse graph $\overleftrightarrow{G}$ for each $v \in V$, hence*

$$\sum_{a \in \delta^-(v)} z(a) - \sum_{a \in \delta^+(v)} z(a) = 0,$$

- *and $z(a) = 0$ for all $a \notin A_x$.*

The proof can be found in [Bue22]. Next, we analyze the cost difference between two flows, which is given via the costs of the difference flow.

Lemma 5.9. *Given two flows x, x' over the same network $(G, u, \mathbf{c})$, where $\mathbf{c}$ is a lexicographic cost function, we define the difference flow $z := x' \bigtriangleup x$. Then, it holds that*

$$\mathbf{c}(z) = \mathbf{c}(x') - \mathbf{c}(x).$$

The proof of this lemma is similar to the proof in [Bue22] with the only difference being that we consider lexicographic costs.

Proof. We have to show that

$$\begin{aligned}
&\mathbf{c}(z) = \mathbf{c}(x') - \mathbf{c}(x) \\
\Leftrightarrow &\sum_{a \in \overleftrightarrow{A}} \mathbf{c}(a) \cdot z(a) = \sum_{a \in \overleftrightarrow{A}} \mathbf{c}(a) \cdot x'(a) - \sum_{a \in \overleftrightarrow{A}} \mathbf{c}(a) \cdot x(a) \\
\Leftrightarrow &\sum_{a \in A} \left(\mathbf{c}(a) \cdot z(a) + \mathbf{c}(\overleftarrow{a}) \cdot z(\overleftarrow{a}) \right) = \sum_{a \in A} \mathbf{c}(a) \cdot x'(a) - \sum_{a \in A} \mathbf{c}(a) \cdot x(a) \\
\Leftrightarrow &\sum_{a \in A} \mathbf{c}(a) \cdot \left(z(a) - z(\overleftarrow{a}) \right) = \sum_{a \in A} \mathbf{c}(a) \cdot \left(x'(a) - x(a) \right).
\end{aligned}$$

It holds that

$$\begin{aligned} z(a) - z(\overleftarrow{a}) &= \max\{0, x'(a) - x(a)\} - \max\{0, x(a) - x'(a)\} \\ &= \begin{cases} 0, & x(a) = x'(a) \\ x'(a) - x(a), & x(a) < x'(a) \\ -x(a) + x'(a), & x(a) > x'(a) \end{cases} \\ &= x'(a) - x(a), \end{aligned}$$

hence the first equation holds and the lemma is valid. □

Now, we have all the definitions and lemmata to show the correctness of Theorem 5.1:

Theorem 5.1. *Let $(G, u, \mathbf{c})$ be a network with a lexicographic cost function $\mathbf{c}$ and a balance function b. A b-flow x is a lexicographic min cost flow if and only if there exists no x-augmenting cycle with lexicographic negative costs.*

Proof. We show the theorem by proving both implications of the equivalence statement. Suppose that x is a lexicographic min cost b-flow over a network $(G, u, \mathbf{c})$ with lexicographic costs $\mathbf{c}$ and a balance function b. We indicate that this implies that there does not exist a lexicographic negative cycle in the residual network $(G, u, \mathbf{c})$ via proof by contradiction.

Assume that there exists an x-augmenting lexicographic negative cycle C, hence a cycle with $\mathbf{c}(C) < \mathbf{0} = (0, \ldots, 0)$ and $\min_{a \in C} u_x(a) =: \gamma > 0$. Then, we can augment the flow x along cycle C by γ and obtain a new b-flow x' with costs

$$\begin{aligned} \mathbf{c}(x') = \sum_{a \in A} \mathbf{c}(a) \cdot x'(a) &= \sum_{a \in A \setminus (C \cap \overleftarrow{C})} \mathbf{c}(a) \cdot x(a) + \sum_{a \in C \cap A} \mathbf{c}(a) \cdot (x(a) + \gamma) + \sum_{a \in \overleftarrow{C} \cap A} \mathbf{c}(a) \cdot (x(a) - \gamma) \\ &= \sum_{a \in A} \mathbf{c}(a) \cdot x(a) + \sum_{a \in C} \gamma \cdot \mathbf{c}(a) = \mathbf{c}(x) + \underbrace{\gamma}_{>0} \cdot \underbrace{\mathbf{c}(C)}_{<\mathbf{0}} < \mathbf{c}(x). \end{aligned}$$

The flow x' also satisfies the balances b and has smaller costs than the flow x, hence x is not a lexicographic min cost flow and we have a contradiction. Thus, the implication is correct.

Now, we show the reverse implication and assume that there does not exist an x-augmenting lexicographic negative cycle C. Again, we show the implication via proof by contradiction and assume that x is not a lexicographic min cost flow.

