Path-Constrained Network Flows

Dipl.-Math. oec. Maren Martens
aus Berlin

Vom Fachbereich Mathematik
der Universität Dortmund
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Berichter: Prof. Dr. Martin Skutella
Prof. Dr. Friedrich Eisenbrand

Tag der Disputation: 19. April 2007

Contents

Introduction

Holidays! Grab the tent and the sleeping back, jump into the car, and off to coast! Finally, you get your deserved holidays! Looking forward to a joyful time of vacation, you take the first miles of the highway. At long last you got some days of recreation! But then the traffic becomes heavier and heavier and finally you are caught in a jam. Annoyance! Is that how you planned to spend your holidays? Have you really wanted to spend hours and hours on the road watching other people arguing in their cars? Watching bored truckers or sleeping children? No, probably not! Now you maybe start wondering if there are no better possibilities to channel the flood of tourists and trucks. The radio communication does obviously not perform its task. There must be an alternative to avoid situations like this. Is there nobody who thinks about this? Nobody who cares if you are at the end of your rope?

The answer is: Sure! There is someone who cares about you being caught in the jam. There is someone who would like to eliminate every single jam that appears. But unfortunately, the methods to do this are not yet fully developed. Devices, such as navigation systems, are not yet as widely spread as they could be, in particular not those which take the current traffic situation into account. But a fair amount of research is done in this area and people involved in this research still hope to be able to better the conditions of transportation soon.

From the theoretical point of view, this is where network flows come into play. The network that is considered in this application is that of streets and highways. Enhanced examinations could as well include railways and ship or flight paths. Different streets are connected with each other by crossings. In a mathematical network every single part of a street—that between two different crossings—is presented as an arc. Crossings are translated into nodes. Flow in a mathematical network is routed from specified start nodes to specified destination nodes. A network flow itself is simply a function on the arcs that indicates how many units of flow are routed along every arc. A unit of flow could, for example, be a car or a truck, or maybe only a single passenger, or a single container of goods.

Certainly, the model of network flows finds more applications than that of road traffic. It is also useful to abstract data traffic, communication networks in general, or, e.g., production chains. Even scheduling problems can be

modeled using network flows. Anyway, in the following we stay with the example of road traffic for illustration of the theoretical constructs.

Network flows have been studied since the 1950s. The first considerations aimed for finding maximum flows in networks with arc capacities and only one start and one destination node [35]. Assigning capacities to the arcs is reasonable from the practical point of view, because traffic jams originate exactly from the fact that the throughput of a road is limited. The point is that the number of cars that can use a specific road without any blockage is confined per time unit, no matter if this time is chosen to be a minute, an hour, or even the life time of the road. The wider a road is the more cars can use it per time unit. For the following let the considered time unit be a day. Then the first question studied in network flow theory was the one of how many units of flow (e.g., cars) can be sent from one specified start node to one specified destination node within one day while obeying the road capacities.

Starting from this point, extensions of the basic model have been developed. These cover problems with several start and destination nodes or problems with side constraints on the paths. Involving several start and destination nodes makes the model more realistic from the view of transportation. A network of roads is usually not only used to transport persons/goods from A to B, but also to transport—at the same time—some other persons/goods from C to D, A to D, or whatsoever. Side constraints on the paths could, e.g., be length bounds. In length-bounded flow problems, we assume to have a length function on the arcs, meaning, e.g., that we specify for each arc how much time it takes to travers it. Sometimes goods have to reach their destination soon. This could be the case in the transportation of perishable goods or drugs that are indispensable to life. Specifying a length bound in a network flow means that no good may be routed along a path that exceeds the given time bound.

Length bounds on the paths along which flow is routed yield one side constraint that we consider in this thesis. But the main requirement for most parts of this thesis is to limit the number of paths that may be used to route things from one start node to a destination. Think of a vendor who possesses only a finite fleet of trucks to distribute goods. In this case goods may not be split into arbitrarily many groups to be sent to their destination. The number of trucks gives an upper bound on the number of chunks into which the goods are decomposed. Each truck cannot use more than one path to get from A to B. Therefore, the number of trucks bounds the number of paths that may be used to route the goods. The corresponding theoretical problem in which an upper bound for the number of flow carrying paths is given is named the *k-splittable flow problem*. In the special case that we

have even only one truck to route the goods, we obtain the *unsplittable flow problem*. These problems are considered in Chapters 2 and 3. In Chapter 3 we even combine the length-bounded and the k-splittable flow problem, meaning that we look for flows that use at most k paths that are not longer than a given bound. We also take care of the capacities of trucks. In Chapter 3 we consider the problem in which each of the k paths that are used to route a commodity carries no more units of flow than the corresponding truck can dispatch. A similar constraint is considered in Chapter 4 where we search for maximum flows that route at most one unit of flow along each path.

All of the problems just stated are so called *NP-hard* problems. This means in colloquial language that they are so hard to solve that an optimal solution could probably not be found before it is needed. For our examples this means that we could probably not find an optimal way to deliver all goods before they are spoilt or before the person who needed the drug, which was indispensable for his life, died. Therefore, we forbear from computing optimal solutions to the considered problems and settle for solutions that are provably close to optimum and faster to compute.

Overview of the Thesis

Chapter 1. We start with a short survey on basic notation, definitions, and algorithms in network flow theory. We introduce the s-t-flow problem that is the origin of network flows and then continue with minimum cost flows that are a tool for us in subsequent chapters. We also give an introduction to multicommodity flows whose study pervades the whole thesis. Furthermore, we shortly introduce dynamic flows. These are not studied in detail in this thesis, but mentioned in Chapter 3.

Chapter 2. Classical network flow theory allows decomposition of flow into several chunks of arbitrary sizes traveling through the network on different paths. In Chapter 2 we consider the unsplittable flow problem where all flow traveling from a source to a destination must be sent on only one path. We prove a lower bound of $\Omega(\log m / \log \log m)$ on the performance of a general class of algorithms for minimizing the congestion of an unsplittable flow, where m is the number of arcs in the underlying network. The algorithms in this class start with a solution for the classical (fractional) multicommodity flow problem, compute a path decomposition, and select one of its paths for each commodity in order to obtain an unsplittable flow. Our lower bound matches the best known upper bound for randomized rounding—an algorithm from the mentioned class that was introduced by Raghavan and

Thompson [88].

In the single source multicommodity flow problem commodities must be routed from a common source to different sinks. This scenario is pretty well known from real life where wholesalers must send goods to their customers or a post office wants to deliver parcels. In this context it seems to be also realistic to consider the case that any commodity must be routed unsplittably, i.e., on a single path. We present an interesting connection between fractional single source multicommodity flows and unsplittable single source multicommodity flows, namely that each single source multicommodity flow can be represented as a convex combination of unsplittable flows that have congestion at most 2. It has already been known [26] that an arbitrary single source multicommodity flow can be made unsplittable by losing only a factor 2 with respect to congestion. Our result gives a positive answer to the question whether, for an arbitrary cost function on the arcs, there also exists a suitable unsplittable flow whose congestion is only twice as large as the original one and whose costs do not exceed the costs of the original (fractional) flow. The proof of our result is constructive in the sense that it involves an algorithm which computes the required unsplittable flows. We also show that the congestion of 2 for the unsplittable flows is tight.

Chapter 3. In many applications users are allowed to use a limited number of paths to route commodities rather than to use only exactly one path. The k-splittable flow problem is a generalization of the unsplittable flow problem where the number of paths is bounded for each commodity. We study a new variant of this problem with additional constraints on the amount of flow being sent along each path. We present approximation results for two versions of this problem with the objective to minimize the congestion of the flow. The key idea is to reduce the problem under consideration to an unsplittable flow problem while only losing a constant factor in the performance ratio.

Classical network flow problems do not impose restrictions on the choice of paths on which flow is sent. Only the arc capacities of the network have to be obeyed. This scenario is not always realistic. In fact, there are many problems for which, e.g., the number of paths being used to route a commodity or the length of such paths has to be small.

These restrictions are considered in the length-bounded k-splittable s-t-flow problem: The problem is a variant of the well known classical s-t-flow problem with the additional requirement that the number of paths that may be used to route the flow and the maximum length of those paths are bounded.

Our main result is that, for any $\epsilon > 0$, we can efficiently compute a length-bounded k-splittable s-t-flow which sends a factor $(1-\epsilon)/2$ of the maximum

flow value while exceeding the length bound by at most a factor $1/\epsilon$. We also show how this result leads to a constant factor approximation algorithm for dynamic k-splittable s-t-flows.

Chapter 4. Classical network flow theory does not impose any restrictions on the amount of flow being sent along each path. Nevertheless, it might be undesired in real life that the flow value of the paths exceeds some given bound. This could be due to the unsteadiness of some paths whose effects can be decreased when using many different paths.

In Chapter 4 we study the maximum one flow problem which is identical to the classical network flow problem except that it bounds the flow value of any path by one. We consider the single commodity as well as the multicommodity case. Both cases are NP-hard, but admit an FPTAS in the fractional case. In the integral case, the single commodity problem is APX-hard, whereas it is even NP-hard to approximate the multicommodity case within a factor $O(m^{1/2-\epsilon})$, for any $\epsilon > 0$. We present a randomized $O(\log m)$-approximation algorithm for the single commodity case and a randomized $O(\sqrt{m})$-approximation algorithm for the multicommodity case. The latter of these approximation ratios is best possible, unless P = NP.

Integrating demands for each commodity in the multicommodity case, we obtain an FPTAS for the problem of minimizing the congestion of a fractional one-flow satisfying all demands. Further, we state how an $O(\log m / \log \log m)$-approximation can be found for the corresponding integral problem.

Moreover, Chapter 4 introduces two interesting combinatorial problems that are closely related to the maximum one flow problem. We use this relationship to prove NP-hardness for both of these problems.

Chapter 5. We conclude with an overview of the problems considered in Chapters 2 to 4 in the context of abstract flows. Abstract flows depart from the structure of graphs and base on some abstract buildings called *paths*. These paths are ordered subsets of given *edges* in which, however, the particular order may be different for all subsets.

Most parts of Chapters 2, 3, and 4 are from joint work with Martin Skutella. Fruitful discussions for parts of Section 2.4 were also stimulated by Fernanda Salazar.

Acknowledgements

First of all I thank Martin Skutella for offering me the possibility to work in his group and for supervising my thesis. I am grateful to him for his advice and for having so much time for discussions. Furthermore, I thank Fritz Eisenbrand for the willingness to take the second assessment of this thesis.

Many thanks also to Joachim Reichel, Martin Hadac, Sabine Piana, Martin Martens, and Peter Hachenberger for their careful proof-reading of several parts of the manuscript and for their helpful comments. Further thanks are addressed to my roommates Nadine Baumann and Daniel Dressler for useful discussions on stylistic topics concerning this thesis.

Chapter 1

Preliminaries

1.1 Introduction

This chapter provides a survey on basic notation and algorithms in network flow theory. It is not meant to be an introduction to the whole theory of network flows, but rather introduces the reader shortly to the problems considered in this thesis. We start defining some basic notions and then pass into an overview on network flows. For those who desire a more detailed introduction to the theory of network flows, we refer to [2, 68, 93]. A comprehensive compendium of complexity theory can be found in [39]. For more information about approximation algorithms in combinatorial optimization we suggest the book by Hochbaum [51].

1.2 Basic Vocabulary and Notation

Throughout this whole thesis we consider *directed graphs* (which we also refer to as *digraphs*). A digraph is a pair $D = (V, A)$ consisting of a set of *vertices* V (also called *nodes*) and a family of *arcs* $A \subseteq V \times V$. Sometimes we refer to so called *(undirected) graphs* that are usually denoted by $G = (V, E)$ where V is again a set of vertices, but E is a family of *edges* which are unordered subsets of V consisting of exactly two elements. For different problems, there exist known transformations between the undirected and the directed case. To transform an undirected graph into a directed one for the study of network flows, it is useful to replace each edge $\{u, v\}$ (sometimes also denoted as (u, v)) by two new vertices u_1, u_2 and five arcs (u, u_1), (v, u_1), (u_2, u), (u_2, v), and (u_1, u_2) (see also Figure 1.1). Here, we only study digraphs, and therefore, give the following definitions only for those. But generally, each algorithm that is designed for digraphs can also be used for undirected graphs by transforming the undirected graph into a directed one as described above. For negative results, i.e., counter examples, the transfer works only in the opposite way.

Usually, we refer to the number of nodes in a graph by $n := |V|$ and to the number of arcs by $m := |A|$.

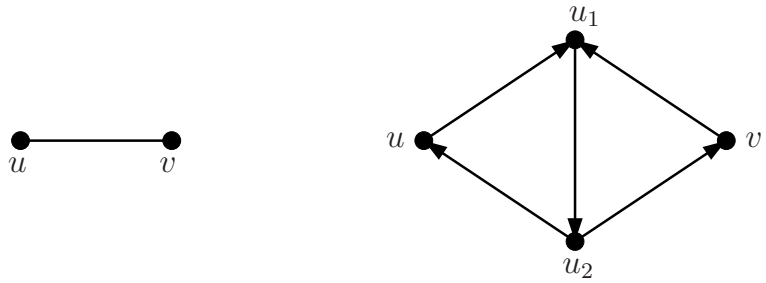

Figure 1.1: Transformation of an undirected graph into a digraph.

For any vertex $v \in V$, we define $\delta^+(v) := \{a \in A \mid a = (v, w) \text{ for some } w \in V\}$ as the set of outgoing arcs from v. The set of incoming arcs to v is given by $\delta^-(v) := \{a \in A \mid a = (w, v) \text{ for some } w \in V\}$. For a set of vertices $V' \subseteq V$, the set $\delta^+(V') := \{a \in A \mid a = (v, w) \text{ for some } v \in V',\ w \in V \setminus V'\}$ is that of arcs leaving V' and the set $\delta^-(V') := \{a \in A \mid a = (v, w) \text{ for some } v \in V \setminus V',\ w \in V'\}$ is that of arcs entering V'.

Two arcs (v, u) and (v', u') are called *parallel*, if $v = v'$ and $u = u'$. A *path* in a digraph is a sequence of arcs $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ $(k \in \mathbb{N})$ where $(v_{i-1}, v_i) \in A$, for all $i = 1, \ldots, k$. For any two vertices $s,\ t \in V$, an *s-t-path* is a path for which $v_0 = s$ and $v_k = t$. A *cycle* is a path for which $v_0 = v_k$. Using the notion "path" we mean a path that is not a cycle. If we consider both, it will be mentioned explicitly. A path is called *simple* if the vertices contained in it are mutually distinct.

Often we consider functions on the arcs. A very popular function is a *cost* (or *length*) *function* $c : A \to \mathbb{R}_0^+$. For any nodes $s,\ t \in V$, a *shortest s-t-path* (according to c) is an s-t-path P such that $\sum_{a \in P} c(a)$ is minimal among all s-t-paths. The value $\sum_{a \in P} c(a)$ is also called P's *length* or its *cost*. In the context of network flows, we also consider capacity functions $u : A \to \mathbb{R}^+$ on the arcs. A digraph with a capacity function and at least one specific "source-sink-pair" (s, t) is called a *network*.

It is also common to consider length functions not only mapping to nonnegative values. But for this thesis we do not need this general case. For nonnegative length functions, a standard algorithm to find a shortest s-t-path is the labeling algorithm by Dijkstra [25], that runs in $O(n^2)$ time. This algorithm even computes shortest paths from s to each other node in the considered digraph. To compute shortest paths between all pairs of nodes, it is popular to apply the algorithm by Moore, Bellman, and Ford [14, 34, 80], that runs in $O(nm)$ time. This algorithm additionally allows negative arc lengths, as long as these do not imply negative cycles.

A *cut* in a digraph $D = (V, A)$ is a set of nodes $V' \subseteq V$. If we are additionally given a capacity function u on the arcs, we can define the capacity

of a cut V' as $u(V') := u(\delta^+(V')) := \sum_{a \in \delta^+(V')} u(a)$. It is also common to regard a cut as the set of arcs $\delta^+(V')$ instead of considering it as the set of nodes V'. Given $s,\ t \in V$, a cut V' is called an *s-t-cut*, if $s \in V'$ and $t \in V \setminus V'$. A *minimum (s-t-)cut* is a(n) (s-t-)cut of minimum capacity.

In this thesis, we mainly consider optimization problems. In an *optimization problem* we are given an *objective function* that is either to be minimized or to be maximized and side *constraints* that need to be obeyed by a *feasible solution* to the problem. An *optimal solution* is a feasible solution whose objective function value (also called *objective value*) is optimal (either minimal or maximal, depending on the underlying objective). An *instance* of a problem is a structure consisting of specified input data for the considered problem. We say that a problem can be solved *efficiently*, if there exists an algorithm for it that runs in time polynomial in the input size. If a problem cannot be solved efficiently, it is often still possible to give an *approximation algorithm*, i.e., an algorithm which runs in polynomial time (i.e., polynomial in the input size) and outputs a feasible, but not necessarily optimal solution. For any $\rho > 1$, a *ρ-approximation algorithm* is one that outputs a *ρ-approximation* for the underlying problem. A *ρ-approximation* is a feasible solution whose objective value v is at most a factor ρ away from the optimal objective value v^*, i.e., $v \leq \rho v^*$ for a minimization problem and $v \geq (1/\rho)v^*$ for a maximization problem. A *polynomial time approximation scheme* (PTAS) is an algorithm that, given a parameter $\epsilon > 0$, outputs a $(1+\epsilon)$-approximation. The running time of a PTAS is required to be polynomial in the input size for every fixed ϵ, but can depend arbitrarily on ϵ. If the runtime of such an algorithm is polynomial in the input size and $1/\epsilon$, we call it a *fully polynomial time approximation scheme* (FPTAS).

1.3 Network Flows

Here, we give a short introduction to different network flow problems. After an overview on the basic s-t-flow problem in Section 1.3.1, we shortly discuss the minimum cost flow problem in Section 1.3.2. In Section 1.3.3 we introduce multicommodity flows. These form the main subject of this thesis. The dynamic flow problem defined in Section 1.3.4 is of minor importance for our work, but will appear in Section 3.5 and is therefore also shortly introduced here.

1.3.1 s-t-Flows

Given a digraph $D = (V, A)$, a capacity function $u : A \to \mathbb{R}^+$, and specified nodes $s,\ t \in V$, an *s-t-flow* is a function $f : A \to \mathbb{R}_0^+$ that obeys all arc capacities, i.e., $f(a) \leq u(a)$ for all $a \in A$, and meets the flow conservation requirement, i.e., $\sum_{a \in \delta^+(v)} f(a) = \sum_{a \in \delta^-(v)} f(a)$ for all $v \in V \setminus \{s, t\}$. The *value* of an *s-t*-flow is given by $|f| := \sum_{a \in \delta^+(s)} f(a) - \sum_{a \in \delta^-(s)} f(a)$. A *maximum s-t-flow* is an *s-t*-flow of maximum value. Formulated as a linear program (LP), the maximum *s-t*-flow problem is the following.

$$\begin{aligned}
\max \quad & \sum_{a \in \delta^+(s)} f(a) - \sum_{a \in \delta^-(s)} f(a) & \\
\text{s.t.} \quad & \sum_{a \in \delta^+(v)} f(a) - \sum_{a \in \delta^-(v)} f(a) = 0 & \forall\ v \in V \setminus \{s, t\} \\
& 0 \leq f(a) \leq u(a) & \forall\ a \in A
\end{aligned}$$

Ford and Fulkerson [35, 37] proved that in every network the value of a maximum *s-t*-flow equals the capacity of a minimum *s-t*-cut. This result is known as the *Max-Flow-Min-Cut Theorem*. They also gave an important procedure to find a maximum *s-t*-flow known as the Ford-Fulkerson algorithm. This procedure will not necessarily terminate if the arc capacities are not integral. In the case that $u(a) \in \mathbb{N}$ for all $a \in A$, it terminates with a maximum *s-t*-flow f after $O(|f|m)$ time. The Ford-Fulkerson algorithm was revised by Edmonds and Karp [28] who improved its runtime to $O(m^2 n)$. If all arc capacities are integral, both methods, the one by Ford and Fulkerson and the algorithm by Edmonds and Karp, compute a maximum *s-t*-flow that is integral.

The procedure by Ford and Fulkerson also proves that any *s-t*-flow can be decomposed into flows on at most m different *s-t*-paths and cycles. Since the cycles do not provide any additional flow from s to t, they can be omitted and we obtain a path decomposition of the flow. Computing an *s-t*-flow on paths is equivalent to solving the following LP where $\mathcal{P}$ is the set of all *s-t*-paths.[1]

$$\begin{aligned}
\max \quad & \sum_{P \in \mathcal{P}} x_P & \\
\text{s.t.} \quad & \sum_{\substack{P \in \mathcal{P}: \\ a \in P}} x_P \leq u(a) & \forall\ a \in A \\
& x_P \geq 0 & \forall\ P \in \mathcal{P}
\end{aligned}$$

[1] In LP based path formulations of network flows, we prefer to name the variables x.

Here, x_P denotes the *flow value* of $P \in \mathcal{P}$. Note that any s-t-flow in path variables directly translates to an s-t-flow in arc variables, too. Thus, it is always possible to switch between arc and path formulations of s-t-flows immediately.