Then, there exists another flow x', which satisfies the balances b and has smaller costs $\mathbf{c}(x') < \mathbf{c}(x)$. We observe that there exists a difference flow $z := x' \bigtriangleup x$ which is not the empty flow. Furthermore, we know from Lemma 5.8 that the flow conservation holds for z and hence there exist cycles $C_1, \ldots, C_n$ in the residual network $(G, u, \mathbf{c})_x$ and values $\gamma_1, \ldots, \gamma_n \in \mathbb{N}_0$ such that the flow z is a linear combination of those cycles with the respective values (by the Flow Decomposition, see Lemma 2.8).

By Lemma 5.9, it holds that $\mathbf{c}(z) = \mathbf{c}(x') - \mathbf{c}(x)$. Furthermore, we know that $\mathbf{c}(x') < \mathbf{c}(x)$ and

thus

$$0 > \mathbf{c}(z) = \gamma_1 \cdot \mathbf{c}(C_1) + \ldots \gamma_n \cdot \mathbf{c}(C_n).$$

We derive that there exists a cycle C_i, $1 \leq i \leq n$, such that $\mathbf{c}(C_i) < \mathbf{0}$ and hence there exists an x-augmenting cycle with lexicographic negative costs and we have a contradiction. It follows that the reverse implication is correct and the theorem holds. □

We have proven an optimality criterion for static lexicographic min cost flows.

5.2 Cost Minimization at each Point in Time

In this section, we observe temporally repeated flows from one source to one sink over a time horizon T. We aim to find flows that maximize multiple objectives and focus on networks with costs. Here, the immediate question is whether we can find a maximal flow with minimal total costs. In [Ham89], the author claims that such flows can be found via an alteration of the *temporally repeated flow technique* [FJF58], hence by calculating maximal temporally repeated flows using static flows but with an adapted cost function.

We were not able to reproduce this hypothesis, but we show that we can adapt the temporally repeated flow technique to find a maximal temporally repeated flow where the costs over all points in time are minimal. Furthermore, we observe the problem of finding maximal flows with minimal total costs and give an intuition on the difficulties of the problem.

We start with the temporally repeated flow technique and the algorithm proposed by Ford and Fulkerson [FJF58]. The algorithm uses the following statement about the relation between a static flow and a temporally repeated flow created from the static flow.

Lemma 5.10. *Given a flow network (G, u, τ), let x be a feasible static flow with path decomposition $y : P \to \mathbb{R}_{\geq 0}$, f the associated temporally repeated flow with time horizon T and $\tau(p) \leq T$ for all $p \in P$. Then, it holds that*

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

Proof. This is proven by the following equations:

$$\begin{aligned}
\text{value}(f) &= \sum_{p \in P} y(p) \cdot (T - \tau(p)) = T \cdot \sum_{p \in P} y(p) - \sum_{p \in P} \tau(p) \cdot y(p) \\
&= T \cdot \text{value}(x) - \sum_{p \in P} \tau(p) \cdot y(p) = T \cdot \text{value}(x) - \sum_{p \in P} \Big(\sum_{a \in p} \tau(a) \Big) \cdot y(p) \\
&= T \cdot \text{value}(x) - \sum_{a \in A} \tau(a) \cdot \Big(\sum_{\substack{p \in P \\ a \in p}} y(p) \Big) = T \cdot \text{value}(x) - \sum_{a \in A} \tau(a) \cdot x(a).
\end{aligned}$$

Hence the hypothesis holds. □

The previous lemma shows that the value of a temporally repeated flow can be calculated via the underlying static flow, the transit times in the network, and the time horizon. The following algorithm from [FJF58] uses this property for the computation of a temporally repeated flow with maximal value. The algorithm computes a static flow that maximizes the right-hand side of the equation in Lemma 5.10 and constructs the associated temporally repeated flow, which consequently has maximal value.

Listing 5.1: Ford-Fulkerson Algorithm for Maximal (s,t)-Flows over Time

```
1 Input: Network (G,u,τ), source and sink s,t∈V, and time horizon T
2 Output: Temporally repeated flow f with time horizon T
3   with maximal flow
4
5 Calculate static (s,t)-flow x that maximizes T·value(x) − ∑_{a∈A} τ(a)·x(a)
6 Calculate path decomposition y : P → ℝ_{≥0}
7 Construct temporally repeated flow f with time horizon T
8 Return f
```

In the next lemma, we show that such a static flow can be computed in polynomial time. This implies that the algorithm has polynomial runtime.