1.3.2 Minimum Cost Flows

In a minimum cost flow problem (also called *min cost flow problem*), we are generally given multiple sinks and sources. It is common to define a function $b : V \to \mathbb{R}$ on the nodes with $\sum_{v \in V} b(v) = 0$ that represents the supply or the demand of a node. A node $v \in V$ is called a *source*, if it has a supply, i.e., $b(v) > 0$, or a *sink*, if it has a demand, i.e., $b(v) < 0$. Furthermore, we are given a cost function $c : A \to \mathbb{R}$ on the arcs.

Then the problem can be stated as follows.

$$\begin{aligned} \min \quad & \sum_{a \in A} c(a) f(a) & \\ \text{s.t.} \quad & \sum_{a \in \delta^+(v)} f(a) - \sum_{a \in \delta^-(v)} f(a) = b(v) & \forall\, v \in V \\ & 0 \leq f(a) \leq u(a) & \forall\, a \in A \end{aligned}$$

A feasible solution to this linear program is called a *b-flow*, an optimal solution is a *min cost flow*. The feasibility can be checked in polynomial time using a maximum s-t-flow computation.

Among the first algorithms to solve the min cost flow problem was the cycle cancelling method which is a result of an optimality criterion by Klein [58]. This method does not necessarily terminate, but was revised by Goldberg and Tarjan [42] who created the minimum mean cycle-cancelling algorithm, which runs in $O(m^3 n^2 \log n)$ time. If there are no negative cycles with respect to the arc costs, another important algorithm to solve the min cost flow problem will be the successive shortest path algorithm. This is a result of an optimality criterion by Jewell [57], Iri [55], and Busacker and Gowen [19]. Edmonds and Karp [28] showed that for integral arc capacities and supplies this algorithm can be implemented with a running time of $O(nm+B(n^2+m))$ where $B := (\sum_{v \in V} |b(v)|)/2$. Since this runtime is only pseudo-polynomial, the algorithm was revised several times. One revision was given by Orlin [83] and runs in time $O(n \log m(n^2 + m))$.

1.3.3 Multicommodity Flows

In addition to a digraph $D = (V, A)$ and arc capacities $u : A \to \mathbb{R}^+$, a multicommodity flow problem consists of source-sink-pairs (s_i, t_i) for $i = 1, \ldots, K$, for some $K \in \mathbb{N}$. A source-sink-pair (s_i, t_i) is also called ith *request* or ith *commodity*. We usually denote the set of s_i-t_i-paths by $\mathcal{P}_i$ and define $\mathcal{P} := \bigcup_{i=1}^{K} \mathcal{P}_i$.

A *maximum multicommodity flow* is an optimal solution to the following LP. A feasible, but not necessarily optimal solution is called a *multicommodity flow*.

$$\begin{aligned} \max \ & \sum_{i=1}^{K} \sum_{P \in \mathcal{P}_i} x_P & & \\ \text{s.t.} \ & \sum_{i=1}^{K} \sum_{\substack{P \in \mathcal{P}_i: \\ a \in P}} x_P \leq u(a) & & \forall\ a \in A \\ & x_P \geq 0 & & \forall\ P \in \mathcal{P} \end{aligned}$$

For multicommodity flows, we prefer the pathwise description, but it is also possible to define a multicommodity flow arcwise by using different flows for different commodities. Using $\mathcal{P} = \bigcup_{i=1}^{K} \mathcal{P}_i$, we can shorten the given LP formulation. (The resulting LP looks exactly like the one given in Section 1.3.1.) Note the difference of a multicommodity flow and a b-flow where we have only a single commodity with multiple sources and sinks.

We can also have a demand d_i for each commodity i. In this case we use the notion "multicommodity flow", although a flow does not necessarily obey all arc capacities. Such a flow has a *congestion*. This is the minimum value $\alpha \geq 1$ by which the arc capacities must be scaled to be met. We consider the *problem to minimize the congestion* that can be formulated by the following LP.

$$\begin{aligned} \min \ & \alpha & & \\ \text{s.t.} \ & \sum_{\substack{P \in \mathcal{P}: \\ a \in P}} x_P \leq \alpha u(a) & & \forall\ a \in A \\ & \sum_{P \in \mathcal{P}_i} x_P = d_i & & \forall\ i = 1, \ldots, K \\ & x_P \geq 0 & & \forall\ P \in \mathcal{P} \\ & \alpha \geq 1 & & \end{aligned} \tag{1.1}$$

Using this LP formulation, the congestion of a flow x can be defined as the minimum value $\alpha \geq 1$ such that all inequalities from (1.1) are obeyed.

The dual linear programs of the problems above can be separated in polynomial time. It follows from a result by Grötschel, Lovasz, and Schrijver [47] that the dual problems can also be solved in polynomial time. By the theory of linear programming (see, e.g., Schrijver [92]) it follows that we can also solve the primals in polynomial time. To avoid solving linear programs one can use the FPTAS given by Garg and Könemann [40] or the revision by Fleischer [33] for either of the problems.

1.3.4 Dynamic Flows

So far we have only considered *static network flows*. All the problems stated here can also be transfered to a dynamic scenery. For a dynamic flow problem we have *transit times* in addition, i.e., a function $\tau : A \to \mathbb{R}_0^+$. The transit time of a path P in the digraph D is defined as $\tau(P) := \sum_{a \in P} \tau(a)$.

For two vertices $s,\ t \in V$, a *dynamic s-t-flow* is a function $f : A \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$, where $f(a, \theta)$ indicates the amount of flow that enters arc a at time θ. This function must obey all arc capacities, i.e., $f(a, \theta) \leq u(a)$ for all $a \in A, \theta \in \mathbb{R}_0^+$. Further, it must meet the flow conservation requirement, i.e., for all $v \in V \setminus \{s, t\},\ \theta > 0$,

$$\sum_{a \in \delta^+(v)} \int_0^\theta f(a, x)dx = \sum_{a \in \delta^-(v)} \int_0^{\theta - \tau(a)} f(a, x)dx \qquad .$$

A dynamic s-t-flow has *time horizon* $T \geq 0$, if $f(a, \theta) = 0$ for all $a \in A,\ \theta \geq T - \tau(a)$. The *value* of such a flow is

$$|f| := \sum_{a \in \delta^+(s)} \int_0^T f(a, \theta)d\theta - \sum_{a \in \delta^-(s)} \int_0^{T - \tau(a)} f(a, \theta)d\theta \qquad .$$

The standard method to compute a *maximum dynamic s-t-flow*, i.e., a dynamic s-t-flow of maximum value, for a given time horizon is an algorithm by Ford and Fulkerson [36] which primarily reduces the problem to a static min cost flow problem with arc costs τ.

Chapter 2

Unsplittable Flows

2.1 Introduction

In classical network flow theory, flow being sent from a source to a destination may be split into a large number of chunks of arbitrary sizes traveling on different paths through the network. This effect is undesired or even forbidden in many applications. For this reason we consider the unsplittable flow problem (UFP) which was introduced by Kleinberg [61]: Given a network with capacities on the arcs and several source-sink-pairs (commodities) with associated demand values, route the demand of each commodity on exactly one path leading from its source to its sink without violating arc capacities. For the special case of unit capacities and unit demands, we get the arc-disjoint paths problem which is well-known to be NP-complete [39]. Kleinberg [61] specified the following three optimization versions of the unsplittable flow problem. *Minimum congestion:* Find the smallest value $\alpha \geq 1$ such that there exists an unsplittable flow that violates the capacity of any arc at most by a factor α. *Minimum number of rounds:* Partition the set of commodities into a minimum number of subsets (rounds) and find a feasible unsplittable flow, i.e., one that obeys all arc capacities, for each subset. *Maximum routable demand:* Find a feasible unsplittable flow for a subset of demands maximizing the sum of demands in the subset. Here, we are mainly interested in the minimum congestion problem.

Related Results from the Literature

The unsplittable flow problem has been well studied in the literature. Raghavan and Thompson [88, 87] introduce a randomized rounding technique which gives an $O(\log m / \log \log m)$-approximation algorithm for the minimum congestion problem provided that the maximum demand is bounded from above by the minimum arc capacity (the *balance condition*).[1] For any instance of the problem, their technique first solves the related (fractional) multicom-

[1] Unless stated otherwise, the balance condition is always assumed to be met for the UFP.

modity flow problem and then chooses exactly one of the occuring flow paths for each commodity at random, such that each flow path is chosen with probability equal to its flow value divided by the demand of its commodity. Their procedure even yields a constant factor approximation if either the ratio of the minimum arc capacity and the maximum demand or the congestion of an optimal fractional routing is at least $\Omega(\log m)$. For undirected graphs Leighton, Rao, and Srinivasan [71] prove that the performance ratio of randomized rounding is in $\Omega(\log m/\log\log m)$. Their proof can also be adapted to the directed case, but in this case they only give a proof for an integrality gap of $\Omega(\log m/\log\log m)$. Chuzhoy and Naor [24] show that the directed case of the UFP is $\Omega(\log\log m)$-hard to approximate unless $NP \subseteq DTIME(n^{O(\log\log\log n)})$.[2] Before this result was found, only APX-hardness for the UFP was known (see, e.g., Kleinberg [62]). In the special case of unit demands and unit edge capacities (the *edge-disjoint paths problem*) Andrews and Zhang [3] very recently proved that there is no $(\log\log m)^{1-\epsilon}$-approximation for the undirected congestion minimization problem, unless $NP \subseteq ZPTIME(n^{polylog\ n})$.[3]

For the maximum routable demand problem, Azar and Regev [5] present a strongly polynomial algorithm with approximation ratio $O(\sqrt{m})$. Kolman and Scheideler [67] even give a strongly polynomial $O(\sqrt{m})$-approximation algorithm for the problem without the balance condition. These results also translate to the directed setting. On the other hand, Guruswami, Khanna, Rajaraman, Shepherd, and Yannakakis [48] show that in the directed case there is no approximation algorithm with performance ratio $O(m^{\frac{1}{2}-\epsilon})$ for any $\epsilon > 0$, unless $P = NP$. To get better approximation results one has to incorporate additional graph parameters into the bound. Baveja and Srinivasan [13] develop a rounding technique to convert an arbitrary solution to an LP-relaxation into an unsplittable flow within a factor of $O(\sqrt{m})$ or $O(d)$ where d denotes the length of a longest path in the LP solution. Using a result by Kleinberg and Rubinfeld [60], showing that $d = O(\Delta^2\alpha^{-2}\log^3 n)$ for uniform capacity graphs with some expansion parameter α and maximum degree Δ, one can achieve a better bound. Kolman and Scheideler [67] use a new graph parameter, the "flow number" F, which provides information on the level of interconnectedness of a network, and improve the ratio further to $O(F)$ for undirected graphs; they show that $F = O(\Delta\alpha^{-1}\log n)$. Kolman and Scheideler also show that there is no better approximation in general,

[2] $DTIME(f(s))$ is the class of decision problems solvable by a deterministic Turing machine in time $O(f(s))$ where s is the input size of the respective problem.

[3] $ZPTIME(f(s))$ is the class of problems solvable by randomized algorithms that always return the correct answer and whose expected running time is in $O(f(s))$ where s is the input size of the respective problem.

unless $P = NP$. Chekuri and Khanna [20] study the uniform capacity UFP for the case that the task is to satisfy as many requests as possible. Using results from [5] or [13] they find an $O(n^{2/3})$-approximation algorithm for the problem in undirected graphs, an $O(n^{4/5})$-approximation algorithm for the problem in directed graphs, and an $O(\sqrt{n}\log n)$-approximation algorithm for the problem in directed acyclic graphs.

The problem of minimizing the number of rounds has not been studied in such extend as the other two problems. Kleinberg [61] considered it in different special settings. There are more results on the similar problem of optical routings where each path is assigned a wavelength and two paths of the same wavelength may not share an arc. Minimizing the number of wavelengths results in the problem of minimizing the number of rounds in a network with unit capacities and unit demands. Optical routings were, for example, studied in [1, 4, 8, 89].

If all commodities share a common source vertex, the unsplittable flow problem gets considerably easier. The first results for the single source unsplittable flow problem come from Kleinberg [62]. He considers all three optimization problems from above and presents constant factor approximation algorithms for all of them—in both cases, the directed and the undirected one. For the directed case, Dinitz, Garg, and Goemans [26] present a 2-approximation algorithm for minimal congestion, a 4.43-approximation algorithm for the maximal routable demand, and a 5-approximation algorithm for the minimal number of rounds. These results improve the ones formerly found by Kolliopoulos and Stein [65]. Skutella [95] considers the problem with arc costs and a hard budget. He obtains a 3-approximation algorithm for minimal congestion and 8-approximation algorithms for the maximal routable demand and the minimal number of rounds.

Organization of this Chapter

The contribution of this chapter is twofold. In the first part we study a class of known algorithms for the UFP which we call *path-selecting algorithms*. These algorithms start with a standard multicommodity flow spreading flow among many paths and then choose exactly one path for each commodity. Among those algorithms is randomized rounding. It is shown by Leighton, Rao, and Srinivasan [71] that for undirected graphs the performance ratio of randomized rounding is $\Omega(\log m/\log\log m)$. This lower bound for the performance ratio can be adapted to the directed case. In Section 2.3 we give an alternative proof for the lower bound. Our proof is much simpler and even covers the case that all commodities share a common source and a

common sink. Moreover, our proof works for both cases, the directed and the undirected one, and we show that starting with an optimal solution to the classical multicommodity flow problem which uses a minimal number of paths does not help to break the $\Omega(\log m/\log\log m)$ lower bound for randomized rounding. Parts of Section 2.3 have already been published in [75, 76].

In Section 2.4 we consider only single source multicommodity flows. We establish an interesting connection between fractional and unsplittable flows. What we show is that each feasible single source multicommodity flow can be written as a convex combination of unsplittable flows of congestion at most 2. The unsplittable flows route the same commodities as the initial flow and for each commodity the demand equals the initial demand rounded (up or down) to the next $d_{max}/2^\ell$, for some $\ell \in \mathbb{N}$. Here, d_{max} is the maximum demand of the underlying problem. The proof for the existence of the convex combination is constructive. In Section 2.4.2 we present an algorithm which computes unsplittable flows with appropriate weights for the convex combination. We give a detailed analysis of the algorithm proving its correctness and the existence of the requested unsplittable flows in Section 2.4.3. We also show that the congestion of 2 is tight for the unsplittable flows, i.e., there are networks and fractional flows for which each convex combination of unsplittable flows includes at least one flow with congestion arbitrarily close to 2.

2.2 Problem Definition and Notation

An instance of the unsplittable flow problem consists of a digraph $D = (V, A)$ with arc capacities $u : A \to \mathbb{R}^+$ and a set $\mathcal{T} \subseteq V \times V$ of $K \in \mathbb{N}$ *requests* or *commodities*. Together with the ith request (denoted by (s_i, t_i)) we are given its nonnegative *demand* d_i.

An unsplittable flow satisfying the given demands sends d_i units of flow along a single s_i-t_i-path P_i, for each $i = 1, \ldots, K$. We say that this flow is *feasible*, if it obeys all arc capacities, i.e., if

$$\sum_{\substack{i\in\{1,\ldots,K\}:\\ a\in P_i}} d_i \;\leq\; u(a) \qquad \text{for all } a \in A. \tag{2.1}$$

The unsplittable flow problem was introduced by Kleinberg [61] and generalizes the arc disjoint paths problem which is well known to be NP-complete [39]. In general, it is not possible to find a feasible unsplittable flow. It is common to consider optimization problems for the UFP, such as the problem to minimize the congestion. The *congestion* of an unsplittable flow is defined in analogy with that of a standard multicommodity flow. It

is the minimum value $\alpha \geq 1$ such that by multiplying the right hand sides of the inequalities in (2.1) with α these inequalities become true for the underlying flow. The problem to minimize the congestion of an underlying UFP is that of finding an unsplittable flow of minimum congestion.

In this chapter we use the notion *unsplittable flow* twofold. On the one hand, for a set of paths as defined above, and on the other hand, for a multicommodity flow (given arcwise) which can be decomposed into flows on paths such that, for each $i = 1, \ldots, K$, the d_i units of flow departing in s_i and arriving in t_i are routed along a single path only.

In the following we use $d_{min} := \min_{i=1,\ldots,K} d_i$ and $d_{max} := \max_{i=1,\ldots,K} d_i$ to denote the minimum and the maximum demand, respectively. We say that the *balance condition* is met, if $d_{max} \leq u(a)$ for all $a \in A$. Throughout this chapter the balance condition is always assumed to be met, unless stated otherwise. To distinguish between standard multicommodity and unsplittable flows we sometimes use the notion *fractional* to underline that a flow is not necessarily unsplittable.

2.3 A Lower Bound for Path-Selecting Algorithms

In this section we give a lower bound for the performance ratio of a special class of algorithms for the minimum congestion version of the UFP. We call those algorithms *path-selecting*, because they start with a solution to the classical multicommodity flow problem, compute a path decomposition, and select one of its paths for each commodity in order to obtain an unsplittable flow. The probably most popular method among these algorithms is *randomized rounding*. Randomized rounding has proved to be a very successful tool in the design of approximation algorithms for many NP-hard discrete optimization problems during the last decade. The general idea is as follows: Formulate the problem as a 0/1-integer linear program, solve the linear programming relaxation, and randomly turn the resulting fractional solution into an integral solution by interpreting the fractional values as probabilities. This idea was originally introduced in 1987 by Raghavan and Thompson [88] for the edge-disjoint paths problem which is a special case of the unsplittable flow problem. It is shown by Leighton, Rao, and Srinivasan [71] that for undirected graphs the performance ratio of randomized rounding is $\Omega(\log m / \log \log m)$ and for directed graphs there is an integrality gap of the same size. Since the integrality gap only specifies the ratio of the objective values of an optimal fractional and an optimal integral solution, it does not say anything about the gap between an optimal solution for an instance of the UFP and a solution obtained by randomized rounding.

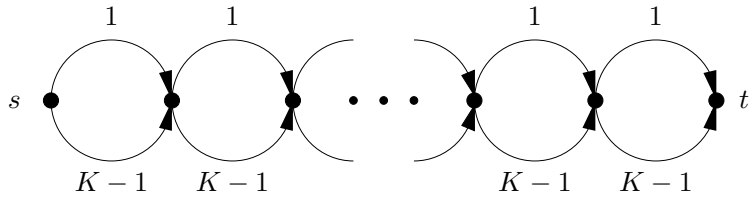

Figure 2.1: An instance of the unsplittable flow problem consisting of K commodities with unit demands, sharing a common source s and a common sink t. There are K^K consecutive pairs of parallel arcs. The numbers at the arcs indicate their capacities.

We give an alternative proof for the result in [71]. Our proof works for both cases, the directed and the undirected one, and more generally for all path-selecting algorithms. (Nevertheless, we apply it only to randomized rounding for clarity of presentation.) Moreover, we show that starting with an optimal solution to the classical multicommodity flow problem which uses a minimal number of paths does not help to break the $\Omega(\log m/\log\log m)$ lower bound for randomized rounding.

Theorem 2.1. The performance ratio of randomized rounding for the minimum congestion version of the unsplittable flow problem is $\Omega(\log m/\log\log m)$.

We present the proof for this lower bound only for the directed case. For the undirected case it works analogously.

Proof. Consider an instance of the UFP consisting of K^K consecutive pairs of parallel arcs and K commodities with unit demands, sharing a common source s (the "first" node of the consecution of arcs) and a common sink t (the "last" node of the consecution of arcs). Each "upper" arc in a pair has a capacity of 1 and the "lower" arcs have capacities $K-1$. This instance of the UFP is depicted in Figure 2.1. Obviously, there exists an unsplittable flow with congestion 1 where the flow value on each arc is equal to its capacity. Of course, the latter property must hold for any optimal fractional solution. Consider the following fractional path decomposition: Each commodity $i = 1, \ldots, K$ is distributed among K paths $P_1^i, \ldots, P_K^i$ with flow values equal to $1/K$. The K^K upper arcs of the graph are labeled by K-tuples $(j_1, \ldots, j_K) \in \{1, \ldots, K\}^K$ such that every arc gets a different label. For $i, j = 1, \ldots, K$, path P_j^i uses an upper arc labeled $(j_1, \ldots, j_K)$ if and only if $j_i = j$. The underlying intuition of this construction is that, for each subset of paths containing exactly one path of each commodity, there is an upper arc that is exactly used by the paths in the considered subset.

We argue that, for the described path decomposition, randomized rounding always yields an unsplittable flow of congestion K: By construction of

the fractional path decomposition, for every possible random choice, there exists an upper arc of unit capacity which is used by all K commodities. In particular, the flow on this arc exceeds its capacity by a factor K. Since $m = 2K^K$, we get $K = \Omega(\log m / \log\log m)$. □

The performance of randomized rounding depends on the particular choice of an optimal fractional solution, i.e., on the paths along which flow is sent. At first sight, the fractional solution (path decomposition) used in the proof above seems rather artificial for the considered network. We show in the following that this solution is indeed an extreme point (i.e., a basic solution) for the path-based LP-formulation of the fractional flow problem.

Let $\mathcal{P}$ be the set of all s-t-paths in the network under consideration. We formulate the problem of finding an optimal fractional flow as a linear program with variables $f_{P,i} \geq 0$ denoting the amount of flow of commodity $i \in \{1, \ldots, K\}$ sent along path $P \in \mathcal{P}$. The constraints of the linear program are as follows:

$$\sum_{i=1}^{K} \sum_{\substack{P \in \mathcal{P}: \\ a \in P}} f_{P,i} \;=\; 1 \qquad \text{for all upper arcs } a, \tag{2.2}$$

$$\sum_{P \in \mathcal{P}} f_{P,i} \;=\; 1 \qquad \text{for all } i \in \{1, \ldots, K\}. \tag{2.3}$$

Since the sum of capacities of every pair of parallel arcs equals the total flow value K, constraints (2.2) suffice to enforce all capacity constraints. Let $\mathcal{P}_i = \{P_1^i, \ldots, P_K^i\}$ be the subset of paths used by commodity i in the path decomposition described in the proof of Theorem 2.1.

Lemma 2.2. The columns of the matrix given by the left hand sides of (2.2) and (2.3) corresponding to the variables $f_{P,i}$, with $i = 1, \ldots, K$ and $P \in \mathcal{P}_i$, are linearly independent.

Proof. We assume by contradiction that there exists a nontrivial linear combination of these columns which yields the zero vector. For $i, j = 1, \ldots, K$, the coefficient of the column corresponding to variable $f_{P,i}$ with $P = P_j^i$ in this linear combination is denoted by a_{ij}. By symmetry, we assume without loss of generality that $a_{11} > 0$. Considering the row corresponding to the ith constraint of (2.3), we observe that $\sum_{j=1}^{K} a_{ij} = 0$, for all $i = 1, \ldots, K$. Thus, for all $i = 2, \ldots, K$, there exists a nonnegative coefficient $a_{ij_i} \geq 0$. However, considering the constraint in (2.2) corresponding to the upper arc labeled $(1, j_2, \ldots, j_K)$, we get $a_{11} + \sum_{i=2}^{K} a_{ij_i} = 0$ which is a contradiction and therefore concludes the proof. □

Lemma 2.2 implies that the path decomposition from above yields a basic feasible solution to the linear program. Nevertheless, the chosen path decomposition seems odd since it uses K^2 paths while there is an obvious path decomposition using only K paths (i.e., the optimal unsplittable flow solution). It is an interesting question whether the lower bound $\Omega(\log m/\log\log m)$ still holds if the randomized rounding algorithm starts with a path decomposition of an optimal fractional solution using a minimal number of paths. This is obviously not true for the instance discussed above.

In the following, we modify the instance such that an optimal fractional solution using a minimal number of paths still yields the lower bound $\Omega(\log m/\log\log m)$ for randomized rounding. We slightly perturb the right hand sides of constraints (2.2) and (2.3). To enforce the balance condition, we perturb the upper arc capacities to values slightly above 1 and the demands to values slightly below 1. Furthermore, we also modify the capacities of the lower arcs such that again the capacities of all pairs of arcs are equal to the total demand.

By choosing a random perturbation, with probability 1 there are no degenerate basic solutions anymore, i.e., all basic variables (paths) have a positive value (for more details on perturbations of linear programs we refer to the standard literature such as, e.g., [15]). This means that all basic solutions use a minimal number of paths. In particular, by Lemma 2.2, there is a perturbed version of our fractional solution considered above which is a basic solution to the modified instance and thus uses a minimal number of paths. Since the perturbation can be arbitrarily small, this basic solution to the perturbed instance is arbitrarily close to the fractional solution considered above such that the expected congestion of an unsplittable flow obtained by randomized rounding is in $\Omega(\log m/\log\log m)$. Moreover, the congestion of an optimal unsplittable flow is still arbitrarily close to 1. This yields the following corollary.

Corollary 2.3. The performance ratio of randomized rounding for the minimum congestion version of the unsplittable flow problem is $\Omega(\log m/\log\log m)$, even if the algorithm starts with an optimal fractional solution using a minimal number of paths.

2.4 Convex Combinations for Multicommodity Flows

In this section we only consider the case that all requests share the same source. In this situation, it is common to speak of the *single source* multicommodity (or unsplittable) flow problem. We describe an algorithm which is given an arbitrary fractional single source multicommodity flow and com-

putes unsplittable flows with congestion at most 2 as well as coefficients for a convex combination of these that results in the input flow. The demands of the unsplittable flows slightly differ from the ones in the original flow: They respectively equal the next (upper or lower) fraction $d_{max}/2^{\ell}$, for some $\ell \in \mathbb{N}$.

The result that each single source multicommodity flow can be presented as a convex combination of unsplittable flows with congestion at most 2 is interesting from the following point of view: Dinitz, Garg, and Goemans [26] have already proven that an arbitrary single source multicommodity flow can be made unsplittable by losing only a factor 2 with respect to congestion (or even by sending only at most additional d_{max} units of flow along each arc). Concerning this result the following question arose: Given an arbitrary cost function on the arcs, does there also exist a suitable unsplittable flow whose costs do not exceed the costs of the original (fractional) flow? This question is answered by the algorithm that is developed in this section. The requested unsplittable flow exists if and only if the original (fractional) flow can be presented as a convex combination of unsplittable flows with congestion at most 2 (or of those with at most additional d_{max} units of flow along each arc).

In Section 2.4.2 we present an algorithm that computes suitable unsplittable flows for such a convex combination. A detailed analysis of the algorithm follows in Section 2.4.3. Beforehand, we introduce some definitions in Section 2.4.1 that simplify the following presentations. Further, we give detailed definitions for single source multicommodity flows in order to clarify our notation.

2.4.1 Definitions

An instance of the single source multicommodity flow problem is defined as follows. We are given a digraph $D = (V, A)$ with arc capacities $u : A \to \mathbb{R}^+$, a source $s \in V$ and $K \in \mathbb{N}$ sinks $t_1, \ldots, t_K \in V$. Sources and sinks are also called *terminals*. We assume without loss of generality that the terminals are pairwise distinct.

A single source multicommodity flow in this network is a function $f : A \to \mathbb{R}_0^+$ that satisfies flow conservation, i.e.,

$$\sum_{a \in \delta^-(v)} f(a) = \sum_{a \in \delta^+(v)} f(a) \qquad \text{for all } v \in V \setminus \{s, t_1, \ldots, t_K\}.$$

To ensure that s is the unique source of f, we must require in addition that

$$\sum_{a \in \delta^-(t_i)} f(a) \geq \sum_{a \in \delta^+(t_i)} f(a) \qquad \text{for all } i = 1, \ldots, K.$$

We say that a flow is *feasible*, if it obeys the arc capacities, i.e., $f(a) \leq u(a)$ for all $a \in A$. A single source multicommodity flow satisfies *demands* $d_1, \ldots, d_K \in \mathbb{R}_0^+$, if

$$\sum_{a \in \delta^-(t_i)} f(a) = \sum_{a \in \delta^+(t_i)} f(a) + d_i \qquad \text{for all } i \in \{1, \ldots, K\}.$$

In this classical single source multicommodity flow problem it is allowed to split flow from the source to a sink and let it travel on different paths through the network. In contrast, the single source unsplittable flow problem forbids to use more than a single path in order to send flow from the source s to a sink t_i. Thus, the problem is not to find a flow function f as defined above, but rather to find a single s-t_i path P_i with a nonnegative flow value f_i, for each $i \in \{1, \ldots, K\}$. The resulting flow is feasible (obeys the arc capacities), if

$$\sum_{\substack{i \in \{1,\ldots,K\}: \\ a \in P_i}} f_i \leq u(a) \qquad \text{for all } a \in A.$$

Flow conservation is satisfied trivially. An unsplittable single source multicommodity flow satisfies demands $d_1, \ldots, d_K$, if $f_i = d_i$ for all $i \in \{1, \ldots, K\}$.

Sometimes the arc flow of an unsplittable flow is used. The flow on arc $a \in A$ of the unsplittable flow given above is defined as $f(a) := \sum_{i:a \in P_i} f_i$. Thus, we can build the sum $f + g$ of two flows f and g arcwise, even if one of the flows is given pathwise.

We say that a *terminal path* is a maximal (not necessarily directed) simple path in D with endpoints in $\{s, t_1, \ldots, t_K\}$, where *maximal* means that it is not extendable at either of its endpoints. A (not necessarily directed) cycle or a terminal path is called an *augmenting structure*.

For a natural number ℓ, we define $r_f^\ell(a) := f(a) \mod \frac{d_{max}}{2^\ell}$, for $a \in A$. A flow is called *q-integral*, for some $q \in \mathbb{R}^+$, if the flow value on each arc is an integral multiple of q.

Note that any single source multicommodity flow that is q-integral can be decomposed into flows on paths with flow value q.

2.4.2 Constructing the Convex Combination

We now present an algorithm that computes—given a fractional single source multicommodity flow f^{init}—unsplittable flows with weights such that the weighted sum of the flows equals f^{init}. The congestion of all these unsplittable flows is at most 2, their demands are found by rounding the ones in f^{init}. In more detail, each flow exceeds all arc capacities only by at most an additive d_{max}. If nothing else is declared, d_{max} and d_{min} refer to the initial flow f^{init}.

In the following we shortly sketch the idea of the algorithm, before we give a more detailed characterization of it later on: We want to round each demand to the two nearest (upper and lower) values of the form $d_{max}/2^\ell$, for some $\ell \in \mathbb{N}$, and then send these rounded demands unsplittably.[4] The largest value we need to consider for ℓ is

$$L := \left\lceil \log \frac{d_{max}}{d_{min}} \right\rceil \quad ,$$

since $d_{max}/2^L$ is the largest fraction $d_{max}/2^\ell$ ($\ell \in \mathbb{N}$) that is at most d_{min}. (Thus, $d_{max}/2^L$ is the smallest demand that we obtain from rounding all demands to $(d_{max}/2^\ell)$-integrality, for some $\ell \in \mathbb{N}$.)

We start by making the input flow $(d_{max}/2^L)$-integral. This is done by augmenting flow on cycles and terminal paths. The particular augmenting structure and the increment of flow are chosen such that after augmentation the flow on at least one additional arc is $(d_{max}/2^L)$-integral. Augmentation on a cycle does not change the demands, whereas augmentations on terminal paths yield changes of demands to $(d_{max}/2^L)$-integrality. We always augment in both directions of an augmenting structure and thus obtain two new flows in each step that will form a convex combination of the "parent" flow if suitable weights are chosen. Depending on the direction into which flow is augmented on a path, demands are rounded up or down. (See Figure 2.2 for an illustration of the algorithm.) When a flow is $(d_{max}/2^L)$-integral, demands of value $d_{max}/2^L$ can be satisfied unsplittably. For this reason, we want to prevent such demands and corresponding flow carrying paths from further changes. Thus, for each sink t with demand $d_{max}/2^L$, we decrease the current flow along a corresponding flow carrying s-t-path by $d_{max}/2^L$. These paths with the flow values are stored for all subsequent flows. After the decrement, all demands are at least $d_{max}/2^{L-1}$ and we turn to make the (remaining) flow $(d_{max}/2^{L-1})$-integral. We proceed in this manner until the (remaining) flow is d_{max}-integral. Then all (remaining) demands equal d_{max} and are served unsplittably.

For the sake of simple presentation and analysis of the algorithm, we give a recursive description of it in Algorithm 1. The initial call to start the recursive algorithm is given by DECOMP(D,f^{init},$\emptyset$,1) where f^{init} is the single source multicommodity flow in the digraph D that is to be decomposed into unsplittable flows. The third parameter is a set of paths with corresponding flow values. If (P, ϕ) is in this set, it means that all subsequent flows route

[4] We could also do without rounding demands that are initially satisfied unsplittably. These demands and their corresponding paths could simply be excluded from further consideration. Nevertheless, we do not set that rounding aside for simplicity of notation.

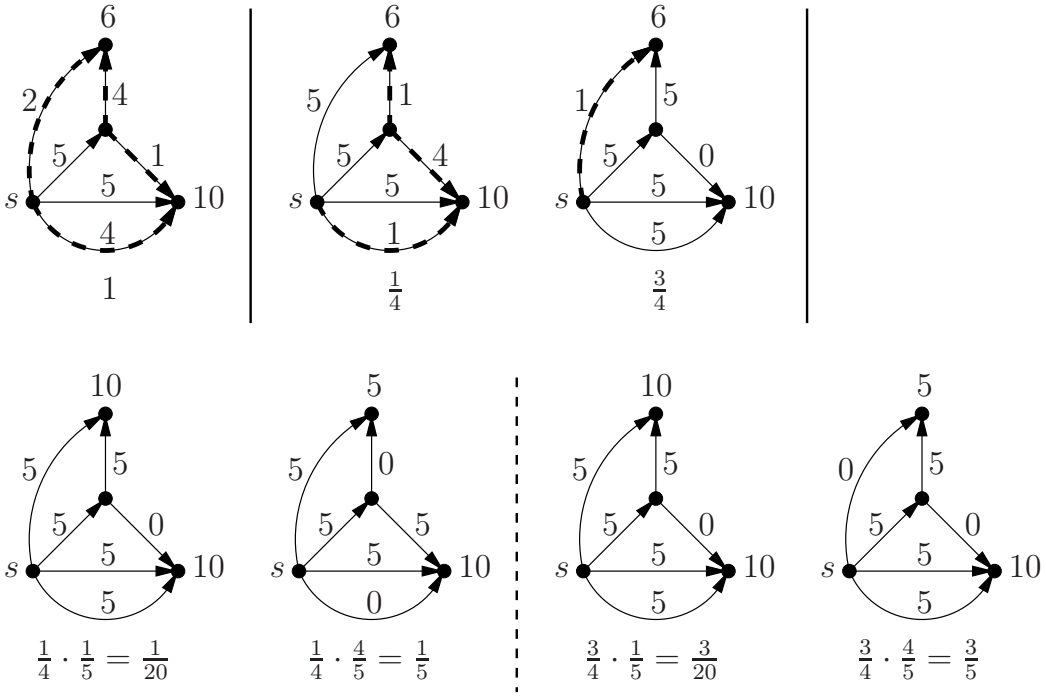

Figure 2.2: Steps of the algorithm to 5-integrality. We start with the flow in the upper left corner and read from left to right first. Since $d_{max}/2^L$ equals 5, we need to make the flow 5-integral. The dashed arcs indicate the augmenting structure that is used. Since we augment flow in either direction of an augmenting structure, each non-5-integral flow produces two new flows that are separated from their "parents" by a vertical line. We consider the first flow. If we use the indicated augmenting structure clockwise, we may augment by 3. Then the arc with flow value 2 becomes 5-integral. Using the augmenting structure counterclockwise, we may augment by 1. Then the unit of flow on the arc with flow value 1 recedes. The number below each flow indicates its weight in the convex combination.

ϕ units of flow along P. The last parameter indicates the weight of the flow that is to be decomposed. For the initial flow this weight equals 1.

A call of DECOMP(D,f^{init},$\emptyset$,1) effects the following: In each step of the recursion, it first updates the input flow f by iteratively deleting demands $d_{max}/2^\ell$, for $\ell \in \mathbb{N}$, if f is $(d_{max}/2^\ell)$-integral. The related flow carrying paths are added to the current unsplittable flow given by $\mathcal{P}$. Afterwards it decomposes the (remaining) flow f into two new flows f_1 and f_2 that result from a single augmentation in both directions of a suitable augmenting structure. Further steps of the recursion decompose f_1 and f_2 into unsplittable flows.

The augmentation itself works as follows. Let us assume that the flow f that remains after the update is $(d_{max}/2^\ell)$- but not $(d_{max}/2^{\ell-1})$-integral, for some $\ell \in \{1, \ldots, L\}$. Then we consider the subgraph $\tilde{D}$ of D that consists of all arcs $a \in A$ whose flow values $f(a)$ are not $(d_{max}/2^{\ell-1})$-integral. Starting

Algorithm 1: DECOMP(D,f,$\mathcal{P}$,w)

Input: A digraph $D = (V, A)$, a single source multicommodity flow f in D with source s, a set $\mathcal{P}$ of paths from s to pairwise distinct nodes t in D with corresponding flow values, and a positive weight $w \in (0, 1]$. The maximum (minimum) demand, that f satisfies, is d_{max} (d_{min}).

Output: A set of unsplittable flows with weights that sum up to w yielding a conic combination of $w(f + f_{\mathcal{P}})$, where $f_{\mathcal{P}}$ denotes the flow that is given by $\mathcal{P}$.

for $i = \left\lceil \log \frac{d_{max}}{d_{min}} \right\rceil$ **downto** 0 **do**
  if f *is* $(d_{max}/2^i)$*-integral* **then**
    for *each sink* t *having demand* $d = d_{max}/2^i$ *in* f **do**
      Determine an arbitray flow carrying s-t-path P in D.
      Set $f(a) := f(a) - d$ for all $a \in P$.
      Set $\mathcal{P}' := \mathcal{P} \cup \{(P, d)\}$.
    end
  end
end
if $f \equiv 0$ **then**
  return $(\mathcal{P}', w)$.
end
Set $\ell := \min\{\min\{j \in \mathbb{N} | f \text{ is } (d_{max}/2^j)\text{-integral}\}, \left\lceil \log \frac{d_{max}}{d_{min}} \right\rceil + 1\}$.
Let $C \subseteq A$ be an augmenting structure with $r_f^{\ell-1}(a) \neq 0 \; \forall \; a \in C$.
Set $C^+ := \{a \in C \mid C \text{ traverses } a \text{ in forward direction}\}$,
  $C^- := \{a \in C \mid C \text{ traverses } a \text{ in backward direction}\}$.
Set $\delta_1 := \min\{\min_{a \in C^-} r_f^{\ell-1}(a), \min_{a \in C^+} \frac{d_{max}}{2^{\ell-1}} - r_f^{\ell-1}(a)\}$,
  $\delta_2 := \min\{\min_{a \in C^+} r_f^{\ell-1}(a), \min_{a \in C^-} \frac{d_{max}}{2^{\ell-1}} - r_f^{\ell-1}(a)\}$.
For $a \in A \setminus C$ set $f_1(a) := f(a)$ and $f_2(a) := f(a)$.
For $a \in C^+$ set $f_1(a) := f(a) + \delta_1$ and $f_2(a) := f(a) - \delta_2$.
For $a \in C^-$ set $f_1(a) := f(a) - \delta_1$ and $f_2(a) := f(a) + \delta_2$.
Set $w_1 := \frac{\delta_2}{\delta_1+\delta_2} w$ and $w_2 := \frac{\delta_1}{\delta_1+\delta_2} w$.
return DECOMP(D,f_1,$\mathcal{P}'$,w_1) $\cup$ DECOMP(D,f_2,$\mathcal{P}'$,w_2).

from an arbitrary arc we follow an undirected path (in either direction) until there is no more incident arc or we get to a node which has already been visited. The first criterion results in a terminal path, the second one in a

cycle. (We prove this later on in Lemma 2.5.) For the resulting augmenting structure C we augment by the minimum δ of gaps between flow values and the next lower multiples of $d_{max}/2^{\ell-1}$ for arcs that are used by C in backward direction and of gaps between flow values and the next upper multiples of $d_{max}/2^{\ell-1}$ for arcs that are used by C in forward direction. Defining C^- as the backward arcs in C and C^+ as the forward arcs in C we can write δ as follows.

$$\delta = \min\{\min_{a \in C^-} r_f^{\ell-1}(a), \min_{a \in C^+} \frac{d_{max}}{2^{\ell-1}} - r_f^{\ell-1}(a)\}$$

Now let f_1 and f_2 be the flows resulting from augmenting along C and its "counterpart", i.e., C in the opposite direction. Let δ_1 and δ_2 be the corresponding augmentation values. Then the weight of f_i (for $i = 1, 2$) is given by the weight of f multiplied with $\delta_{3-i}/(\delta_1 + \delta_2)$.

2.4.3 Analysis of the Construction

In this section we prove that the flows and weights that are returned by DECOMP(D,f^{init},$\emptyset$,1) yield a convex combination for f^{init}. Further, the produced flows are unsplittable and all its demands are of the form $d_{max}/2^{\ell}$, for some $\ell \in \{0, \ldots, \lceil \log(d_{max}/d_{min}) \rceil\}$.

First let us prove that the algorithm is well defined in the sense that the input specification is always fulfilled.

Lemma 2.4. In each call DECOMP(D,f,$\mathcal{P}$,w) that results from a call DECOMP(D,f^{init},$\emptyset$,1) it holds that

1. f is a single source multicommodity flow in D,
2. $\mathcal{P}$ is a set of paths from the source of f to pairwise distinct nodes t in D with corresponding flow values, and
3. $w \in (0, 1]$.

Proof. Note that the parameters of the initial call DECOMP(D,f^{init},$\emptyset$,1) fulfill the input specification.

Consider any call of DECOMP(D,f,$\mathcal{P}$,w) such that the parameters f, $\mathcal{P}$, and w satisfy the input specification. We show that f_1, $\mathcal{P}$', and w_1 as well as f_2, $\mathcal{P}$', and w_2 as defined in Algorithm 1 fulfill the input specification, too. This is done by proving that they obey 1 to 3. Then the lemma is proven by induction.