Lemma 5.11. *Let (G,u,τ) be a network, $s \in V$ a sink, $t \in V$ a source, and T a time horizon. Then, a static flow that maximizes*

$$T \cdot \text{value}(x) - \sum_{a \in A} \tau(a) \cdot x(a)$$

such that there exists a path decomposition $y : P \to \mathbb{R}_{\geq 0}$ with $\tau(p) \leq T$ for all $p \in P$ with $y(p) > 0$ can be computed in polynomial time via a minimal cost flow tranformation.

The following proof is from [Bue22].

Proof. First, we show that such a static flow x can be computed by finding a minimal cost circulation considering the transit times τ as the costs. Let $\overline{G}$ be the graph G extended by an arc (t,s) with capacity $u(t,s) := \infty$ and $\tau(t,s) := -T$. Furthermore, $\overline{x}$ is any circulation in $\overline{G}$ and x the corresponding flow in G, then the total transit time is

$$\begin{aligned}\tau(\overline{x}) &= -T \cdot \overline{x}(t,s) + \sum_{a \in A} \tau(a) \cdot \overline{x}(a) = -T \cdot \overline{x}(t,s) + \sum_{a \in A} \tau(a) \cdot x(a) \\ &= -T \cdot \text{value}(x) + \sum_{a \in A} \tau(a) \cdot x(a).\end{aligned}$$

We see that a minimal cost circulation regarding τ maximizes $T \cdot \text{value}(x) - \sum_{a \in A} \tau(a) \cdot x(a)$ for the corresponding (s,t)-flow. A minimal cost circulation can be computed in polynomial time regarding the size of the network (G,u,τ) (see [Kle67]).

It remains to show that, for a minimal cost circulation $\overline{x}$ in $\overline{G}$, the corresponding static flow x in G satisfies for every flow decomposition $y : P \cup C \to \mathbb{R}_{\geq 0}$ that

$$\tau(p) \leq T \text{ for all } p \in P \text{ with } y(p) > 0 \quad \text{and} \quad \tau(q) = 0 \text{ for all } q \in C \text{ with } y(q) > 0.$$

- Suppose that $\tau(p) > T$ and $y(p) > 0$ for a path $p \in P$. Then, there exist backward arcs $\overleftarrow{p}$ in the residual network of $\overline{G}$ with costs

$$\tau(\overleftarrow{p} \cup (s,t)) = -\underbrace{\tau(p)}_{<T} + T < 0.$$

  There exists a negative cycle in the residual graph, thus $\overline{x}$ is not optimal and we have a contradiction.

- Suppose that $\tau(q) \neq 0$ and $y(q) > 0$ for a cycle $q \in C$. Then, it is either $\tau(q) > 0$ and there exist backward arcs $\overleftarrow{q}$ in the residual graph $\overline{G}$ with costs $\tau(\overleftarrow{q}) < 0$. Thus, there exists a negative cycle, $\overline{x}$ is not optimal and we have a contradiction. If $\tau(q) < 0$, then the contradiction follows directly.

Hence, we can find a static flow satisfying the conditions. □

In the remainder of this section, we adapt the algorithm in Listing 5.1 by updating the cost function. We consider networks (G, u, τ, c) with costs. First, we show that we can alter the algorithm such that it computes a maximal temporally repeated flow, where the *maximal costs over all points in time* are minimal.

Definition 5.12 (Costs at a Point in Time)**.** *Let (G, u, τ, c) be a network with a flow over time f with time horizon T. For a point in time $\theta \in \{0, \dots, T\}$, the* costs at θ *are*

$$c(f, \theta) := \sum_{a \in A} c(a) \cdot \left(\sum_{\xi=\theta-\tau(a)+1}^{\theta} f_a(\xi) \right).$$

Let us look at an example observing maximal temporally repeated flows on the network in Figure 5.1 and considering the costs at different points in time.

Example 5.13. Figure 5.1 contains a very simple network, where the path via node v_1 has the same length as the path via node v_2. For a time horizon $T = 20$, the algorithm in Listing 5.1 computes either the flow depicted in Figure 5.2 or the flow depicted in Figure 5.3. Both flows have the same value. Let us examine the costs at several points in time.

In Figure 5.2 at each point in time smaller than 8, the costs are 0, since there is only flow along the arc (s, v_1) which has zero costs. At time 8 the costs are 3, at time 9 the costs are 6 and remain the same for all points in time smaller than 19.

In Figure 5.3, the costs at points 0 and 1 are 0. Then, the costs increase from 2 at point 2 to 16 at point 9. They remain the same until they start decreasing again at point 13. We can see that the maximal costs over all points in time in this flow are 16 and greater than the maximal costs over all points in time for the flow in Figure 5.2 (where the maximum is 6).