We prove that f_1, $\mathcal{P}$', and w_1 obey 1 to 3. The same can be proven analogously for f_2, $\mathcal{P}$', and w_2. In the following we use s to denote the source in f.

1. We show that no additional source emerges from the decrement of flow in the for-loop or the augmentation of flow. The decrement of flow only deletes existent flow along paths from s to some sinks and does not create a new source. Augmentation along a cycle does not change the amount of flow that appears or vanishs in any node.

   We must show that augmentation along a terminal path P does not cause the creation of a new source. Consider a sink t at the end of P. Then there is only one arc $a \in A$ incident to t whose flow value is not $(d_{max}/2^{\ell-1})$-integral, for ℓ as used in Algorithm 1. First assume that $a \in \delta^+(t)$. Since t is a sink, the inflow in t is at least by a value $x := d_{max}/2^{\ell-1} - r_f^{\ell-1}(a)$ larger than the outflow from t. By the choice of δ_1 and δ_2, the flow along a is increased by at most x. Therefore, t cannot become a source. If $a \in \delta^-(t)$, the inflow in t is at least by a value $r_f^{\ell-1}(a)$ larger than the outflow from t. Since the flow along a is decreased by at most $r_f^{\ell-1}(a)$, it follows that t cannot become a source in this case either.

   Therefore, the single source in f_1 is given by s.

2. To obtain $\mathcal{P}$', we merely add s-t-paths with corresponding weights to $\mathcal{P}$ for some sinks t with positive demands. The amount of flow arriving in t in f is 0 afterwards. (Remember that we assumed the sinks to be pairwise distinct.) The proof of Lemma 2.5 shows that any augmenting structure cannot end in a sink with demand 0. Thus, the demand of any sink for which a path is in $\mathcal{P}'$ cannot become positive again. It follows that all sinks for which corresponding flow paths are in $\mathcal{P}'$ are pairwise distinct.

3. Note that after the update in the for-loop, f is either the zero flow or not yet d_{max}-integral. (For a flow that is d_{max}-integral all demands would have been deleted.) If $f \equiv 0$, we are done. In Lemma 2.5 we prove that otherwise there exists a suitable augmenting structure in f. Thus, $\delta_1, \delta_2 > 0$. It follows with $w_1 \leq w$ that $w_1 \in (0, 1]$.

This completes the proof. □

Lemma 2.5, which has already been mentioned, is also helpful to prove that our algorithm terminates.

Lemma 2.5. If a flow f in D is not $(d_{max}/2^{\ell})$-integral, for some $\ell \in \mathbb{N}$, then there exists an augmenting structure $C \subseteq A$ with $r_f^{\ell}(a) \neq 0$, for all $a \in C$.

Proof. Consider the subgraph $\tilde{D}$ of D that consists of all arcs $a \in A$ whose flow values $f(a)$ are not $(d_{max}/2^\ell)$-integral, for some $\ell \in \mathbb{N}$. Assume that we start from an arbitrary arc in $\tilde{D}$, follow an undirected path in $\tilde{D}$ in both directions as long as possible and do not find a cycle. We are done if we prove that both ends of the determined path are terminals.

Let $v \in V$ be an end node of the path. We show that v must be the source or a sink with positive demand. Since v is an end node of the path and by definition of $\tilde{D}$, there is exactly one arc $a \in A$ incident to v whose flow value $f(a)$ is not $(d_{max}/2^\ell)$-integral. Suppose that $a \in \delta^+(v)$. Then there exist $p,\ q \in \mathbb{N}$ with

$$\sum_{a' \in \delta^+(v)} f(a') = p \cdot \frac{d_{max}}{2^\ell} + r_f^\ell(a) \qquad \text{and}$$

$$\sum_{a' \in \delta^-(v)} f(a') = q \cdot \frac{d_{max}}{2^\ell} \qquad .$$

Since $r_f^\ell(a) \in (0, d_{max}/2^\ell)$, it follows that $\sum_{a' \in \delta^+(v)} f(a') \neq \sum_{a' \in \delta^-(v)} f(a')$. Since flow conservation is obeyed in all nodes that are no terminals (or a sink with demand equal to 0), it follows that v must be the source or a sink with positive demand. An analogous argument holds if $a \in \delta^-(v)$. □

The following lemma proves that if f is not decreased in the for-loop of DECOMP(D,f,$\mathcal{P}$,w), the flows f_1 and f_2 are "more integral" than f.

Lemma 2.6. For any flow f that is augmented in Algorithm 1 and its corresponding value ℓ as defined in the algorithm, it holds that f_1 and f_2 each have at least one more arc than f whose flow value is $(d_{max}/2^{\ell-1})$-integral.

Proof. First note that the flow f is not $(d_{max}/2^{\ell-1})$-integral for ℓ as defined in Algorithm 1. Further $\ell \geq 1$, because otherwise f would be d_{max}-integral which is a contradiction to the specification of the for-loop. Then the claim follows from the definition of δ_1 and δ_2 and the augmentation rule. Let C be the augmenting structure that leads from f to f_1 and f_2 and let $a \in C$ be the arc for which the minimum defining δ_1 is attained. If $a \in C^-$, it follows that $f_1(a) = f(a) - r_f^{\ell-1}(a)$. Since it holds by definition that $r_f^{\ell-1}(a) = f(a) \mod \frac{d_{max}}{2^{\ell-1}}$, we have $f_1(a) \mod \frac{d_{max}}{2^{\ell-1}} = 0$.

Analogous equations hold for $a \in C^+$ and for f_2. Since we only change flow on arcs in C and the flow values on all arcs in C are not $(d_{max}/2^{\ell-1})$-integral, flow values on $(d_{max}/2^{\ell-1})$-integral arcs remain unchanged. □

Before we turn to proving that the flows and weights that are returned by DECOMP(D,f,$\mathcal{P}$,w) yield a conic combination of wf, we show that the

procedure indeed terminates and outputs unsplittable flows whose demands are of the form $d_{max}/2^\ell$, for some $\ell \in \{0, \dots, \lceil \log(d_{max}/d_{min}) \rceil\}$.

Corollary 2.7. DECOMP(D,f,$\mathcal{P}$,w) terminates. The output is a set of unsplittable flows whose demands are of the form $d_{max}/2^\ell$, for some $\ell \in \{0, \dots, \lceil \log(d_{max}/d_{min}) \rceil\}$.

Proof. We have already proven that the flows f_1 and f_2 that result from DECOMP(D,f,$\mathcal{P}$,w) have fewer positive demands than f or are "more integral". The first property eventually results in a decrement of the input flow to the zero flow. In every recursive call of Algorithm 1 in which the number of positive demands is not decreased for the respective input, the flow value on at least one of its arcs changes to a "higher" integrality. After at most m steps we therefore change from $(d_{max}/2^\ell)$-integrality to $(d_{max}/2^{\ell-1})$-integrality for some $\ell \in \{1, \dots, L\}$ (or respectively from the initial state to $(d_{max}/2^L)$-integrality). At this point demands of value $(d_{max}/2^{\ell-1})$ and corresponding flow are deleted in the for-loop. If no such demands exist, we go to "higher" integralities and delete demands at the latest when the flow is d_{max}-integral. Therefore, at some point all demands are deleted and the algorithm terminates.

It follows from the preceding proofs that all paths in $\mathcal{P}'$ connect the source in f with pairwise distinct sinks. Thus, $\mathcal{P}'$ yields an unsplittable flow. It follows directly from the specification of the algorithm that all demands served by $\mathcal{P}'$ are of the form $d_{max}/2^\ell$, for some $\ell \in \{0, \dots, \lceil \log(d_{max}/d_{min}) \rceil\}$. □

In the following we use $f_\mathcal{P}$ to denote the flow that is given by some set $\mathcal{P}$ of paths with corresponding flow values. We prove the following helpful lemma in order to show that DECOMP(D,f,$\mathcal{P}$,w) returns the specified output.

Lemma 2.8. Consider DECOMP(D,f,$\mathcal{P}$,w). It holds that

$$w(f + f_\mathcal{P}) = w_1(f_1 + f_{\mathcal{P}'}) + w_2(f_2 + f_{\mathcal{P}'}) \tag{2.4}$$

and $w_1 + w_2 = w$.

Proof. The second part of the lemma follows immediately from the definition of w_1 and w_2. Equation (2.4) can be proven as follows.

For all arcs $a \in A$ that are not in the augmenting structure C that leads from f to f_1 and f_2, it holds that $f_1(a) = f_2(a) = f(a) - (f_{\mathcal{P}'}(a) - f_\mathcal{P}(a))$. Since w_1 and w_2 sum up to w, equation (2.4) follows immediately for such arcs.

Now consider an arc $a \in C^+$. It holds that

$$\begin{aligned} f_1(a) &= f(a) - (f_{\mathcal{P}'}(a) - f_{\mathcal{P}}(a)) + \delta_1 \qquad \text{and} \\ f_2(a) &= f(a) - (f_{\mathcal{P}'}(a) - f_{\mathcal{P}}(a)) - \delta_2 \qquad . \end{aligned}$$

Hence, it follows with $w_1 + w_2 = w$ that

$$\begin{aligned} w_1 f_1(a) + w_2 f_2(a) &= wf(a) - w(f_{\mathcal{P}'}(a) - f_{\mathcal{P}}(a)) + w_1\delta_1 - w_2\delta_2 \\ &= wf(a) - w(f_{\mathcal{P}'}(a) - f_{\mathcal{P}}(a)) + \frac{\delta_2\delta_1}{\delta_1+\delta_2}w - \frac{\delta_1\delta_2}{\delta_1+\delta_2}w \\ &= wf(a) + wf_{\mathcal{P}}(a) - w_1 f_{\mathcal{P}'}(a) - w_2 f_{\mathcal{P}'}(a) \qquad . \end{aligned}$$

The proof is analogous for $a \in C^-$. □

The following corollary demonstrates the correctness of the output of DECOMP(D,f,$\mathcal{P}$,w). With this result we are finished proving the correctness of the algorithm as described in Section 2.4.2. To prove our main result we still need to show that all unsplittable flows that are returned by DECOMP(D,f^{init},$\emptyset$,1) have congestion at most 2. This is done in Theorem 2.11 in Section 2.4.4.

Corollary 2.9. The flows and weights returned by DECOMP(D,f,$\mathcal{P}$,w) yield a conic combination of $w(f + f_{\mathcal{P}})$ whose weights sum up to w.

Proof. By induction on the depth of recursion we prove the claim using Lemma 2.8.

From a call DECOMP(D,f,$\mathcal{P}$,w) that causes no further calls we receive $(\mathcal{P}', w)$. It holds that $f - (f_{\mathcal{P}'} - f_{\mathcal{P}}) \equiv 0$. Thus, the output is correct.

Consider a call DECOMP(D,f,$\mathcal{P}$,w) that calls DECOMP(D,f_1,$\mathcal{P}'$,w_1) and DECOMP(D,f_2,$\mathcal{P}'$,w_2). It follows from the induction hypothesis that DECOMP(D,f_1,$\mathcal{P}'$,w_1) and DECOMP(D,f_2,$\mathcal{P}'$,w_2) return sets $\{(\mathcal{P}_1^i, w_1^i) \mid i \in I\}$ and $\{(\mathcal{P}_2^j, w_2^j) \mid j \in J\}$ with some index sets $I,\ J$. These fulfill

$$w_1(f_1 + f_{\mathcal{P}'}) = \sum_{i\in I} w_1^i f_{\mathcal{P}_1^i} \quad \text{and} \quad w_2(f_2 + f_{\mathcal{P}'}) = \sum_{j\in J} w_2^j f_{\mathcal{P}_2^j}$$

as well as $\sum_{i\in I} w_1^i = w_1$ and $\sum_{j\in J} w_2^j = w_2$. Lemma 2.8 says that

$$w(f + f_{\mathcal{P}}) = w_1(f_1 + f_{\mathcal{P}'}) + w_2(f_2 + f_{\mathcal{P}'}) = \sum_{i\in I} w_1^i f_{\mathcal{P}_1^i} + \sum_{j\in J} w_2^j f_{\mathcal{P}_2^j}$$

and $\sum_{i\in I} w_1^i + \sum_{j\in J} w_2^j = w_1 + w_2 = w$. □

Corollary 2.10. The flows and weights returned by DECOMP(D,f^{init},$\emptyset$,1) yield a convex combination of f.

Proof. By Corollary 2.9, DECOMP(D,f^{init},$\emptyset$,1) returns flows and weights that yield a conic combination of f^{init} whose weights sum up to 1. Thus, we have a convex combination. □

2.4.4 Upper Bound for the Congestion

Together with the algorithm from Section 2.4.2 and its analysis in Section 2.4.3 the following theorem is the last component for the proof of our main result. We prove that each feasible single source multicommodity flow can be written as a convex combination of unsplittable flows of congestion at most 2. This holds only for the case that the balance condition is met. A more general and even stronger result is the following.

Theorem 2.11. For a single source multicommodity flow f^{init} in $D = (V, A)$ and any arc $a \in A$, it holds that the flow along a in any flow produced by DECOMP(D,f^{init},$\emptyset$,1) exceeds $f^{\text{init}}(a)$ by at most an additive d_{max}.

Proof. Consider the progression of the input flow while DECOMP(D,f^{init},$\emptyset$,1) is running. Let f^0 be a flow that occurs on the way to $(d_{max}/2^L)$-integrality of the input flow. Further, let $\mathcal{P}^0$ be the current unsplittable flow while f^0 is considered in the algorithm.

By the choice of δ_1 and δ_2, it holds for all $a \in A$ that

$$f^0(a) + f_{\mathcal{P}^0}(a) \leq f^{\text{init}}(a) + \frac{d_{max}}{2^L} - \left(f^{\text{init}}(a) \mod \frac{d_{max}}{2^L} \right) \quad , \qquad (2.5)$$

because once the flow on a is $(d_{max}/2^L)$-integral, i.e., rounded to at most the next multiple of $d_{max}/2^L$, it is not changed again on the way to $(d_{max}/2^L)$-integrality.

After $(d_{max}/2^L)$-integrality was reached, the input flow is iteratively augmented to $(d_{max}/2^{L-\ell})$-integrality for gradually increasing $\ell \in \{1, \ldots, L\}$. Let f^ℓ be a flow that occurs while DECOMP(D,f^{init},$\emptyset$,1) is running and that is $(d_{max}/2^{L-\ell})$-integral, but not $(d_{max}/2^{L-\ell-1})$-integral. Further, let $f^{\ell-1}$ be any anchestor of f^ℓ, i.e., any of the flows that (indirectly) caused the creation of f^ℓ, that is $(d_{max}/2^{L-\ell+1})$-integral. Again let $\mathcal{P}^\ell$ and $\mathcal{P}^{\ell-1}$ be the corresponding unsplittable flows.

In analogy with (2.5), it follows from the choice of δ_1 and δ_2 that for all $a \in A$

$$f^\ell(a) + f_{\mathcal{P}^\ell}(a) \leq f^{\ell-1}(a) + f_{\mathcal{P}^{\ell-1}}(a) + \frac{d_{max}}{2^{L-\ell}} - \frac{d_{max}}{2^{L-\ell+1}} \quad .$$

We can prove an analogous equation if some integrality step is omitted, i.e., if there is no anchestor of f^ℓ that is $(d_{max}/2^{L-\ell+1})$-integral. Let ℓ' be the largest integer that is smaller than ℓ and for which an anchestor of f^ℓ exists that is $(d_{max}/2^{L-\ell'})$-integral. Then it holds that

$$f^\ell(a) + f_{\mathcal{P}^\ell}(a) \leq f^{\ell'}(a) + f_{\mathcal{P}^{\ell'}}(a) + \frac{d_{max}}{2^{L-\ell}} - \frac{d_{max}}{2^{L-\ell'}} \quad .$$

We obtain iteratively, for all $\ell \in \{0, \ldots, L\}$, that

$$f^\ell(a) + f_{\mathcal{P}^\ell}(a) \leq f^{\text{init}}(a) + \frac{d_{max}}{2^{L-\ell}} - \left(f^{\text{init}}(a) \mod \frac{d_{max}}{2^L} \right) \quad . \tag{2.6}$$

Now consider the point when the flow f is changed to $f' \equiv 0$ in the for-loop. Let $\mathcal{P}$ and $\mathcal{P}'$ be the corresponding unsplittable flows. Then $\mathcal{P}'$ is one of the output flows of the algorithm. Since d_{max} is the maximum demand in f, it follows that f is d_{max}-integral. With (2.6) we have

$$\begin{aligned} f_{\mathcal{P}'}(a) &= f(a) + f_{\mathcal{P}}(a) \\ &\leq f^{\text{init}}(a) + d_{max} - (f^{\text{init}}(a) \mod \frac{d_{max}}{2^L}) \\ &\leq f^{\text{init}}(a) + d_{max} \quad . \end{aligned}$$

Note that it even holds, for all $a \in A$, that $f_{\mathcal{P}'}(a) < f^{\text{init}}(a) + d_{max}$. To obtain this result, we have to regard that $f^0(a) + f_{\mathcal{P}^0}(a) \leq f^{\text{init}}(a)$, if $f^{\text{init}}(a)$ is $(d_{max}/2^L)$-integral. □

If we assume f^{init} to be feasible, the next result follows immediately, for instances of the single source multicommodity flow problem that meet the balance condition.

Corollary 2.12. If a single source multicommodity flow f^{init} in $D = (V, A)$ obeys arc capacities $u : A \to \mathbb{R}^+$ (and the balance condition is met), then all flows produced by DECOMP(D,f^{init},$\emptyset$,1) have congestion at most 2.

We close this section by proving that our result is tight.

Lemma 2.13. There exists a network and a feasible fractional single source multicommodity flow f^{init} such that in each convex combination of unsplittable flows forming f^{init} there is at least one flow with congestion arbitrarily close to 2.

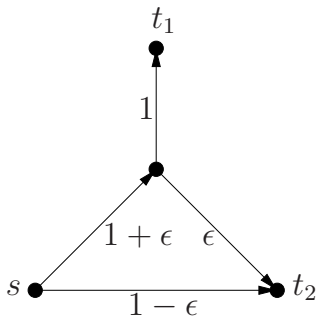

Figure 2.3: An example showing that the congestion of 2 is tight.

Proof. Consider the network depicted in Figure 2.3 with source s, sinks t_1, t_2, and demands 1 for both commodities. The number next to an arc a is the flow value $f^{\text{init}}(a)$, the capacity of a is the maximum of $f^{\text{init}}(a)$ and 1. ϵ is an arbitrary positive number smaller than 1.
Consider the arc with flow value $1+\epsilon$. Obviously we have to route commodity 1 on it in each unsplittable flow that participates in a convex combination forming f^{init}. But there must also be at least one unsplittable flow that routes commodity 2 on this arc. Thus, we obtain a flow value of 2 on it and a congestion of $2/(1+\epsilon)$. □

Chapter 3

k-Splittable Flows

3.1 Introduction

A natural generalization of the unsplittable flow problem is the k-splittable flow problem (k-SFP) introduced by Baier, Köhler, and Skutella [11]. In the k-SFP, there is an upper bound on the number of paths that may be used to route a commodity. Already for the single-commodity case, the k-SFP is NP-complete [11]. Of course, the optimization versions of the unsplittable flow problem (UFP) discussed in Chapter 2, i.e., minimum congestion, minimum number of rounds, and maximum routable demand, naturally generalize to the k-SFP.

The notion of k-splittable flows is motivated by transportation problems where divisible goods have to be shipped through a network using a bounded number of containers and each container must be routed along some path through the network.

Path Capacities. In the first part of this chapter we introduce a new variant of the k-splittable flow problem with additional upper bounds on the amount of flow being sent along each path. The motivation for these bounds comes from the following packing and routing problem: A commodity must be shipped using a given number of containers of given sizes. First, one has to make a decision on the fraction of the commodity packed into each container. Then, the containers must be routed through a network whose arcs correspond, for example, to ships or trains. Each arc has a capacity bounding the total size or weight of containers which are being routed on it. Each container being used, must be routed along some path through the network. In particular, the size of the container induces an upper bound on the amount of flow being sent along the chosen path. It is therefore called *path capacity*.

We consider two variants of the problem for which we provide an intuitive motivation:

1. When loading a ship, the *total weight* of all containers on the ship must not exceed the capacity of the ship. The weight of a container is determined by the actual amount of flow assigned to it. Thus, in the first variant with *weight capacities*, the capacity of an arc bounds the

actual amount of flow being sent on paths containing this arc. This is the classical interpretation of arc capacities.

2. When loading a train, the *total size* of the containers is usually more important than their total weight. Therefore, in the model with *size capacities*, the capacity of an arc bounds the total path capacity of all paths being routed through that arc. Notice that these capacity constraints are more restrictive than the classical ones in the first model.

We are mainly interested in the corresponding NP-hard optimization problem to minimize the congestion. A precise and formal definition of the problem under consideration is given in Section 3.3.