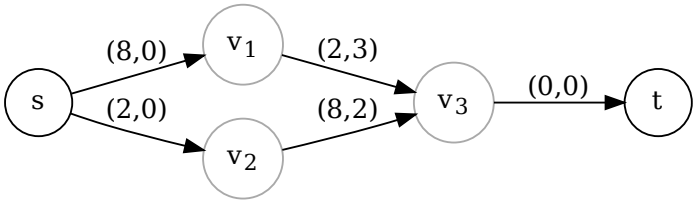

Figure 5.1: A network, where the labels (τ, c) on the arcs represent the transit time and the costs. The capacity is 1 for each arc.

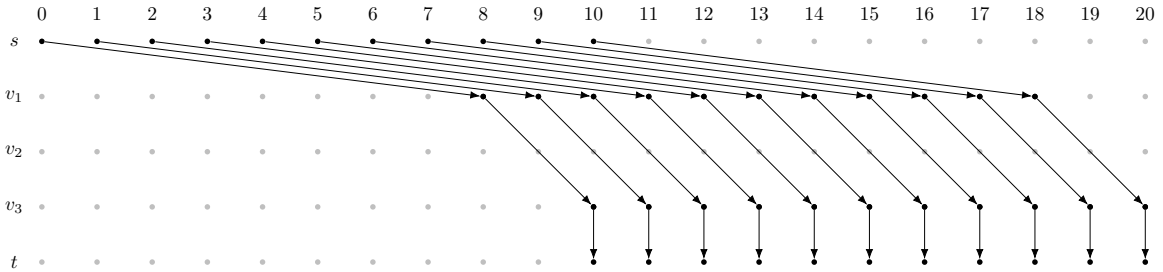

Figure 5.2: A maximal temporally repeated flow on the network depicted in Figure 5.1, where the maximal costs for all points in time are minimal.

For a static flow that maximizes the term in Equation (5.1), we show that the associated temporally repeated flow is maximal and that the static flow minimizes a certain term. This insight will help us to compute a maximal temporally repeated flow with minimal costs over all points in time.

Lemma 5.14. *Let (G, u, τ, c) be a network and T a time horizon. Set $M := \sum_{a \in A} c(a) \cdot u(a) + 1$. Let x be a feasible static flow that maximizes*

$$M \cdot T \cdot \text{value}(x) - \sum_{a \in A} \left(M \cdot \tau(a) + c(a) \cdot \tau(a)\right) \cdot x(a) \tag{5.1}$$

with path decomposition $y : P \to \mathbb{R}_{\geq 0}$ and $\tau(p) \leq T$ for all $p \in P$. Let f be the associated temporally repeated flow with time horizon T.

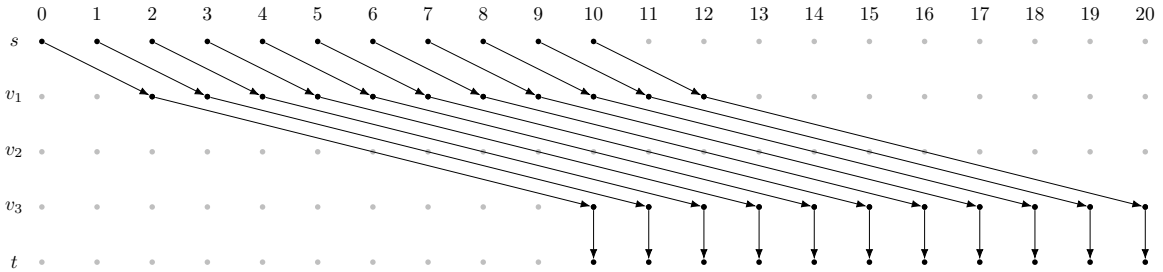

Figure 5.3: A maximal temporally repeated flow on the network depicted in Figure 5.1. The maximal costs for all points in time are not minimal.

Then, f has maximal value and x is the associated static flow which minimizes $\sum_{a\in A} c(a)\cdot \tau(a)\cdot x(a)$.

Proof. With Lemma 5.10, it holds that

$$\begin{aligned}
&M\cdot T\cdot \text{value}(x) - \sum_{a\in A}\big(M\cdot\tau(a)+c(a)\cdot\tau(a)\big)\cdot x(a)\\
&= M\cdot T\cdot \text{value}(x) - M\cdot\sum_{a\in A}\tau(a)\cdot x(a) - \sum_{a\in A} c(a)\cdot\tau(a)\cdot x(a)\\
&= M\cdot\Big(T\cdot\text{value}(x) - \sum_{a\in A}\tau(a)\cdot x(a)\Big) - \sum_{a\in A} c(a)\cdot\tau(a)\cdot x(a)\\
&= M\cdot\text{value}(f) - \sum_{a\in A} c(a)\cdot\tau(a)\cdot x(a).
\end{aligned}$$

Since $x(a)\le u(a)$ for all $a\in A$, it follows that $c(x)=\sum_{a\in A} c(a)\cdot x(a)\le\sum_{a\in A} c(a)\cdot u(a)<M$. Furthermore, it follows from Theorem 3.20 that $\text{value}(f)\in\mathbb{N}_0$. Thus, $M\cdot\text{value}(f)\in\mathbb{N}_0$ and maximizing the term in Equation (5.1) yields a static flow that maximizes the value of the associated temporally repeated flow and then minimizes the product of costs and transit time for the static flow. □

If the underlying static flow x minimizes the sum $\sum_{a\in A} c(a)\cdot\tau(a)\cdot x(a)$, then this implies that it minimizes the maximal costs over all points in time for the temporally repeated flow (if some additional properties are satisfied).

Lemma 5.15. *We assume the settings stated in Lemma 5.14. If $\tau(p)\le\frac{1}{2}T$ for all $p\in P$, then f is a maximal flow, where the maximal costs for all points in time $\theta\in\{0,\dots,T\}$ are minimized, hence f minimizes*

$$\max\{\, c(f,\theta)\mid\theta\in\{0,\dots,T\}\,\}.$$

Proof. Suppose that x maximizes the term in Equation (5.1). Then, the associated temporally repeated flow f has maximal value and $\sum_{a\in A} c(a)\cdot\tau(a)\cdot x(a)$ is minimized.

Claim: There exists a point in time $\zeta\in\{0,\dots,T\}$ such that, for all $a\in A$,

$$f_a(\zeta-\tau(a)+1)=f_a(\zeta-\tau(a)+2)=\ldots=f_a(\zeta-1)=f_a(\zeta)=x(a).$$

Since $\tau(p)\le\frac{T}{2}$ for all $p\in P$ and the transit times are integer, we derive that $\tau(p)\le\left\lfloor\frac{T}{2}\right\rfloor$ and we set $\zeta:=\left\lfloor\frac{T}{2}\right\rfloor$. Let $(v,w)=a\in A$ and $p\in P$ with $a\in p$, then it follows that $p\in P_a(\zeta-\tau(a)+1)\cap P_a(\zeta-\tau(a)+2)\cap\cdots\cap P_a(\zeta-1)\cap P_a(\zeta)$, hence flow is sent along path p at arc a at the time points $\zeta-\tau(a)+1,\dots,\zeta$. We prove this by checking the definition of the sets $P_a(\theta)$ given in Definition 2.16, where it says that $p\in P_a(\theta)$ if $\tau(p_{[s,v]})\le\theta$ and $\tau(p_{[w,t]})\le T-\theta$ for $\theta\in\{0,\dots,T\}$.

It holds that

$$\tau(p_{[s,v]}) \leq \tau(p) - \tau(a) \leq \left\lfloor \frac{T}{2} \right\rfloor - \tau(a) < \zeta - \tau(a) + 1 < \zeta - \tau(a) + 2 < \ldots < \zeta - 1 < \zeta$$

and

$$\tau(p_{[w,t]}) + \zeta - \tau(a) + 1 < \tau(p_{[w,t]}) + \zeta - \tau(a) + 2 < \ldots < \tau(p_{[w,t]}) + \zeta \leq \left\lfloor \frac{T}{2} \right\rfloor - \tau(a) - \zeta \leq T,$$

hence the inequalities are satisfied and $p \in P_a(\zeta - \tau(a) + 1) \cap P_a(\zeta - \tau(a) + 2) \cap \cdots \cap P_a(\zeta - 1) \cap P_a(\zeta)$. Then, we obtain that $f_a(\zeta - \tau(a) + 1) = f_a(\zeta - \tau(a) + 2) = \ldots = f_a(\zeta) = x(a)$.
Since $f_a(\theta) \leq x(a)$ for $\theta \in \{0, \ldots, T\}$ and via the previous claim, it holds that

$$\begin{aligned} \max\{\, c(f,\theta) \mid \theta \in \{0,\ldots,T\} \,\} &= \max\left\{ \sum_{a \in A} c(a) \cdot \left(\sum_{\substack{\xi = \max \\ \{\theta - \tau(a) + 1,\, 0\}}}^{\theta} f_a(\xi) \right) \,\middle|\, \theta \in \{0,\ldots,T\} \right\} \\ &= \sum_{a \in A} c(a) \cdot \left(\sum_{\xi = \zeta - \tau(a) + 1}^{\zeta} x(a) \right) = \sum_{a \in A} c(a) \cdot \Big(\tau(a) \cdot x(a) \Big) \end{aligned}$$

for $\zeta := \left\lfloor \frac{T}{2} \right\rfloor$. Hence, minimizing $\sum_{a \in A} c(a) \cdot \tau(a) \cdot x(a)$ minimizes the maximal costs over all points in time $\theta \in \{0, \ldots, T\}$. □

If we update the algorithm in Listing 5.1 such that the static flow maximizes the term in Equation (5.1), then it follows from Lemma 5.15 that the updated algorithm yields a temporally repeated flow with maximal values that minimizes the maximal costs over all points in time, see Listing 5.2.