Length-Bounded Flows. A natural restriction in the area of transportation is to bound the length of paths that may be used to ship some commodity from its source to its destination. We therefore consider a generalization of k-splittable s-t-flows by imposing bounds on the lengths of the paths in $\mathcal{P}$, which is the set of all s-t-paths in the underlying network.

We consider the *maximum length-bounded k-splittable s-t-flow problem*: Given $k,\ L \in N$, find a maximum k-splittable s-t-flow which is L-length-bounded. This constitutes a natural combination and generalization of k-splittable and length-bounded s-t-flows.

Dynamic Flows. A crucial characteristic of network flows occurring in real-world applications is flow variation over time and the fact that flow does not travel instantaneously through a network but requires a certain amount of time (*transit time*) to travel through each arc. Both characteristics are captured by *dynamic flows* which specify a flow rate for each arc and each point in time. The *quickest s-t-flow problem* is to send a given amount of flow from s to t such that the last unit of flow arrives at the sink t as early as possible, i.e., within minimum time T. We consider the *quickest k-splittable s-t-flow problem*, where, as in the static setting described above, the number of s-t-paths used to send flow is bounded by k. This dynamic flow problem is NP-hard since already its 'static' counterpart is NP-hard.

Related Results from the Literature

As mentioned above k-splittable flows were introduced by Baier, Köhler, and Skutella [11]. They first consider a special form of the k-SFP in which each commodity uses the maximum number of paths allowed and is equally split up among the chosen paths. They call this problem the *uniform exactly-k-splittable flow problem*. It is shown that a maximum uniform exactly-k-splittable s-t-flow can be computed in polynomial time by a variant of the

classical augmenting path algorithm. In contrast, it is NP-hard to find a maximum k-splittable s-t-flow. But it is shown that the value of a maximum k-splittable s-t-flow is at most twice as large as the value of a maximum uniform exactly-k-splittable s-t-flow. Thus, computing a maximum uniform exactly-k-splittable s-t-flow yields a 2-approximation algorithm for the maximum k-splittable s-t-flow problem.

Koch, Skutella, and Spenke [63] improve on these results for graphs of bounded treewidth. They show that on those graphs and for a constant number of paths, the problem can be solved in polynomial time. If the number of paths is part of the input, the problem is still NP-hard. Koch et al. present a polynomial time approximation scheme for this case.

For the general setting with an arbitrary number of source-sink-pairs (commodities) the uniform exactly-k-SFP contains the UFP as a special case and is therefore NP-hard. But note that any instance of it can be solved by solving a special instance of the UFP on the same graph. Thus, it holds that any ρ-approximation algorithm for the problem of minimizing the congestion for the UFP provides a ρ-approximation algorithm for the problem of minimizing the congestion for this special k-SFP.[1] To approximate optimal solutions of the general k-SFP, Baier et al. [11] again use the fact that a uniform exactly-k-splittable flow of minimum congestion for any instance of the k-SFP is a k-splittable flow which approximates the congestion of an optimal k-splittable flow for the same instance within a factor 2.

Further results are known for similar problems in undirected networks. Bagchi, Chaudhary, Scheideler, and Kolman [7] consider fault tolerant routings. To ensure connection for each commodity for up to $k-1$ edge failures in the network, they require $k \in \mathbb{N}$ edge disjoint flow paths per commodity. This problem is called the k-disjoint flow problem (k-DFP). Bagchi et al. [7] also consider the integral splittable flow problem (ISF). Bagchi [6] extends the considerations of [7] to the k-SFP. The aim of Bagchi et al. is always to maximize the sum of satisfied demands subject to meeting all edge capacity constraints. In the k-DFP a demand d for any request is to be satisfied by a flow on k disjoint paths, each path carrying d/k units of flow. In the ISF integral demands need to be satisfied by an arbitrary number of paths, where the flow value of any path has to be integral. In contrast to [11] and the considerations made here, Bagchi does not admit different bounds on the numbers of paths for different commodities in the k-SFP. For all of the mentioned problems, Bagchi et al. introduce simple greedy algorithms in the

[1] For this result it is important to assume that the bound on the number of paths is at most m for each commodity. This can be done without loss of generality, because each s-t-flow can be decomposed into flows on at most m paths and cycles. (The cycles can be omitted.)

style of greedy algorithms for the UFP given by Kolman and Scheideler [67]. With these algorithms they obtain approximation ratios of $O(k^3 F \log(kF))$ for the k-DFP and $O(k^2 F)$ for the k-SFP on the conditions that they have unit edge capacities and the maximum demand is at most k times larger than the minimum edge capacity. Here F is a network measure introduced by Kolman and Scheideler [67]—the *flow number*. This number gives a hint on the interconnectedness of a network. For the ISF Bagchi et al. obtain an approximation ratio of $O(F)$ for any instance with uniform edge capacities in which the maximum demand is at most the capacity of the edges. The ISF has earlier been studied by Guruswami et al. [48] who obtain an approximation ratio of $O(\sqrt{m\Delta}\log^2 m)$ for it, where Δ is the maximum degree of a vertex in the considered graph.

Lovász, Neumann-Lara, and Plummer [72] were the first who studied problems with length bounds for the flow paths. They generalize the vertex version of Menger's Theorem[2] to paths of bounded length with respect to unit edge lengths. Exoo [30] and Niepel and Safaríková [82] generalize the results of this article to edge disjoint paths. Itai, Perl, and Shiloach [56] prove that it is strongly NP-hard to compute the maximum number of edge-disjoint length-bounded paths, whereas Bley [16] proves that computing the maximum number of vertex-disjoint length-bounded paths is even APX-complete. Some heuristics for different problems imposing length bounds can be found in [85, 17]. The first results for length-bounded flows were obtained by Mahjoub and McCormick [74] and Baier et al. [10, 9]. Baier et al. present various complexity and algorithmic results, including that the integrality gap of the problem is in $\Omega(\sqrt{m})$ and that there is no polynomial algorithm which computes a length-bounded path-flow from a given edge-flow that is known to correspond to a length-bounded path-flow. Baier [9] gives an extensive survey of what is known for length-bounded flows and presents, e.g., an FPTAS for the maximum length-bounded multicommodity flow problem.

Ford and Fulkerson [36, 37] introduce dynamic s-t-flows. It follows from their work that the quickest s-t-flow problem can be solved in polynomial time. They consider the problem of sending the maximum amount of flow from a source s to a sink t within a given time horizon. In order to solve the quickest s-t-flow problem, where a given amount of flow must be sent from s to t as fast as possible, one can use binary search. Burkard, Dlaska, and Klinz [18] present a strongly polynomial algorithm for the quickest s-t-flow problem using Meggido's method of parametric search [79]. For the single

[2]The vertex version of Menger's Theorem says that the maximum number of vertex disjoint paths from one node s in a graph to another node t in the same graph equals the minimum number of vertices whose deletion destroys all paths from s to t.

commodity case with multiple sources and sinks Hoppe and Tardos [54, 53] present the first polynomial algorithm. In contrast to the situation in static flows, the single commodity dynamic flow problem with multiple sources and sinks does not reduce to an s-t-flow problem. As the algorithm of Hoppe and Tardos uses a discrete time model, Fleischer and Tardos [32] show that it also holds for the continuous time model. The first results on quickest multicommodity flows were given by Hall, Hippler, and Skutella [49] and Fleischer and Skutella [31]. Hall et al. prove the NP-hardness of the quickest multicommodity flow problem. Fleischer and Skutella show that this problem can efficiently be approximated with constant performance guarantee via static length-bounded flow computations.

Organization of this Chapter

In Section 3.2 we define the k-SFP in general and introduce basic notation. The restricted versions of the k-SFP are defined in Sections 3.3.1, 3.4.1, and 3.5.1 respectively. In Section 3.3 we present algorithms that compute approximate minimum congestion k-splittable flows obeying given path capacities (weight/size capacities). The approximation ratio of these algorithms is asymptotically the same as for the unsplittable flow problem. In Section 3.4 we develop an algorithm computing approximations to the maximum length-bounded k-splittable s-t-flow problem. The algorithm yields a bicriteria approximation. We allow the computed flow to violate the given length bound by some constant factor. Then it routes a constant factor of the maximum flow value that is possible obeying the length bound. In Section 3.5 we use the results from Section 3.4 to give a 5.828-approximation algorithm for the quickest k-splittable s-t-flow problem.

Parts of Section 3.3 have already been published in [75, 76]. Parts of Sections 3.4 and 3.5 were published in [77].

3.2 Problem Definition and Notation

An instance of the k-splittable flow problem consists of a digraph $D = (V, A)$ with arc capacities $u : A \to \mathbb{R}^+$ and a set $\mathcal{T} \subseteq V \times V$ of $K \in \mathbb{N}$ *requests* or *commodities*. The ith request is denoted by (s_i, t_i). Together with request i we are given its nonnegative *demand* d_i and the number of paths k_i (containers) it can be routed on.

If we have only one request we use the term *k-splittable s-t-flow problem* and denote the single request by (s, t), its demand by d, and the bound on the number of paths by k.

A feasible solution to the k-SFP consists of s_i-t_i-paths $P_1^i, \ldots, P_{k_i}^i$ with corresponding nonnegative flow values $f_1^i, \ldots, f_{k_i}^i$, for each commodity $i \in \{1, \ldots, K\}$, such that the following requirements are met:

$$\sum_{j=1}^{k_i} f_j^i \;=\; d_i \qquad \text{for all } i \in \{1, \ldots, K\}, \tag{3.1}$$

that is, the whole demand of request i is routed. And

$$\sum_{i=1}^{K} \sum_{\substack{j=1,\ldots,k_i:\\ a \in P_j^i}} f_j^i \;\leq\; u(a) \qquad \text{for all } a \in A,$$

that is, all arc capacities are obeyed.

We do not require that the paths $P_1^i, \ldots, P_{k_i}^i$ are distinct, for $i = 1, \ldots, K$. Furthermore, we allow a path to have flow value 0. In other words, we may use less than k_i paths for commodity i.

We use the notion *k-splittable flow* twofold. On the one hand, for a set of paths $\mathcal{R}$ in which we have at most k_i paths from s_i to t_i for each $i \in \{1, \ldots, K\}$ and corresponding flow values f_P for each $P \in \mathcal{R}$. On the other hand, we use it for a multicommodity flow (given arcwise) which can be decomposed into flows on at most k_i paths from s_i to t_i, for each $i \in \{1, \ldots, K\}$. Sometimes we call a flow *feasible* to underline that it satisfies all arc capacity constraints. To avoid confusion, we use the term *k-splittable s-t-flow* for a k-splittable flow that is given in a network with only one request.

In the single commodity case, we consider the problem to search for a k-splittable s-t-flow of maximum value—called a *maximum k-splittable s-t-flow*. This problem is named the *maximum k-splittable s-t-flow problem*. The *value* of a k-splittable s-t-flow is given by $\sum_{i=1}^{k} f_i$, where f_i denotes the flow value of the ith s-t-path P_i in this flow.

Following the notation of Baier et al. [11, 9], we define the *uniform k-splittable flow problem* as a k-SFP in which all flow carrying s_i-t_i-paths must route the same amount of flow for each $i \in \{1, \ldots, K\}$. The *uniform exactly-k-splittable flow problem* is the uniform k-SFP in which we additionally require that exactly k_i paths are used to route commodity $i \in \{1, \ldots, K\}$. Analogously, we use the notions *uniform (exactly)-k-splittable s-t-flow problem* and *uniform (exactly)-k-splittable (s-t-)flow*.

Remember that the s_i-t_i-paths used in a solution to an instance of the k-SFP need not to be distinct. That means for the uniform versions of the k-SFP, that a solution may contain several copies of an s_i-t_i-path for an $i \in \{1, \ldots, K\}$ or in other words that the total amount of flow being sent along this path may be a multiple of the common flow value.

3.3 k-Splittable Flows with Path Capacities

Since we have already motivated the study of k-splittable flows with path capacities in the introduction of this chapter, we immediately start with a definition of the k-splittable flow problem with path capacities in Section 3.3.1. For both variants of the problem discussed above, we prove that they have essentially the same approximability as the unsplittable flow problem. To be more precise, we show that any ρ-approximation algorithm for the UFP can be turned into a 2ρ-approximation algorithm for either variant of our problem. The underlying idea is to solve the problem in two steps. First, a packing of flow into containers is computed. Then, each container is routed along a path through the network. The latter step can be formulated as an unsplittable flow problem. In Section 3.3.2 and Section 3.3.3, respectively, we present simple packing routines for both variants of the problem. We show that we do not lose more than a factor 2 with respect to congestion when we restrict to the constructed packing.

3.3.1 Problem Specification

An instance of the k-splittable flow problem with path capacities consists of the same elements as an instance of the standard k-SFP. Additionaly, we are given a *path capacity* $t^i_j \geq 0$, for each $i \in \{1,\dots,K\}$, $j \in \{1,\dots,k_i\}$, by which the flow value on the jth path of commodity i must be bounded. We always assume that the path capacities of every commodity suffice to route the entire demand, that is, $\sum_{j=1}^{k_i} t^i_j \geq d_i$, for all $i = 1,\dots,K$.

A feasible solution to the k-SFP with path capacities consists of s_i-t_i-paths $P^i_1,\dots,P^i_{k_i}$ with corresponding nonnegative flow values $f^i_j \leq t^i_j$, for each commodity $i \in \{1,\dots,K\}$ and each $j \in \{1,\dots,k_i\}$, such that (3.1) is satisfied, i.e., all demands are routed entirely. Additionally, we require that

$$\sum_{i=1}^{K} \sum_{\substack{j=1,\dots,k_i:\\ a\in P^i_j}} f^i_j \;\leq\; u(a) \qquad \text{for all } a \in A, \tag{3.2}$$

or

$$\sum_{i=1}^{K} \sum_{\substack{j=1,\dots,k_i:\\ f^i_j>0 \,\wedge\, a\in P^i_j}} t^i_j \;\leq\; u(a) \qquad \text{for all } a \in A, \tag{3.3}$$

where (3.2) must hold if we consider the problem with *weight capacities*, and (3.3) must hold if we consider the problem with *size capacities*. That

means, the corresponding inequalities ensure that no arc capacity constraint is violated.

We consider the satisfaction of all demands and the compliance with all path capacities as hard constraints which must not be violated. Our objective is to violate constraint (3.2) (or (3.3) respectively) only by a preferably small factor—the *congestion*—which will be defined in more detail later on.

Note that the k-SFP with path capacities is strongly NP-hard, because it contains the unsplittable flow problem as a special case (set $k_i = 1$ and $t_1^i = d_i$, for all $i = 1, \ldots, K$). Even the single commodity version of the k-SFP with path capacities is strongly NP-hard since it is a generalization of the strongly NP-hard single source UFP [11]. Therefore we are interested in approximations to the problem. The *congestion* of a flow is the maximum factor by which the capacity of an arc in the graph is violated. Consider any k-splittable flow. For either variant of the k-SFP, we can simply define the congestion of this flow as the smallest value $\alpha \geq 1$ for which multiplying the right hand sides of (3.2) (or (3.3), respectively) makes these inequalities true. Our objective is always to minimize the congestion of the respectively considered problem.

3.3.2 k-Splittable Flows With Weight Capacities

In this section we consider the first variant of the k-SFP with path capacities, i.e., the case of *weight capacities*. We present a simple packing strategy which allows us to reduce the k-SFP with path capacities to the UFP while losing at most a factor 2 with respect to minimum congestion. To obtain an instance of the UFP, we decompose the demand d_i of commodity i into k_i pieces $load_1^i, \ldots, load_{k_i}^i$ and then replace request i by k_i copies of i, whose demands are set to the values of the demand pieces and must finally be sent unsplittably. It is explained later on how $load_1^i, \ldots, load_{k_i}^i$ are chosen. To make the procedure more vivid, one may always think of packing k_i containers for each commodity i which are then routed along certain paths through the network. In the following we use Π_k and Π_u to denote the currently considered instance of the k-SFP with path capacities and its corresponding instance of the UFP that results from our packing.

After defining the load for each container, we consider an optimal solution to Π_k and interpret its flow values as capacities of bins. This interpretation comes from the idea that we can assign our loads to these resulting bins for the purpose of sending each load (container) along the same path as the corresponding bin is taking in the optimal solution. We show that one can find an assignment of loads to bins such that the bins are overloaded by a

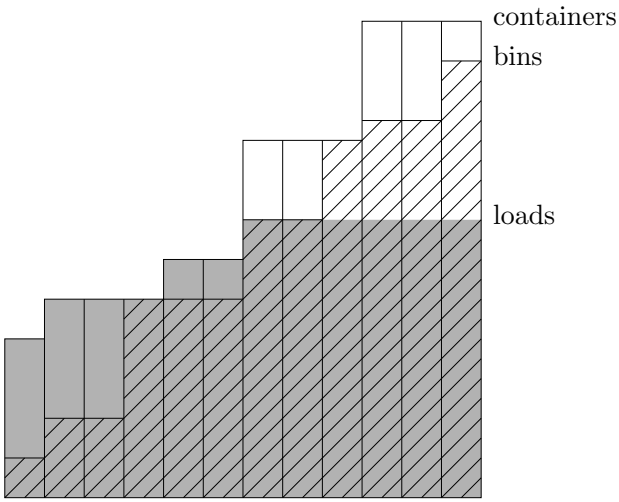

Figure 3.1: An example for the loading of the containers. The beams represent the containers, the filled part the loading defined in (3.4), and the striped part the loads of an optimal solution.

factor of at most 2. Consequently, there exists a solution to Π_k which uses our packing of containers and has a congestion of at most twice the minimum congestion.

For the purpose of assigning the loads to the bins with a small overload factor, it appears to be reasonable to break the demand of any commodity into pieces which are as small as possible. In other words, we want to load the containers such that the maximal load is minimized. One can visualize this by picturing that all containers of the same commodity are lined up in a row and merged with each other and get filled up with water; see Figure 3.1. To simplify notation, we assume without loss of generality that $t_1^i \leq t_2^i \leq \cdots \leq t_{k_i}^i$, for all $i = 1, \ldots, K$. Then, we can assume that the loads $load_j^i$ of commodity i are indexed in nondecreasing order, too. A precise description of the loading strategy depicted in Figure 3.1 is as follows:

$$load_j^i = \min \left\{ t_j^i,\ \frac{1}{k_i - j + 1} \left(d_i - \sum_{j'=1}^{j-1} load_{j'}^i \right) \right\}. \tag{3.4}$$

For any $i \in \{1, \ldots, K\}$, we can simply start with $j = 1$ and then compute the loads in increasing order. Note that two containers of same size get the same load. Thus, it even suffices to compute the load for only one container per size.

Now consider any optimal solution F to the underlying problem of minimizing the congestion for Π_k. Denote its flow values by $bin_1^i, \ldots, bin_{k_i}^i$, for all $i = 1, \ldots, K$. If we arrange the bins in nondecreasing order, for all $i = 1, \ldots, K$, then we can prove the following lemma.

Lemma 3.1. For all $i \in \{1, \ldots, K\}$ the following holds: If $load^i_j \geq bin^i_j$ for some $j \in \{1, \ldots, k_i\}$, then $load^i_{j'} \geq bin^i_{j'}$, for all $j' < j$.

Proof. Since F is an optimal and thus feasible solution, no bin is larger than its path capacity. Hence, a violation of the claim would contradict our loading strategy: Consider $i \in \{1, \ldots, K\}$ and $j \in \{1, \ldots, k_i\}$ such that $load^i_j \geq bin^i_j$. For all containers $j' < j$, we have either $load^i_{j'} = load^i_j \geq bin^i_j \geq bin^i_{j'}$ or $load^i_{j'} = t^i_{j'} \geq bin^i_{j'}$. (See also Figure 3.1.) □

Next we prove that we can pack k_i items of sizes $load^i_1, \ldots, load^i_{k_i}$ into k_i bins of sizes $bin^i_1, \ldots, bin^i_{k_i}$ without exceeding the bin sizes by more than a factor 2. This implies that we can simultaneously route all demands in Π_u, in spite of not allowing a congestion greater than twice the congestion of the optimal solution F. (Remember that Π_u resulted from replacing each request in Π_k by exactly k_i copies and assigining the demands $load^i_1, \ldots, load^i_{k_i}$ to these copies.)

Lemma 3.2. Let $a_1 \leq a_2 \leq \cdots \leq a_n$ and $b_1 \leq b_2 \leq \cdots \leq b_n$ be nonnegative real numbers with $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and satisfying the following: If $a_I \geq b_I$ for some $I \in \{1, \ldots, n\}$, then $a_i \geq b_i$ for all $i = 1, \ldots, I$. Then there exists an assignment $p : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $\sum_{j \in p^{-1}(i)} a_j \leq 2b_i$, for all $i = 1, \ldots, n$.