Listing 5.2: Maximal (s,t)-Flows with Minimal Costs over all Points in Time

```
1  Input: Network (G,u,τ,c), source and sink s,t∈V, and time horizon T
2  Output: Temporally repeated flow f with time horizon T
3    with maximal flow and minimized
4    maximal costs over all points in time
5
6  Set d : A → ℕ_0, a ↦ M·τ(a) + c(a)·τ(a)
7  Calculate static (s,t)-flow x that maximizes M·T·value(x) − Σ_{a∈A} d(a)·x(a)
8  Calculate path decomposition y : P → ℝ_{≥0}
9  Construct temporally repeated flow f with time horizon T
10 Return f
```

Similar to Lemma 5.11, we show that the algorithm has polynomial runtime by proving that a static flow that maximizes the Equation (5.1) can be found in polynomial time. This is true for general cost functions $d : A \to \mathbb{N}_0$ which can be computed in polynomial time from u, τ, and c.

Lemma 5.16. *Given a network (G, u, τ, d), a sink $s \in V$, a source $t \in V$, a time horizon T and $M \in \mathbb{N}$. Then, a static flow that maximizes*

$$M \cdot T \cdot \text{value}(x) - \sum_{a \in A} d(a) \cdot x(a)$$

such that there exists a path decomposition $y : P \to \mathbb{R}_{\geq 0}$ with $\tau(p) \leq T$ for all $p \in P$ with $y(p) > 0$ can be computed in polynomial time via a minimal cost flow tranformation.

The proof of this lemma resembles the proof in Lemma 5.11, except that T is replaced by $M \cdot T$ and τ is partially replaced by d.

Proof. Again, it is possible to compute such a static flow via the computation of a minimal cost circulation in an adapted flow network. Let $\overline{G}$ be the graph G extended by an arc (t, s) with capacity $u(t, s) := \infty$ and $d(t, s) := -M \cdot T$. Furthermore, $\overline{x}$ is any circulation in $\overline{G}$ and x the corresponding flow in G, then

$$\begin{aligned} d(\overline{x}) &= -M \cdot T \cdot \overline{x}(t, s) + \sum_{a \in A} d(a) \cdot \overline{x}(a) = -M \cdot T \cdot \overline{x}(t, s) + \sum_{a \in A} d(a) \cdot x(a) \\ &= -M \cdot T \cdot \text{value}(x) + \sum_{a \in A} d(a) \cdot x(a). \end{aligned}$$

We see that a minimal cost circulation regarding d maximizes $M \cdot T \cdot \text{value}(x) - \sum_{a \in A} d(a) \cdot x(a)$ can be computed in polynomial time regarding the size of the network (G, u, τ, d).

Claim: Let $\overline{x}$ be a minimal cost circulation in $\overline{G}$. Then, the corresponding static flow x in G satisfies for every flow decomposition $y : P \cup C \to \mathbb{R}_{\geq 0}$ that

$$\tau(p) \leq T \text{ for all } p \in P \text{ with } y(p) > 0 \quad \text{and} \quad \tau(q) = 0 \text{ for all } q \in C \text{ with } y(q) > 0.$$

We need to extend the proof of the claim a little bit (in comparison to the proof of Lemma 5.11).

- Suppose that $\tau(p) > T$ and $y(p) > 0$ for a path $p \in P$. Then, it follows that $d(p) > M \cdot T$ and $y(p) > 0$, hence there exist backwards arcs $\overleftarrow{p}$ in the residual graph of $\overline{G}$ with costs

  $$d(\overleftarrow{p} \cup (s, t)) = -\underbrace{d(p)}_{< M \cdot T} + M \cdot T < 0.$$

  There exists a negative cycle in the residual graph, thus $\overline{x}$ is not optimal and we have a contradiction.
- Suppose that $\tau(q) \neq 0$ and $y(q) > 0$ for a cycle $q \in C$. Then, either $\tau(q) > 0$, hence also $d(q) > 0$ and there exist backwards arcs $\overleftarrow{q}$ in the residual graph $\overline{G}$ with costs $d(\overleftarrow{q}) < 0$. Thus, there exists a negative cycle, $\overline{x}$ is not optimal and we have a contradiction. If $\tau(q) < 0$, then also $d(q) < 0$, and the contradiction follows directly.