Proof. Assume we have $a_1 \leq a_2 \leq \cdots \leq a_n$ and $b_1 \leq b_2 \leq \cdots \leq b_n$ as above. If $a_i \leq 2b_i$ for all $i \in \{1, \ldots, n\}$, we are done since the identity function $p = id$ fulfills the requirement. Thus, we assume that there exists an i with $a_i > 2b_i$; let $I := \max\{i \mid a_i > 2b_i\}$. Notice that $a_i \geq b_i$, for all $i \in \{1, \ldots, I\}$. For all $i \in \{1, \ldots, n\}$ with $b_i \leq a_i \leq 2b_i$, we set $p(i) := i$ and exclude these indices from further consideration. The remaining indices are partitioned into a set of *small indices* $\mathcal{I}_s := \{i \mid i \leq I \text{ and } a_i > 2b_i\}$ and a set of *large indices* $\mathcal{I}_\ell := \{i \mid i > I \text{ and } a_i < b_i\}$. Notice that $\mathcal{I}_\ell \neq \emptyset$ since otherwise $\sum_{i=1}^n a_i > \sum_{i=1}^n b_i$. Finally, define the *overload* as $\sum_{i \in \mathcal{I}_s}(a_i - b_i)$ and the *empty space* as $\sum_{i \in \mathcal{I}_\ell}(b_i - a_i)$; notice that the empty space is at least as large as the overload. Moreover, since $a_i > 2b_i$, for each $i \in \mathcal{I}_s$, it follows that $\sum_{i \in \mathcal{I}_s} a_i$ is bounded from above by twice the empty space.

For $i \in \mathcal{I}_\ell$, define $p(i) := i$ and set $u_i := 2b_i - a_i$. Consider the elements $i \in \mathcal{I}_s$ in decreasing order and assign i to the largest index in $\mathcal{I}_\ell$ for which the sum of the elements currently assigned to it does not exceed u_i, i.e., $p(i) := \max\{i' \in \mathcal{I}_\ell \mid \sum_{j \in p^{-1}(i')} a_j \leq u_{i'}\}$.

After this procedure, $\sum_{j \in p^{-1}(i)} a_j \leq 2b_i$, for all $i \in \mathcal{I}_\ell$, because the value a_j which was assigned to b_i last is at most a_i. Since the values assigned to b_i before sum up to at most u_i, we obtain the desired result. On the other hand,

we must have assigned each index in $\mathcal{I}_s$ to an index in $\mathcal{I}_\ell$ since $\sum_{j\in\mathcal{I}_s} a_j$ is at most twice the empty space. □

Note that, for each $i \in \{1,\dots,K\}$, the loads and the bins satisfy the requirements from Lemma 3.2. Thus, we can assign all loads to the bins without exceeding the bin capacities by a factor greater than 2. Now consider the mentioned Π_u: Take Π_k. For all $i \in \{1,\dots,K\}$, replace request i by k_i copies of i with demands equal to $load^i_1,\dots,load^i_{k_i}$. Using the results from above, we obtain the following:

Theorem 3.3. For the problem of minimizing congestion and each instance of the k-SFP with path capacities (weight capacities), there is a corresponding instance of the UFP such that a ρ-approximation to this instance yields a 2ρ-approximation to the underlying instance of the k-SFP with path capacities.

We conclude this section with some remarks on the running time of the resulting 2ρ-approximation algorithms for the k-SFP with path capacities. Notice that the size of Π_u is not necessarily polynomial in the input size of Π_k. The number of copies k_i of request i can be exponential, if in Π_k we are only given the different sizes of containers and the number of containers for each size. As already mentioned above, it will not be a problem to compute the loads in polynomial time. However, in order to achieve polynomial running time when solving the problem of minimizing congestion on Π_u, we must use a polynomial time algorithm for the UFP which is able to handle a compact input format where, for each request, the number of copies of the request is given. In fact, most approximation algorithms for the UFP fulfill this requirement or can easily be modified to fulfill it. As an example, we mention the randomized rounding procedure: In a fractional solution, all copies of a request can be treated as one commodity for which a path decomposition using at most m paths exists. Randomized rounding then assigns to each of these paths a certain number of copies of the corresponding request. This procedure can easily be implemented to run in polynomial time.

3.3.3 k-Splittable Flows With Size Capacities

In this section we consider the second variant of the k-SFP with path capacities, i.e., the case of *size capacities*. If all path capacities are equal to some uniform value T_i, for all commodities i, it is easy to observe that there always exists an optimal solution using the minimum number of paths $\lceil d_i/T_i \rceil$ for commodity i. Therefore, this problem can be formulated as a uniform exactly-k-SFP (see [11] for details). In particular, all results for the latter

problem obtained in [11] also hold for the k-SFP with uniform path capacities. The most important result of [11] in this context is that any instance of the considered problem can be solved by solving a special instance of the UFP on the same graph, and thus any ρ-approximation algorithm for the problem of minimizing the congestion for the UFP provides a ρ-approximation algorithm for the problem of minimizing the congestion for our problem here.

We turn to the general problem with non-uniform path capacities. As in the last section, we assume that $t^i_1 \leq \cdots \leq t^i_{k_i}$, for all $i = 1, \ldots, K$. Our general approach is identical to the one presented in the last section. Again, we first give a simple strategy for the packing of containers. This reduces the problem to a UFP. We prove that at most a factor 2 in the performance is lost due to this packing by comparing it to the packing of an optimal solution (*bins*). Tying in with the previous section, we use Π_k to denote the currently considered instance of the k-SFP with path capacities and Π_u to denote its corresponding instance of the UFP which results from the packing introduced below.

In contrast to the last section, it seems to be reasonable to load each container that is used up to its size. Remember that only the size of a container (and not its actual load) matters for the routing. Among all containers, we try to use the smallest ones in order to be able to obtain reasonable assignments to bins later. The load $load^i_j$ of the jth container of commodity i is determined as follows: Consider the containers of each commodity in order of decreasing size. If the whole (remaining) demand of the commodity fits into the remaining containers with smaller indices, then discard the container. Otherwise, the container is filled up to its size (unless the remaining demand is smaller than the size) and the remaining demand is decreased by this value. This loading strategy can be easily described by the following pseudo-code. For all $i \in \{1, \ldots, K\}$, start with $j = k_i$ and then calculate the flow values in decreasing order as follows:

$$\alpha = d_i - \sum_{l=j+1}^{k_i} load^i_l - \sum_{l=1}^{j-1} t^i_l$$
If $(\alpha > 0)$ then
$$load^i_j = \min \left\{ t^j_i, d_i - \sum_{l=j+1}^{k_i} load^i_l \right\}$$
else
$$load^i_j = 0$$

The strategy above finds the loads in polynomial time if the path capacity of every path is explicitly given in the input. If the encoding of the input

is more compact and, for each commodity i, only specifies the different path capacities together with the number of paths for each capacity, the algorithm can be easily adapted to run still in polynomial time. (One simply has to consider all containers of the same size together and compute the required number of containers of this size as well as the total load that is assigned to them.)

In the following we consider a fixed optimal solution to Π_k. For all $i \in \{1,\dots,K\}$, let $\mathcal{O}_i \subseteq \{1,\dots,k_i\}$ such that $j \in \mathcal{O}_i$ if and only if the jth container of commodity i is used by the optimal solution. Similarly, let $\mathcal{B}_i \subseteq \{1,\dots,k_i\}$ such that $j \in \mathcal{B}_i$ if and only if our loading uses the jth container of commodity i, i.e., $load^i_j > 0$. The following lemma says that the containers used by our loading can be packed into the containers (bins) used by the optimal solution such that the size of any container of the optimal solution is exceeded by at most a factor 2.

Lemma 3.4. For all $i = 1,\dots,K$ with $\mathcal{B}_i \neq \emptyset$, i.e., with $d_i > 0$, there exists an assignment $p : \mathcal{B}_i \to \mathcal{O}_i$ such that

$$\sum_{j' \in p^{-1}(j)} t^i_{j'} \;\leq\; 2t^i_j \qquad \text{for all } j \in \mathcal{O}_i. \tag{3.5}$$

Proof. In the following we keep i fixed; to simplify the notation, we omit all indices and superscripts i. By construction of our packing strategy,

$$\sum_{j \in \mathcal{B}: j > j_0} t_j \;<\; \sum_{j \in \mathcal{O}: j \geq j_0} t_j \qquad \text{for all } j_0 \in \mathcal{B}. \tag{3.6}$$

In particular, it follows from (3.6) that the maximum index in $\mathcal{O}$ is at least as large as the maximum index in $\mathcal{B}$ and therefore the largest container (bin) in the optimal solution is at least as large as the largest container in our packing. While $\mathcal{B} \neq \emptyset$, we construct the assignment p iteratively as follows:

$$j_{\max} \;:=\; \max\{j \mid j \in \mathcal{O}\} \qquad \text{and} \qquad \bar{j} := \min\{j \in \mathcal{B} \mid \sum_{j' \in \mathcal{B}: j' > j} t_{j'} \leq t_{j_{\max}}\} \;.$$

Set $p(j) := j_{\max}$ for all $j \in \mathcal{B}$ with $j \geq \bar{j}$; set $\mathcal{O} := \mathcal{O} \setminus \{j_{\max}\}$ and $\mathcal{B} := \mathcal{B} \setminus \{j \mid j \geq \bar{j}\}$. Notice that property (3.5) holds for $j_{\max}$ since, by (3.6), the container size $t_{j_{\max}}$ is an upper bound on t_j, for all $j \in \mathcal{B}$. Moreover, (3.6) is still fulfilled for the reduced sets $\mathcal{B}$ and $\mathcal{O}$ and we can proceed inductively. This concludes the proof. □

For all $i = 1,\dots,K$ and $j = 1,\dots,k_i$, set $\tilde{d}^i_j := t^i_j$ if $j \in \mathcal{B}_i$, and $\tilde{d}^i_j := 0$, otherwise. We define Π_u by replacing each request i by k_i copies with

demands $\tilde{d}^i_1, \ldots, \tilde{d}^i_{k_i}$. According to Lemma 3.4, there exists a solution to Π_u whose congestion is at most twice the congestion of an optimal solution to Π_k.

Theorem 3.5. For the problem of minimizing congestion and each instance of the k-SFP with path capacities (size capacities), there is a corresponding instance of the UFP such that a ρ-approximation to this instance yields a 2ρ-approximation to the underlying instance of the k-SFP with path capacities.

With respect to the running time of the resulting 2ρ-approximation algorithms for the k-SFP with path capacities, the same remarks hold as we made in the previous section after Theorem 3.3.

We conclude this section with the discussion of a special case of the k-SFP with path capacities where, for each request i, each of its path capacities is a multiple of all of its smaller path capacities. This property holds, for example, if all path capacities are powers of 2. In this case, we can use the same loading strategy as above in order to obtain a ρ-approximation to the underlying problem from any ρ-approximation algorithm for the UFP. This holds, because—with the same terms as used above—for all $i = 1, \ldots, K$, there exists an assignment $p : \mathcal{B}_i \to \mathcal{O}_i$ such that $\sum_{j' \in p^{-1}(j)} t^i_{j'} \leq t^i_j$, for all $j \in \mathcal{O}_i$. The argument is similar to that in the proof of Lemma 3.4: We can use the same strategy. If we choose $\bar{j} := \min\{j \in \mathcal{B} \mid \sum_{j' \in \mathcal{B}: j' \geq j} t_{j'} \leq t_{j_{\max}}\}$ we obtain (except for the last step) that $\sum_{j' \in \mathcal{B}: j' \geq \bar{j}} t_{j'} = t_{j_{\max}}$. Thus, we get rid of the factor 2 in (3.5). (In the last step it might happen that $\sum_{j' \in \mathcal{B}: j' \geq \bar{j}} t_{j'} < t_{j_{\max}}$, but since all containers are assigned after this step, we do not need to take care of this.)

3.4 Length-Bounded k-Splittable Flows

In this section we only consider the single commodity case of k-splittable flows. Again we start with a detailed definition of the considered problem in Section 3.4.1. Afterwards we constructively prove in Section 3.4.2 that there exists a polynomial time algorithm that computes a $(1/\epsilon)L$-length-bounded k-splittable s-t-flow whose flow value is at least $(1-\epsilon)/2$ times the flow value of a maximum L-length-bounded k-splittable s-t-flow. Here, ϵ is an arbitrary positive real number smaller than 1.

Theorem 3.6. For any $\epsilon > 0$, there is a polynomial time algorithm that computes a $(1/\epsilon)L$-length-bounded k-splittable s-t-flow whose flow value is at least $(1-\epsilon)/2$ times the value of a maximum L-length-bounded k-splittable s-t-flow.

As the proof of Theorem 3.6 is constructive, we immediately obtain an approximation algorithm for the considered problem.

3.4.1 Problem Specification

An instance of the length-bounded k-splittable s-t-flow problem basicly consists of all elements from the standard *k-splittable s-t-flow problem*. In addition, we are given *arc lengths* $\ell : A \to \mathbb{R}_0^+$ and a nonnegative *length bound* L.

A feasible solution to the length-bounded k-splittable s-t-flow problem is a k-splittable s-t-flow, specified by a set of s-t-paths $\mathcal{P} = \{P_1, \ldots, P_k\}$ and corresponding flow values $f_1, \ldots, f_k$, that is L-length-bounded. Here, *L-length-bounded* means that $\sum_{a \in P_i} \ell(a) \leq L$ for $i = 1, \ldots, k$, i.e., no path in $\mathcal{P}$ is longer than L.

Without loss of generality, we consider only instances of the length-bounded k-splittable s-t-flow problem for which $k \leq m$, because by [9] for every L-length-bounded s-t-flow on more than m paths there is an L-length-bounded s-t-flow of the same value that uses at most m paths and does not exceed the original flow on each arc. (This result follows immediately from theory of linear programming.)

3.4.2 A Constant Factor Approximation

We derive a simple combinatorial algorithm that computes a length-bounded k-splittable s-t-flow satisfying the properties stated in Theorem 3.6. In this context it is useful to interprete lengths of arcs also as cost coefficients. Doing so, we can apply algorithms for min-cost flow problems in order to approximate maximum length-bounded flows. This insight is used in this section. We first show that a maximum uniform exactly-k-splittable s-t-flow obeying the given length bound L only on average can be found in polynomial time.

Lemma 3.7. For given $k, L \in \mathbb{N}$, a maximum uniform exactly-k-splittable s-t-flow with average path length at most L can be computed in polynomial time.

The proof of Lemma 3.7 is similar to the proof of [11, Theorem 6]. It is based on the insight that a uniform exactly-k-splittable s-t-flow with flow value kf is an f-integral s-t-flow (a flow is called *f-integral* for some $f \in \mathbb{R}^+$ if the flow value on each arc is an integral multiple of f). Moreover, any f-integral s-t-flow of value kf induces a uniform exactly-k-splittable s-t-flow of the same value by constructing an f-integral decomposition into paths and cycles and ignoring the cycles.

Proof of Lemma 3.7. Consider a maximum uniform exactly-k-splittable s-t-flow with average path length at most L and positive flow value, if such a flow exists. There is at least one arc $a \in A$ with a tight capacity constraint since otherwise a better solution can be obtained by increasing the common flow value f on all k paths. Hence, f is equal to the capacity $u(a)$ of arc a divided by the number of paths using this arc. Thus, $f = u(a)/i$ for some arc $a \in A$ and some $i \in \{1, \dots, k\}$.

Based on this insight, we formulate an algorithm: For all $a \in A$ and $i = 1, \dots, k$, compute a $u(a)/i$-integral min-cost s-t-flow of value $F_{a,i} := ku(a)/i$ or find out that no such flow exists. Among all computed flows of total costs at most $F_{a,i}L$ output one with largest flow value. If no such flow exists, output the zero flow.

We show that this procedure indeed outputs a maximum uniform exactly-k-splittable s-t-flow with average path length at most L: Assume that there exists a uniform exactly-k-splittable s-t-flow with average path length at most L and positive flow value. From the considerations above, we know that then there is a $u(a)/i$-integral flow of value $F_{a,i}$ and costs no more than $F_{a,i}L$, for some $a \in A$ and $i \in \{1, \dots, k\}$, namely at least for the a and the i with $u(a)/i = f$. Therefore, our algorithm returns a flow of at least this value. This flow is uniform exactly-k-splittable and has average path length at most L, because for some $a \in A$ and $i \in \{1, \dots, k\}$, it is of value $F_{a,i} = ku(a)/i$ and $u(a)/i$-integral, while its costs are no more than $F_{a,i}L$.

Consider the case that no uniform exactly-k-splittable s-t-flow with average path length at most L and positive flow value exists. Then we want the algorithm to output the zero flow. But since all flows among which the one of largest value is chosen have an average path length of at most L, are uniform exactly-k-splittable and have positive flow value, the algorithm will output the zero flow, if no such flow exists.

The running time of the algorithm is dominated by $k|A|$ min-cost s-t-flow computations and therefore of polynomial time. □

Now we know how to compute a maximum uniform exactly-k-splittable s-t-flow with average path length at most L. For some $f \in \mathbb{R}_0^+$, this flow is f-integral, of value kf, and of costs at most kfL. We show that, for any $\epsilon > 0$, the flow can be turned into a $(1/\epsilon)L$-length-bounded k-splittable s-t-flow while decreasing the flow value only by at most a factor $1 - \epsilon$.

Lemma 3.8. Given an f-integral s-t-flow of value kf and costs at most kfL, a $(1/\epsilon)L$-length-bounded k-splittable s-t-flow of value at least $(1 - \epsilon)kf$ can be found in polynomial time.

Proof. For now we assume that $\epsilon = q/k$, for some $q \in \mathbb{N}$. This makes ϵk integral.

In this case the algorithm works as follows. First, the given f-integral flow is made acyclic by repeatedly canceling flow on cycles. Notice that this step does not increase the costs since all cost coefficients (arc lengths) are nonnegative. Next, we cancel f units of flow along the currently longest flow-carrying s-t-path and repeat this step ϵk times.

The resulting s-t-flow is still f-integral and can be decomposed into flows on at most k paths. Moreover, it has flow value $kf - \epsilon kf = (1-\epsilon)kf$. The length of any flow-carrying s-t-path is at most $(1/\epsilon)L$. Otherwise, all paths on which flow was canceled would have had length strictly larger than $(1/\epsilon)L$. Since ϵkf flow units were deleted from these paths, the costs of the initial flow would have been strictly larger than kfL—a contradiction.

If ϵk is not integral, we can proceed almost the same way. The only difference is that we repeat the canceling of flow only $\lfloor \epsilon k \rfloor$ times. Then we again obtain a $(1/\epsilon)L$-length-bounded flow F, as can be proven using the following argument. Assume that in a further step, we would cancel another $\epsilon kf - \lfloor \epsilon k \rfloor f < f$ units of flow along the longest remaining flow-carrying s-t-path P. Then we obtain a flow value of $kf - \lfloor \epsilon k \rfloor f - (\epsilon kf - \lfloor \epsilon k \rfloor f) = kf - \epsilon kf$. With the same argument as above it follows that the length of any flow-carrying s-t-path would be at most $(1/\epsilon)L$. In particular, the length of P is at most $(1/\epsilon)L$. Thus, we could as well send f units of flow along P (i.e., not cancel the $\epsilon kf - \lfloor \epsilon k \rfloor f$ units) and would not violate the length bound. The resulting flow equals F. It is $(1/\epsilon)L$-length-bounded and routes even more than $(1-\epsilon)kf$ units of flow. □

Note that the computed s-t-flow is not only $(1/\epsilon)L$-length-bounded but has the stronger property that the length of *any* flow-carrying path is at most $(1/\epsilon)L$. This means that *any* path decomposition of this flow has the nice property of being $(1/\epsilon)L$-length-bounded. It is easy to come up with examples showing that this does not hold for arbitrary length-bounded s-t-flows.

Now we have almost come up with a proof of Theorem 3.6. To obtain the desired flow, first a maximum uniform exactly-k-splittable s-t-flow with average path length at most L is computed (see Lemma 3.7). The second step turns this flow into a $(1/\epsilon)L$-length-bounded k-splittable s-t-flow (Lemma 3.8). It remains to prove the performance guarantee $(1-\epsilon)/2$ for the value of the computed flow. This follows from Lemma 3.8 and the following result.

Lemma 3.9. The value of a maximum L-length-bounded k-splittable s-t-flow is at most twice as large as the value of a maximum L-length-bounded

uniform exactly-k-splittable s-t-flow.

The proof of this result is identical to the proof of [11, Theorem 12] and therefore omitted.

Since the problem solved in Lemma 3.7 is a relaxation of the maximum L-length-bounded uniform exactly-k-splittable s-t-flow problem, the value of the flow computed in the first step of our algorithm is at least half as large as the optimum. The second step decreases the flow value by another factor $(1 - \epsilon)$. Thus, our main result stated in Theorem 3.6 follows.

A Note on the Complexity of Length-Bounded k-Splittable Flows

To emphasize that the maximum length-bounded k-splittable s-t-flow problem is indeed harder than the standard maximum length-bounded s-t-flow problem, we want to point out that it is possible to find a maximum length-bounded s-t-flow in polynomial time if all arc lengths are equal to 1 (see, e.g., [9]). It is also shown in [9] that it is NP-complete to decide whether a digraph has a given number of length-bounded arc-disjoint s-t-paths with respect to unit arc lengths. This implies the following result.

Proposition 3.10. Even in networks with unit arc lengths it is NP-complete to decide whether there exists a length-bounded k-splittable s-t-flow of given value.

Proof. It is easy to see that the problem is in NP.