Hence, such a flow can be calculated in polynomial time. □

We consider an application of the algorithm in Listing 5.2 for the problem of the transporta-

tion of beds in hospitals.

Example 5.17. Let us consider a network (G, u, τ, c) that models a hospital where beds need to be transported from the place $s \in V$ to the place $t \in V$. The capacity and the transit time of the pathways in the hospital are modeled by the functions u and τ.

Suppose that we need to send the beds as fast as possible from s to v, but we would like to additionally minimize the number of people working on the transportation. Suppose furthermore that one person is needed to move one bed along a corridor. Then, the maximal number of beds that are in the corridors at the same time is also the number of people that are needed.

If we set the cost function to $c(a) = 1$ for all $a \in A$ then the execution of the algorithm calculates a temporally repeated flow with maximal value (which first and foremost transports as many beds as possible) but meanwhile minimizes the number of working people.

Lastly, we approach the problem of finding a maximal temporally repeated flow with minimal total costs. For a temporally repeated flow f over a network (G, u, τ, c), the costs are

$$\begin{aligned} c(f) &= \sum_{p \in P} c(p) \cdot y(p) \cdot (T - \tau(p)) \\ &= T \cdot \sum_{p \in P} c(p) \cdot y(p) - \sum_{p \in P} c(p) \cdot y(p) \cdot \tau(p) \\ &= T \cdot c(x) - \sum_{p \in P} c(p) \cdot x(p) \cdot \tau(p). \end{aligned}$$

If we want to apply the same technique, we need to find a static flow x that maximizes

$$\begin{aligned} & M \cdot \left(T \cdot \text{value}(x) - \sum_{a \in A} \tau(a) \cdot x(a) \right) + T \cdot c(x) - \sum_{p \in P} c(p) \cdot x(p) \cdot \tau(p) \\ = {} & M \cdot T \cdot \text{value}(x) - M \cdot \sum_{a \in A} \tau(a) \cdot x(a) + T \cdot \sum_{a \in A} c(a) \cdot x(a) - \sum_{p \in P} c(p) \cdot x(p) \cdot \tau(p) \\ = {} & M \cdot T \cdot \text{value}(x) - \sum_{a \in A} \left(M \cdot \tau(a) - T \cdot c(a) \right) \cdot x(a) - \underbrace{\sum_{p \in P} c(p) \cdot x(p) \cdot \tau(p)}_{\text{Technique not applicable.}}, \end{aligned}$$

but this term depends on the path decomposition of the static flow x. It is not possible to maximize such a term via the technique in Lemma 5.11 and Lemma 5.16 and hence this approach fails.

6 Conclusion

Based on the problem of transporting freshly made beds in a hospital, we analyzed different types of flows over time and the restriction of common research problems on those sets of flows. Those special flows aimed to represent solutions that are as structured as possible, hence enabling humans to understand and remember the represented transportation plans as fast as possible, but also as optimal as possible (depending on the respective research problem). We proceeded to analyze the integrality of the problems and proved that the Quickest Transshipment Problem can be efficiently solved for general flows over time if solutions of the Min Cost Flow over Time Problem can be computed in polynomial time.

Then, we considered the Quickest Transshipment Problem for k-uniform flows on trees (where k is at most the number of sinks h) and proposed a naive algorithm that computes a solution that fills one sink per subflow in polynomial time. We enhanced the algorithm and obtained lower and upper bounds for the optimal solution. Next, we analyzed tree networks and proved that each tree network is equivalent to an almost-binary tree network, that allows us to consider only almost-binary tree networks for our research. We showed how to calculate optimal solutions for small almost-binary tree networks. For larger networks, this calculation method seems very inefficient. Unfortunately, it is also not evident how to calculate the optimal solution of a large network from the optimal solutions of smaller parts of the network in a greedy way. We went on to analyze the linear relaxation of the problem and gave an algorithm which allows us to calculate an integer solution from a solution of the relaxation. This solution is an optimal load-consistent k-uniform flow (meaning that the load to each sink is fixed until it drops to zero), but not an optimal k-uniform flow.

In the last part, we proved an optimality criterion for static flows with lexicographic costs that is an adaptation of the theorem by Klein on static flows. Then, we analyzed the algorithm by L.R. Ford and D.R. Fulkerson for finding maximal temporally repeated flows from a single source to a single sink [FJF58]. We extended the algorithm such that it allows us to create maximal temporally repeated flows where the maximal costs over all points in time are minimized. This algorithm could be used to create a transportation plan in a hospital that is as fast as possible but engages as few employees as possible.

For the future, there are many interesting questions that allow for further research. Since the structure of hospitals is usually not a tree, it would be crucial to explore the Quickest Transshipment Problem for k-uniform flows on differently structured graphs, such as series-parallel graphs, cliques, and general graphs. Here, it could be promising to consider the enhanced naive algorithm in Listing 4.2 and analyze how it could be executed on different types of graphs.

The transformation of tree networks could be refined such that the network is transformed

into an equivalent almost-binary tree network with minimal capacities. For this purpose, we need to define another operation that reduces the capacities if possible. This could enhance the runtime of algorithms whose complexity depends on the capacities of the arcs.

For almost-binary trees, we could analyze the solution sets and search for criteria that filter all solutions which might be relevant for the construction of solutions on larger networks. This might enable us to create a greedy-like algorithm for the computation of optimal solutions on tree networks of arbitrary sizes.

Furthermore, we could analyze application scenarios for the algorithm for maximal temporally repeated flows with minimized costs at each point in time. Currently, we use the costs $a \mapsto M \cdot \tau(a) + c(a) \cdot \tau(a)$ (for arcs $a \in A$), but we could also consider other cost functions like $a \mapsto M \cdot \tau(a) + c(a)$, $a \mapsto M \cdot \tau(a) + \tau(a)$, and $a \mapsto M \cdot \tau(a) + \frac{c(a)}{\tau(a)}$, and observe their meanings and applications.

Bibliography

[BLMN10] Alexandre Beaudry, Gilbert Laporte, M. Teresa Melo, and Stefan Nickel. Dynamic transportation of patients in hospitals. *OR Spectr.*, 32(1):77–107, 2010.

[BS67] Robert G. Busacker and Thomas L. Saaty. *Finite Graphs and Networks: An Introduction with Applications*. 1967.

[Bue22] Christina Buesing. Flows over time. Unpublished, January 2022. Lecture Notes Theorie der Graphen- und Netzwerkoptimierung.

[FJF58] L. R. Ford Jr. and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Oper. Res.*, 6(3):419–433, 1958.

[FJF62] D. R. Ford Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, USA, 1962.

[Fle01] Lisa Fleischer. Faster algorithms for the quickest transshipment problem. *SIAM Journal on Optimization*, 12(1):18–35, jan 2001.

[Fol99] Gerald B. Folland. *Real analysis: modern techniques and their applications*, volume 40. John Wiley & Sons, USA, 1999.

[FS03] Lisa Fleischer and Martin Skutella. Minimum cost flows over time without intermediate storage. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 66–75. ACM/SIAM, 2003.

[Ger21] Thomas Gerlinger. Krankenhäuser in Deutschland (Strukturen - Probleme - Reformen). *Aus Politik und Zeitgeschichte/bpb.de*, 2021.

[GKL+18] Corinna Gottschalk, Arie M.C.A. Koster, Frauke Liers, Britta Peis, Daniel Schmand, and Andreas Wierz. Robust flows over time: models and complexity results. *Math. Program.*, 171(1-2):55–85, 2018.

[Gö14] Uschi Götz. Am Ende leiden die Patienten. *Deutschlandfunk*, November 2014.

[Ham89] Horst W. Hamacher. Temporally repeated flow algorithms for dynamic min cost flows. In *Proceedings of the 28th IEEE Conference on Decision and Control,*, pages 1142–1146 vol.2, 1989.

[HT00] Bruce Hoppe and Éva Tardos. The quickest transshipment problem. *Math. Oper. Res.*, 25(1):36–62, 2000.

[Kle67] Morton Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Institute for Operations Research and the Management Sciences (INFORMS)*, 14:205–220, 1967.

[KW04] Bettina Klinz and Gerhard J. Woeginger. Minimum cost dynamic flows: The series-parallel case. *Networks*, 43(3):153–162, 2004.

[PBR17] Maxime Painchaud, Valérie Bélanger, and Angel Ruiz. Discrete-event simulation of an intrahospital transportation service. In Paola Cappanera, Jingshan Li, Andrea Matta, Evren Sahin, Nico J. Vandaele, and Filippo Visintin, editors, *Health Care Systems Engineering*, pages 233–244, Cham, 2017. Springer International Publishing.

[Sch98] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998.

[Sku09] Martin Skutella. *An Introduction to Network Flows over Time*, pages 451–482. Springer, Berlin, Heidelberg, 2009.

[Sla22] Angelika Slavik. Suche Krankenpfleger, biete Gesetz. *Süddeutsche Zeitung*, July 2022.

[SS14] Melanie Schmidt and Martin Skutella. Earliest arrival flows in networks with multiple sinks. *Discret. Appl. Math.*, 164:320–327, 2014.

[Wil19] D.P. Williamson. *Network Flow Algorithms*. Cambridge University Press, 2019.