To show that the problem is NP-complete, we use a reduction of the NP-complete length-bounded arc-disjoint s-t-paths problem in networks with unit arc lengths. To decide whether a digraph has a given number M of length-bounded arc-disjoint s-t-paths with respect to unit arc lengths one can check if there exists a length-bounded M-splittable s-t-flow of value M in the network based upon this digraph with unit capacities. Since these two questions always have the same answer, it is already in networks with unit arc lengths NP-complete to decide whether a length-bounded k-splittable s-t-flow of given value exists. □

The foregoing proof immediately shows that, in general, the maximum length-bounded k-splittable s-t-flow problem is NP-hard.

We conclude this section by presenting a challenging open problem. Given a network with capacities and lengths on the arcs, a single source node s, and k sink nodes $t_1, \ldots, t_k$ with demand values $d_1, \ldots, d_k$, it is NP-hard to find an unsplittable flow that sends d_i units of flow from s to t_i along a single path of length at most L, for $i = 1, \ldots, k$. It is an open problem to find a

bicriteria approximation algorithm which sends a constant fraction of each demand d_i along a single s-t_i-path of length $O(L)$.

3.5 Dynamic k-Splittable Flows

In this section we again consider only the single commodity case of k-splittable flows. Using the results from the previous section and ideas of Fleischer and Skutella [31], we construct an approximation algorithm for the quickest k-splittable s-t-flow problem.

3.5.1 Problem Specification

We are given a network consisting of a digraph $D = (V, A)$, specified vertices $s, t \in V$, a capacity function $u : A \to \mathbb{R}^+$, transit times $\tau : A \to \mathbb{R}_0^+$, and a prescribed demand value d. For any path P in D and $a = (v, w) \in P$, we define P^a as the set of arcs that are traversed in P to reach v. Further, we define $\tau(A') := \sum_{a \in A'} \tau(a)$, for all $A' \subseteq A$.

A dynamic s-t-flow f in the considered network is called *k-splittable* if there are k (not necessarily different) s-t-paths $P_1, \ldots, P_k$ with corresponding flow values $f_i(\theta) \geq 0$, for all $i \in \{1, \ldots, k\}$, $\theta \in \mathbb{R}_0^+$, such that[3]

$$f(a, \theta) = \sum_{\substack{i=1,\ldots,k: \\ a \in P_i}} f_i(\theta - \tau(P_i^a)) \qquad \text{for all } a \in A,\ \theta \in \mathbb{R}_0^+.$$

In other words, $f_i(\theta)$ defines the amount of flow that enters path P_i from s at time θ. Note that a dynamic k-splittable s-t-flow is also completely specified by a set of k paths with corresponding flow values for each point in time.

The quickest k-splittable s-t-flow problem asks for a dynamic k-splittable s-t-flow with minimum time horizon T and value d.

For a dynamic k-splittable s-t-flow f with time horizon T, it holds that $f_i(\theta) = 0$ for all $\theta \geq T - \tau(P_i)$, $i \in \{1, \ldots, k\}$. The flow value of f is given by

$$|f| = \sum_{i=1}^{k} \int_0^{T-\tau(P_i)} f_i(\theta) d\theta \qquad .$$

We sometimes use the notion *static* flow in order to emphasize that some flow is not dynamic.

[3] To simplify notation, we set $f_i(\theta) := 0$, for all $i \in \{1, \ldots, k\}$, $\theta < 0$.

3.5.2 A Constant Factor Approximation

Fleischer and Skutella [31] prove that a dynamic s-t-flow of value d with time horizon T yields a static T-length-bounded s-t-flow of value d/T. (Here and throughout the rest of this section, transit times of arcs are also interpreted as lengths). This static flow can be obtained by essentially averaging the dynamic flow over time. For a dynamic k-splittable s-t-flow f with time horizon T using paths $P_1, \ldots, P_k$ with flow values $f_1(\theta), \ldots, f_k(\theta)$, for $\theta \in \mathbb{R}_0^+$, this means that we obtain a suitable T-length-bounded k-splittable s-t-flow by sending

$$f_i := \frac{1}{T} \int_0^T f_i(\theta) d\theta$$

units of flow along path P_i, for $i = 1, \ldots, k$. This flow has value

$$\sum_{i=1}^{k} f_i = \frac{1}{T} \sum_{i=1}^{k} \int_0^T f_i(\theta) d\theta = \frac{1}{T}|f| \qquad .$$

All arc capacities are obeyed, because it holds, for all $a \in A$, that

$$\begin{aligned} \sum_{\substack{i=1,\ldots,k:\\ a \in P_i}} f_i &= \frac{1}{T} \sum_{\substack{i=1,\ldots,k:\\ a \in P_i}} \int_0^T f_i(\theta) d\theta = \frac{1}{T} \int_0^T \sum_{\substack{i=1,\ldots,k:\\ a \in P_i}} f_i(\theta - \tau(P_i^a)) d\theta \\ &= \frac{1}{T} \int_0^T f(a, \theta) d\theta \leq \frac{1}{T} \int_0^T u(a) d\theta = u(a) \qquad . \end{aligned}$$

Each path P_i ($i \in \{1, \ldots, k\}$) with a positive flow value has length at most T, because f has time horizon T. Thus, we obtain the following result.

Lemma 3.11. A dynamic k-splittable s-t-flow of value d with time horizon T can be turned into a T-length-bounded k-splittable s-t-flow of value d/T in polynomial time.

On the other hand, Fleischer and Skutella [31] show that a static T-length-bounded s-t-flow of value F can be transformed into a dynamic s-t-flow of value d with time horizon $T + d/F$. The underlying transformation sends flow according to the given static flow pattern into the network for d/F time units. Then one has to wait for another T time units until the last unit of flow (traveling on a path of length, i.e., transit time, at most T) arrives at the sink. In more detail, this means for a given T-length-bounded k-splittable s-t-flow of value F with paths $P_1, \ldots, P_k$ and corresponding flow values $f_1, \ldots, f_k$ that we obtain a suitable dynamic k-splittable s-t-flow by sending $f_i(\theta) = f_i$

units of flow along path P_i, for all $i \in \{1, \ldots, k\}$, $\theta \in [0, d/F)$. The resulting flow has value

$$\sum_{i=1}^{k} \int_0^{\frac{d}{F}} f_i(\theta) d\theta = \int_0^{\frac{d}{F}} F d\theta = d \qquad .$$

All arc capacities are obeyed, because

$$\sum_{\substack{i=1,\ldots,k:\\ a \in P_i}} f_i(\theta - \tau(P_i^a)) \leq \sum_{\substack{i=1,\ldots,k:\\ a \in P_i}} f_i \leq u(a) \qquad \text{for all } a \in A.$$

Lemma 3.12. A T-length-bounded k-splittable s-t-flow of value F can be turned into a dynamic k-splittable s-t-flow of value d with time horizon $T + d/F$ in polynomial time.

From now on we use T^* to denote the time horizon of an optimal solution to the underlying quickest k-splittable s-t-flow problem with demand d. By Lemma 3.11 there exists a static T^*-length-bounded k-splittable s-t-flow of value d/T^*. If T^* was known, one could compute a $(1/\epsilon)T^*$-length-bounded k-splittable s-t-flow of value at least $(1-\epsilon)d/(2T^*)$, for any $\epsilon > 0$, (see Theorem 3.6) and transform this flow into a dynamic flow using Lemma 3.12. Not knowing T^*, we can use the following result.

Lemma 3.13. We can find $T \leq T^*$ and a $(1/\epsilon)T$-length-bounded k-splittable s-t-flow of value at least $(1-\epsilon)d/(2T)$ in polynomial time.

Proof. By the definition of T^* and Lemma 3.11 there exists a T^*-length-bounded k-splittable s-t-flow of value d/T^*. Using Lemma 3.9, it follows that there also is a T^*-length-bounded uniform exactly-k-splittable s-t-flow of value at least $d/(2T^*)$. This flow can also be interpreted as a uniform exactly-k-splittable s-t-flow of value at least $d/(2T^*)$ with average path length T^* (or instead with costs at most $d/2$ respectively).

Our algorithm works as follows: Compute a maximum uniform exactly-k-splittable s-t-flow of costs at most $d/2$. (A polynomial algorithm for this problem is stated in [11].) Let its flow value be $f \geq d/(2T^*)$ and let $T := d/(2f) \leq T^*$. According to Lemma 3.8 turn the flow into a $(1/\epsilon)T$-length-bounded k-splittable s-t-flow of value at least $(1-\epsilon)d/(2T)$. (This can be done, because the computed uniform exactly-k-splittable s-t-flow has value $f = d/(2T)$ and cost at most $d/2 = fT$.)

All computations in the algorithm can be done in polynomial time. □

Now we can use Lemma 3.12 in order to obtain a dynamic k-splittable s-t-flow of value d with time horizon $T/\epsilon + 2T/(1-\epsilon) = (1+\epsilon)T/(\epsilon - \epsilon^2)$.

Thus, we have a $(1+\epsilon)/(\epsilon-\epsilon^2)$-approximation algorithm for the quickest k-splittable flow problem. Since the term $(1+\epsilon)/(\epsilon-\epsilon^2)$ attains its minimum for $\epsilon = \sqrt{2}-1$, this yields the following result.

Theorem 3.14. There is a $\sqrt{2}/(3\sqrt{2}-4)$-approximation[4] algorithm for the quickest k-splittable s-t-flow problem.

Proof. Use the $(1+\epsilon)/(\epsilon-\epsilon^2)$-approximation algorithm derived above with $\epsilon = \sqrt{2}-1$. □

[4] $\sqrt{2}/(3\sqrt{2}-4) \approx 5.828$

Chapter 4

One-Flows

4.1 Introduction

The classical maximum s-t-flow problem has been studied from many different points of view. Numerous algorithms are known to solve the problem in polynomial time (see, e.g., [2, 93]). Ford and Fulkerson [35] proved already in the 1950s that there always exists an integral optimal solution to the maximum s-t-flow problem provided that all arc capacities are integral. This follows for example from the fact that the constraint matrix of the natural LP formulation in arc variables is totally unimodular. It is also well known that any s-t-flow can be decomposed into flow along paths and cycles. Omitting flow along cycles (which does not contribute to the flow value) yields an alternative LP formulation of the problem in path variables.

In this chapter we study a new network flow problem in which the flow on any path is bounded by 1. In other words, we add box constraints to the LP formulation of the maximum flow problem in path variables. We call the resulting problem the *maximum one-flow problem*. Our motivation for studying this problem is mainly academic but, from a more practical point of view, the problem can be motivated when we think of problems in transportation or communication networks where every single path might be unreliable. In such situations it is reasonable to diversify a commodity or information among several different paths. This can be accomplished by forbidding to send more than a fixed amount of flow along a single path.

Related Results from the Literature

To the best of our knowledge, the one-flow problem (1FP) has not been studied before. But there is some literature dealing with problems related to it. The problem to compute the number of different (simple) s-t-paths in a network is a special case of the max-1FP. (Consider the case when all arc capacities are infinite.) Valiant [97] shows that this problem is #P-complete under polynomial-time reductions. It is common to compute the number of paths in grids. Such work was, e.g., done by Lucas [73], who shows the

relationship between Pascal's triangle and the "space" grid, or Gessel [41] counting paths in the so called Young's lattice. More general results were, e.g., obtained by Bartholdi [12] and Stanley [96].

Similar to the problem of counting paths in a graph is the edge-disjoint paths problem (EDP) which is the same as the undirected integral multicommodity 1FP with unit demands when we fix all capacities in the considered network to 1. Kleinberg [61] gives an extensive overview of the EDP. Further interesting results were, e.g., obtained by Guruswami et al. [48]. They prove that the maximum EDP is hard to approximate within a factor $O(m^{1/2-\epsilon})$, for any $\epsilon > 0$, and give an $O(\sqrt{m})$-approximation algorithm. More recent results for the EDP were, e.g., found by Andrews and Zhang [3] or Chekuri and Khanna [20]. Considering the EDP as a special packing problem, there are more results by Baveja and Srinivasan [13] and Kolliopoulos and Stein [66]. Most of the results mentioned here can be generalized to the directed case. Very recently, Kolliopoulos [64] came up with an extensive survey on edge-disjoint paths.

Also for more general packing problems many results have been obtained during the last years. Among the most interesting results for the problem considered here are the ones obtained by Plotkin, Shmoys, and Tardos [86] who introduce new approximation algorithms for the problem. Together with Grigoriadis and Khachiyan [43, 44, 45, 46], they adapt techniques for solving multicommodity flow problems (see, e.g., [94, 59, 70, 40, 33]) to a general class of packing and covering problems. Other results on mixed packing and covering problems were, e.g., obtained by Young [99].

In Chapter 3 we have already considered a type of network flow problems with path capacities. Remember that we studied the k-splittable flow problem with path capacities that restricts the flow of any commodity to at most k paths whose flow values may not exceed given bounds.

Chen [22, 23], Chen and Hung [21], and Rosen, Sun, and Xue [91] consider the problem to find k shortest paths from s to t in a network with arc capacities. But their results are rather far away from the problem considered here, because the only role of the arc capacities in their research is that the transmission time of flow through an arc is the arc's length plus the number of paths using this arc divided by its capacity. Nevertheless, the standard k shortest paths problem (for a single source and a single sink) is of great interest for the problem considered here. To solve the Lagrange relaxation of the max-1FP that is obtained by penalizing the violation of arc capacities in the objective function, we make use of a result by Lawler [69] who shows how to compute k shortest (simple) paths in $O(kn^3)$ time. This improves the runtime of an algorithm by Yen [98] by a factor n. Eppstein [29] considers the problem to compute k shortest s-t-paths allowing cycles and presents an

algorithm running in $O(m + n \log n + k)$ time. Fox [38] presents a method for the problem allowing cycles that is based on Dijkstra's algorithm [25] and runs in $O(m + kn \log n)$ time. Using improvements in Dijkstra's algorithm the runtime in [69] can be decreased to $O(kn(m + n \log n))$. Dreyfus [27] and Eppstein [29] also consider the problem to compute k shortest paths (allowing cycles) from a given source to each other vertex. The running times they obtain are $O(kn^2)$ and $O(m + n \log n + kn)$ respectively. Very recently, Roditty [90] presented an approximation algorithm that computes k simple paths from a source s to a sink t such that the length of the ith path is at most 3/2 times the length of the ith shortest simple s-t-path. This algorithm runs in $O(k\sqrt{n}(m + n \log n))$ time.

Organization of this Chapter

We start with a detailed definition of the considered problem in Section 4.2.

In Section 4.3 we establish a close relation between two interesting new combinatorial optimization problems and the special case of the max-1FP on networks consisting of a chain of parallel arcs. The first problem is to cover the edges of a complete graph by cuts of bounded size where the size of a cut is the cardinality of the smaller of the two vertex subsets. The second problem is a packing problem: Consider a set where each element has a given integral weight and find a prespecified number of different subsets such that the number of subsets containing an element is bounded by the element's weight. The two problems are equivalent and, maybe surprisingly, strongly NP-hard. This also yields the strong NP-hardness of the integral max-1FP on chains of parallel arcs. Moreover, we show that already on a chain of parallel arcs of length 3 the max-1FP has an integrality gap.

In Section 4.4 we present complexity results for different versions of the 1FP. As mentioned above, the problem of computing the number of different simple s-t-paths in a network is #P-complete and a special case of the max-1FP. It therefore follows immediately that computing the maximum one-flow value is NP-hard, for the fractional as well as the integral one-flow problem. But even worse, the integral max-1FP is APX-hard even in networks where the number of s-t-paths is polynomially bounded in the size of the network. These complexity results are somehow surprising for a problem that is obtained by "only" adding box constraints to the classical maximum flow problem. We also present an instance where each arc carries an integral amount of flow in a maximum one-flow but no maximum one-flow is integral.

In Section 4.5 we show that the approach of Plotkin et al. [86] to approximate general packing problems yields an FPTAS for the fractional max-1FP.

The core of the algorithm consists of iteratively solving k shortest paths problems on the given network with varying arc length. In Section 4.6 we derive several approximation algorithms for the integral max-1FP. Our main result is a randomized approximation algorithm with performance ratio $O(\log m)$.

Finally, in Section 4.7 we study multicommodity versions of the one-flow problem. We show that the FPTAS from Section 4.5 can be generalized to the fractional multicommodity one-flow problem. For the integral maximum multicommodity one-flow problem we present a randomized $O(\sqrt{m})$-approximation algorithm and show that, unless P=NP, no better approximation is possible. Moreover, we present an $O(\log m / \log\log m)$-approximation algorithm for the problem to find an integral multicommodity one-flow with minimum congestion.

4.2 Problem Definition and Notation

An instance of the maximum one-flow problem provides a digraph $D = (V, A)$ with arc capacities $u : A \to \mathbb{R}^+$ and two specified nodes $s,\ t \in V$. If not stated otherwise, $m := |A|$ denotes the number of arcs in the network. Let $\mathcal{P}$ be the set of simple s-t-paths in D. Then the *maximum one-flow problem (max-1FP)* can be formulated as follows, where the path variable x_P denotes the amount of flow sent along path $P \in \mathcal{P}$:

$$\begin{aligned} \max \quad & \sum_{P \in \mathcal{P}} x_P & & \\ \text{s.t.} \quad & \sum_{\substack{P \in \mathcal{P}: \\ a \in P}} x_P \leq u(a) & \forall\ a \in A & \qquad (4.1) \\ & 0 \leq x_P \leq 1 & \forall\ P \in \mathcal{P} \quad . & \qquad (4.2) \end{aligned}$$

Notice that omitting the constraints $x_P \leq 1$ yields the classical maximum s-t-flow problem. An s-t-flow which fulfills (4.1) and (4.2) is called a *one-flow*. In an *integral one-flow* each s-t-path sends either 0 or 1 unit of flow. To emphasize that a certain one-flow is not necessarily integral, we call it *fractional one-flow* in some cases. If the meaning is clear from the context, we sometimes use the notion one-flow also for a flow which only obeys 4.2. In order to make clear then that a one-flow obeys arc capacities we call it *feasible*.

Note that in general the encoding size of a maximum one-flow is not polynomial in the input size of the problem since one might have to send flow along exponentially many s-t-paths. Therefore, the best one can expect in terms of complexity are algorithms with running time polynomially bounded in the input plus output size.

We also consider the dual of the max-1FP which is given as follows:

$$\begin{aligned} \min \ & \sum_{a\in A} u(a)\lambda_a + \sum_{P\in\mathcal{P}} y_P & & \\ \text{s.t.} \ & y_P + \sum_{a\in P} \lambda_a \geq 1 & & \forall\ P \in \mathcal{P} \\ & y_P,\ \lambda_a \geq 0 & & \forall\ a \in A,\ P \in \mathcal{P} \qquad . \end{aligned}$$

The integer version of the dual can be interpreted as a special minimum cut problem, where each s-t-path must be destroyed and this can be done by deleting either a single arc on the path or the path itself. The deletion of an arc a is in general more expensive than that of a whole path ($u(a)$ instead of 1), but can also destroy more than one path at the same time.

The dual separation problem of the classical maximum s-t-flow problem is a shortest path problem. It is not difficult to observe that the dual separation problem of the max-1FP can be solved by computing the k shortest s-t-paths with respect to the dual arc lengths λ_a, where k is the number of paths $P \in \mathcal{P}$ with $y_P > 0$ plus 1. The k shortest paths problem also plays an important role in solving the Lagrange relaxation of the max-1FP that is obtained by penalizing the violation of capacity constraints (4.1) in the objective function. We discuss this issue in more detail later on.

In the remainder of this section we assume that the minimal arc capacity $u_{min} := \min_{a\in A} u(a)$ is at least 1.

4.3 A Special Case of the Max-1FP and Related Problems

We start with the study of the max-1FP on a restricted class of networks that are given by chains of parallel arcs. In order to obtain a better understanding of the max-1FP on this particular class of networks we consider two equivalent combinatorial optimization problems, one of which is a covering and the other a packing problem. Although these two problems are easy to formulate and seem quite natural, they have not appeared in the literature before to the best of our knowledge.

We consider networks which consist of $n+1$ vertices $v_0, v_1, \ldots, v_n$ and $2n$ arcs ($n \in \mathbb{N}$) such that there are two parallel arcs from v_{i-1} to v_i, for $i = 1, \ldots, n$. Vertex v_0 is the source and v_n is the sink. We call one arc of each pair of parallel arcs the *upper* and the other one the *lower arc*. All lower arcs have infinite capacity. The capacity of the ith upper arc is $c_i \in \mathbb{N}$, for $i = 1, \ldots, n$. We call such a capacitated network a *chain of parallel arcs*; see Figure 4.1 for an example.

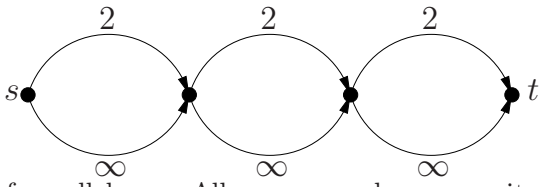

Figure 4.1: A chain of parallel arcs. All upper arcs have capacity 2 while the lower arcs have infinite capacity.

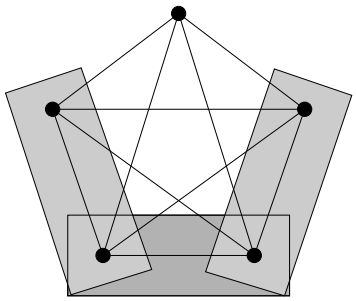

Figure 4.2: A solution to the bounded cut cover problem for $q = 5$, $k = 3$ and $c_1 = c_2 = c_3 = 2$.

An integral one-flow is given by a set of s-t-paths that are pairwise different. Notice that two s-t-paths in the considered network are different if and only if there is a pair of arcs where one path uses the upper arc and the other path uses the lower arc. In particular, the ith pair of arcs can distinguish a subset of at most c_i paths from all other paths. This motivates the following problem.

Bounded Cut Cover Problem

GIVEN: k numbers $c_1, \ldots, c_k \in \mathbb{N}$ and a number $q \in \mathbb{N}$.

TASK: Find k subsets $M_1, \ldots, M_k \subseteq \{1, \ldots, q\}$ with $|M_i| \leq c_i$, for $i = 1, \ldots, k$, such that for any pair $j, \ell \in \{1, \ldots, q\}$ with $j \neq \ell$ there is some $i \in \{1, \ldots, k\}$ with $|M_i \cap \{j, \ell\}| = 1$; or decide that no such family of subsets exists.

The name that we choose for this problem stems from the following graph-theoretic interpretation: Consider a complete undirected graph with vertex set $\{1, \ldots, q\}$. The question is whether the edges of the complete graph can be covered by k cuts where, for $i = 1, \ldots, k$, the ith cut partitions the vertex set into two subsets the smaller of which has cardinality at most c_i. In Figure 4.2 we give an example.

Observation 4.1. The bounded cut cover problem has a solution if and only if there exists an integral one-flow of value q in a chain of k parallel arcs where the capacities of the upper arcs are $c_1, \ldots, c_k$.

The solution to the instance of the bounded cut cover problem depicted in Figure 4.2 represents a one-flow of value 5 in the chain of parallel arcs given in Figure 4.1. Next we consider the following fractional relaxation of the bounded cut cover problem.

Fractional Bounded Cut Cover Problem

GIVEN: k numbers $c_1, \ldots, c_k \in \mathbb{N}$ and a number $q \in \mathbb{R}^+$.

TASK: For some $r \geq q$, find weights $x_1, \ldots, x_r \in [0,1]$ with $\sum_{j=1}^{r} x_j = q$ and determine k subsets $M_1, \ldots, M_k \subseteq \{1, \ldots, r\}$ with $\sum_{j \in M_i} x_j \leq c_i$, for $i = 1, \ldots, k$, such that for any pair $j, \ell \in \{1, \ldots, r\}$ with $j \neq \ell$ there is some $i \in \{1, \ldots, k\}$ with $|M_i \cap \{j, \ell\}| = 1$; or decide that this is not possible.

There is again a graph-theoretic interpretation of the problem. The task is to find a complete graph with weights on the vertices such that the weight x_i of every vertex i is between 0 and 1 and the weights sum up to q. Moreover, the edges of the complete graph must be covered by k cuts such that the ith cut partitions the vertex set into two subsets the lighter of which has total weight at most c_i. Associating the weighted nodes of the complete graph with s-t-paths of corresponding flow value yields the following observation.

Observation 4.2. The fractional bounded cut cover problem has a solution if and only if there exists a (fractional) one-flow of value q in a chain of k parallel arcs where the capacities of the upper arcs are $c_1, \ldots, c_k$.

It is a natural question to ask whether the fractional bounded cut cover problem allows for larger values of q with feasible solutions than the non-fractional version. By Observations 4.1 and 4.2, this is equivalent to the question whether for chains of parallel arcs there always exists a maximum one-flow that is integral. In the fractional bounded cut cover problem the price of the additional degree of freedom given by the possibility to assign fractional weights to the nodes is an increase in the number of nodes (since the node weights still have to sum up to q). On the one hand, a larger number of nodes makes it more difficult to cover all edges of the complete graph. On the other hand, fractional weights on the vertices allow for more balanced cuts that contain more edges. We show below that there exist instances with a larger feasible value of q in the fractional version of the problem than in the integral version. Before we discuss this issue in more detail, we present another equivalent packing problem.

In a chain of parallel arcs, every s-t-path is uniquely determined by the subset of upper arcs contained in the path. Therefore, computing an integral one-flow of value q corresponds to finding a family of q pairwise distinct

subsets of $\{1,\dots,k\}$ such that $i \in \{1,\dots,k\}$ is contained in at most c_i of these subsets.

Capacitated Set Packing Problem

GIVEN: k numbers $c_1,\dots,c_k \in \mathbb{N}$ and a number $q \in \mathbb{N}$.

TASK: Find q pairwise distinct subsets of $\{1,\dots,k\}$ such that element $i \in \{1,\dots,k\}$ is contained in at most c_i of these subsets, for $i = 1,\dots,k$; or decide that no such family of subsets exists.

We also consider the following fractional relaxation of the capacitated set packing problem.

Fractional Capacitated Set Packing Problem

GIVEN: k numbers $c_1,\dots,c_k \in \mathbb{N}$ and a number $q \in \mathbb{R}^+$.

TASK: For some $r \geq q$, find pairwise distinct subsets $N_1,\dots,N_r$ of $\{1,\dots,k\}$ with weights $x_1,\dots,x_r \in [0,1]$ such that $\sum_{j=1}^{r} x_j = q$ and $\sum_{j:i\in N_j} x_j \leq c_i$, for $i = 1,\dots,k$; or decide that this is not possible.

Observation 4.3. The (fractional) capacitated set packing problem has a solution if and only if there exists an integral (fractional) one-flow of value q in a chain of k parallel arcs where the capacities of the upper arcs are $c_1,\dots,c_k$. In particular, the (fractional) capacitated set packing problem is equivalent to the (fractional) bounded cut cover problem.

The following instance shows that the fractional capacitated set packing problem in general allows for strictly larger values of q with feasible solutions than the non-fractional version. Due to Observation 4.3, the same holds for the bounded cut cover problem. Let $k = 3$ and $c_1 = c_2 = c_3 = 2$. It is not difficult to check that $q = 5$ is the largest value of q with a feasible solution to the non-fractional version of the problem: Choose for example the subsets $\emptyset$, $\{1\}$, $\{2\}$, $\{3\}$, and $\{1,2,3\}$. But there is a solution to the fractional version of the problem with $q = 5.5$: Choose subsets $\emptyset$, $\{1\}$, $\{2\}$, and $\{3\}$ all with weight 1. In addition choose subsets $\{1,2\}$, $\{1,3\}$, and $\{2,3\}$ all with weight $1/2$. The corresponding instance of the max-1FP is depicted in Figure 4.1.

Observation 4.4. The instance of the max-1FP depicted in Figure 4.1 has an integrality gap of $11/10$.

The instance depicted in Figure 4.1 also shows that in general there is no integral optimal solution to the dual problem of the max-1FP. An optimal

dual solution destroys the path using all lower arcs and half of each path that uses exactly one upper arc; further, one half of each upper arc is deleted.

In Section 4.4 we present an instance of the max-1FP proving another interesting result concerning the integrality of the max-1FP. This is given in the following proposition. We do not prove the result here, because the considered instance is slightly more complex than a chain of parallel arcs.

Proposition 4.5. The existence of a maximum one-flow where the flow value on each arc is integral does in general not imply the existence of an integral path decomposition where each path carries at most one unit of flow.

We now return to the capacitated set packing problem. In contrast to the classical set packing problem and many similar problems known from the literature, the capacitated set packing problem allows to choose arbitrary subsets, i.e., they do not have to belong to a given family of subsets. This might make the problem seem to be easier. But we can prove the following somehow surprising theorem.

Theorem 4.6. The capacitated set packing problem is strongly NP-hard. If q is polynomially bounded in k, the problem is even strongly NP-complete[1].

Proof. In the following we use the notion r-subset to describe a subset of cardinality $r \in \mathbb{N}$ of a given ground set. To prove the NP-hardness of the capacitated set packing problem, we use a reduction of the strongly NP-complete 3-PARTITION problem.

3-PARTITION

GIVEN: An integer $B \in \mathbb{N}$ and $k = 3\ell$ numbers $s_1, \ldots, s_k \in \mathbb{N} \cap (B/4, B/2)$, for some $\ell \in \mathbb{N}$, such that $\sum_{i=1}^{k} s_i = \ell B$.

TASK: Find a partition $A_1, \ldots, A_\ell$ of $\{1, \ldots, k\}$ such that $\sum_{i \in A_j} s_i = B$ for all $j = 1, \ldots, \ell$; or decide that no such partition exists. (Note that $|A_j| = 3$, for all $j = 1, \ldots, \ell$, since $s_i \in (B/4, B/2)$ for $i = 1, \ldots, k$.)

Consider any instance of 3-PARTITION. We construct an instance of the capacitated set packing problem as follows. Let $B_3^<$ be the set of 3-subsets of $\{1, \ldots, k\}$ such that for each $\mathcal{M} \in B_3^<$ it holds that $\sum_{i \in \mathcal{M}} s_i < B$. For $i = 1, \ldots, k$, let $B_3^<(i)$ be the sets in $B_3^<$ that contain i. Then the capacity of i is given as $c_i := |B_3^<(i)| + k + 1$. The integer q is set to $q := |B_3^<| + \binom{k}{2} + k + \ell + 1$.

[1] If we use the term *NP-complete*, we always allude to the corresponding decision problem.

Note that all numbers here are computable in time polynomial in the size of the 3-PARTITION instance, since there are only $\binom{k}{3} = O(k^3)$ different 3-subsets of $\{1, \ldots, k\}$. Further, all numbers c_i and q are polynomial in k.

Now we show that a feasible partition for the instance of 3-PARTITION exists if and only if there is a feasible family of subsets for the instance of the capacitated set packing problem.

Let us start with the forward implication and assume that for the considered instance of 3-PARTITION there is a feasible partition. Then we can choose q subsets of $\{1, \ldots, k\}$ as follows. First choose the empty set, all 1-subsets, all 2-subsets, and all sets in $B_3^<$. Further, choose the sets $A_1, \ldots, A_\ell$ of a 3-Partition. Then we have a total number of $1 + k + \binom{k}{2} + |B_3^<| + \ell = q$ subsets. Any $i \in \{1, \ldots, k\}$ appears in $1 + (k-1) + |B_3^<(i)| + 1 = c_i$ of the chosen subsets. Thus, we have a feasible family of subsets for the considered instance of the capacitated set packing problem.

Now let us turn to the backward implication and assume that for the constructed instance of the capacitated set packing problem there exists a feasible family of subsets. Then there exists a family S of q pairwise distinct subsets of $\{1, \ldots, k\}$ such that each $i \in \{1, \ldots, k\}$ appears in at most c_i of the subsets in S. We show by contradiction that the empty set as well as all 1-subsets of $\{1, \ldots, k\}$, all 2-subsets of $\{1, \ldots, k\}$, and all sets in $B_3^<$ must be in S.

1. $\emptyset \in S$
   Assume that $\emptyset \notin S$. Since we have q subsets in S, it holds that

   $$\sum_{\mathcal{M} \in S} |\mathcal{M}| \geq k + 2\binom{k}{2} + 3(|B_3^<| + \ell + 1) \quad ,$$

   because the first sum is smallest, if we use all k of the 1-subsets of $\{1, \ldots, k\}$, all $\binom{k}{2}$ of the 2-subsets of $\{1, \ldots, k\}$, and no subsets of $\{1, \ldots, k\}$ with more than three elements. From the group of 3-subsets we have to pick at least $|B_3^<| + \ell + 1$ to end up with a total number of q subsets.

   The total capacity of elements in $\{1, \ldots, k\}$ is $\sum_{i=1}^{k} c_i = 3|B_3^<| + 2\binom{k}{2} + 2k$ which is less than $\sum_{\mathcal{M} \in S} |\mathcal{M}|$—a contradiction.

2. All 1-subsets have to be in S.
   Assume that there is a 1-subset which is not in S. By an argument analogous to the one above, it follows that

   $$\sum_{\mathcal{M} \in S} |\mathcal{M}| \geq (k-1) + 2\binom{k}{2} + 3(|B_3^<| + \ell + 1) \quad ,$$

because the first sum is smallest, if we use the empty set, which does not appear in the latter sum, the $k-1$ remaining 1-subsets of $\{1,\dots,k\}$, all $\binom{k}{2}$ of the 2-subsets of $\{1,\dots,k\}$, and no subsets of $\{1,\dots,k\}$ with more than three elements.

Again the sum of elements used in total is larger than the total capacity of elements in $\{1,\dots,k\}$.

3. All 2-subsets have to be in S.
   Assume that there is a 2-subset which is not in S. Here it holds that
   $$\sum_{\mathcal{M}\in S} |\mathcal{M}| \geq k + 2(\binom{k}{2} - 1) + 3(|B_3^<| + \ell + 1) \quad ,$$
   which is again larger than the total capacity of elements in $\{1,\dots,k\}$.

4. All sets in $B_3^<$ have to be in S.
   Assume that there are $r \geq 1$ subsets in $B_3^< \setminus S$. To obtain a total number of q subsets in S, the selection must contain $r + \ell$ subsets of $\{1,\dots,k\}$ whose respective sum of elements is greater or equal to B. Let the family of these subsets be denoted by T.

   Since we know that all 1-subsets of $\{1,\dots,k\}$ and all 2-subsets of $\{1,\dots,k\}$ are in S, an element $i \in \{1,\dots,k\}$ may be used by at most $c_i - 1 - (k-1) = |B_3^<(i)| + 1$ other subsets. The total size of elements that may be used is therefore at most
   $$\sum_{i=1}^{k} s_i(|B_3^<(i)| + 1) = \ell B + \sum_{i=1}^{k} s_i |B_3^<(i)| = \ell B + \sum_{\mathcal{M}\in B_3^<} \sum_{i\in\mathcal{M}} s_i \quad .$$

   The total size of elements that is needed by T and the $|B_3^<| - r$ sets from $B_3^< \cap S$ is given by
   $$\sum_{\mathcal{M}\in T} \sum_{i\in\mathcal{M}} s_i + \sum_{\mathcal{M}\in B_3^<\cap S} \sum_{i\in\mathcal{M}} s_i \quad .$$

   It follows that
   $$\sum_{\mathcal{M}\in T} \sum_{i\in\mathcal{M}} s_i + \sum_{\mathcal{M}\in B_3^<\cap S} \sum_{i\in\mathcal{M}} s_i \leq \ell B + \sum_{\mathcal{M}\in B_3^<} \sum_{i\in\mathcal{M}} s_i \quad .$$

   This implies
   $$(\ell + r)B \leq \sum_{\mathcal{M}\in T} \sum_{i\in\mathcal{M}} s_i \leq \ell B + \sum_{\mathcal{M}\in B_3^<\setminus S} \sum_{i\in\mathcal{M}} s_i < \ell B + rB \quad ,$$
   which yields a contradiction.

Now we know that the empty set, all 1-subsets of $\{1,\dots,k\}$, all 2-subsets of $\{1,\dots,k\}$, and all sets in $B_3^<$ must be in S. Then, for each $i \in \{1,\dots,k\}$, we have a remaining capacity of $c_i - 1 - (k-1) - |B_3^<(i)| = 1$. Since S contains q subsets, it uses $q - 1 - k - \binom{k}{2} - |B_3^<| = \ell$ subsets in addition to the ones mentioned. These subsets must be 3-subsets, since we have only 3ℓ elements and all subsets with less than three elements are already used. Thus, they form a partition of $\{1,\dots,k\}$. Further, the respective sum of sizes of the elements in the subsets must be greater or equal to B, since all sets in $B_3^<$ are also already used. But now it follows immediately that for each of these subsets its total sum equals B, because $\sum_{i=1}^{k} s_i = \ell B$ and if any subset had a total size of more than B, another subset would have to have one of less than B.

These arguments show that for the considered instance of 3-PARTITION a feasible partition exists if there is a feasible family of subsets for the constructed instance of the capacitated set packing problem.

As already mentioned, in this reduction all numbers (in particular q) are polynomial in k. That proves that the capacitated set packing problem is strongly NP-hard, even if q is polynomially bounded in k. For any instance to this restricted problem, it can be checked in time polynomial in its encoding size whether a given family of subsets is a solution for this instance. This completes our proof. □

As an immediate consequence of Theorem 4.6 we can state the following hardness results.

Theorem 4.7. 1. The bounded cut cover problem is strongly NP-hard. If q is polynomially bounded in k, then the problem is even strongly NP-complete.

2. The problem of finding an integral one-flow of maximum value for a chain of parallel arcs is strongly NP-hard, even if the maximum flow value is polynomially bounded in the size of the network (i.e., number of vertices).

It follows that, in contrast to the problem to count the number of s-t-paths in a digraph, the integral max-1FP is already strongly NP-hard in acyclic networks.

Note that in the integral one-flow problem the flow value bounds the number of paths that are used to route a flow. Thus, we can derive the following from part 2 of Theorem 4.7.

Corollary 4.8. Unless P = NP, there is no algorithm for the integral max-1FP in chains of parallel arcs whose runtime is pseudo-polynomial in input plus output size.

Since it is already NP-complete to decide if for a given instance of 3-PARTITION a feasible partition exists, the corresponding capacitated set packing decision problem is also NP-complete. (This follows from the proof of Theorem 4.7.) It follows immediately that it is even strongly NP-hard to compute only the value of a maximum integral one-flow in acyclic networks.

We conclude this section with an open problem. The instance of the max-1FP depicted in Figure 4.1 has an additive integrality gap of 1/2. Do there exist instances with additive integrality gap 1 or larger? This question is also interesting in view of the bounded cut cover problem and the capacitated set packing problem.

4.4 Complexity Results

In contrast to the classical maximum s-t-flow problem, not all instances of the max-1FP admit an integral optimum (see Section 4.3). We obtain different hardness results for the fractional and the integral version of the max-1FP that are discussed in Section 4.4.1. In Section 4.4.2 we present an interesting result concerning the integrality of one-flows. This has already been mentioned in Proposition 4.5.

4.4.1 NP-Hardness Results

As already mentioned, it follows from the #P-completeness of the problem to compute the number of different simple s-t-paths in a network that the general max-1FP is NP-hard. (Consider the case that all arc capacities are infinite.)

Theorem 4.9. The general max-1FP is NP-hard.

It also follows immediately from this reduction that the integral max-1FP is strongly NP-hard as well. But for the integral max-1FP we can even prove APX-hardness using a reduction of MAX 3SAT.

Theorem 4.10. The integral max-1FP is APX-hard. This holds, even if we restrict to networks in which the number of s-t-paths is polynomially bounded in the size of the network. (Thus, there cannot exist a PTAS with a runtime polynomial in input plus output size, unless P = NP.)

Proof. We use a reduction of MAX 3SAT-3—a special kind of MAX 3SAT. In contrast to the usual 3SAT problem, MAX 3SAT-3 has the additional constraint on the input that each variable appears in at most three clauses. MAX 3SAT-3 is still APX-hard [84].

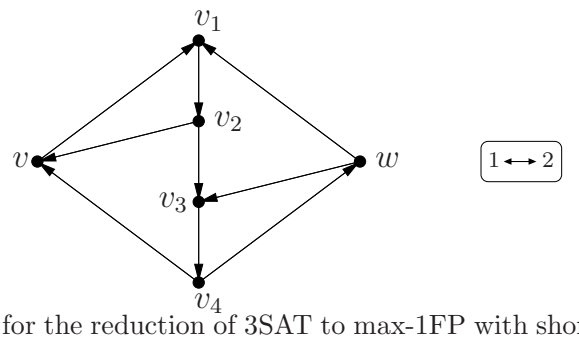

Figure 4.3: Gadget for the reduction of 3SAT to max-1FP with shortcut notation on the right.

Let the considered 3SAT instance consist of clauses C_i, for $i = 1, \ldots, m$. We say that a variable *appears purely* if it turns up unnegated. We assume that no variable appears only in exactly one clause or appears either only purely or only negated. This can be assumed without loss of generality, because the values of such variables can be set in a preprocessing step.

The corresponding instance of the max-1FP is constructed as follows. First insert a source s, a sink t, and nodes c_i with arcs (s, c_i) for all clauses C_i $(i = 1, \ldots, m)$. We need to distinguish three types of variables:

1. Variables that appear exactly twice, once purely and once negated.

2. Variables that appear purely once and negated twice.

3. Variables that appear purely twice and negated once.

For each variable of the first type we insert a node v, an arc (v, t) and arcs (c_i, v) for both $i \in \{1, \ldots, m\}$ for which C_i contains the considered variable. In the following let V_1 be the set of nodes in the constructed graph resulting from these variables.

For each variable x of the second type we insert the following gadget (see also Figure 4.3). The gadget consists of six nodes v, v_1, v_2, v_3, v_4, and w and the arcs (v_1, v_2), (v_2, v_3), (v_3, v_4), (v, v_1), (v_2, v), (v_4, v), (w, v_1), (w, v_3), and (v_4, w). It is connected to s and t by the arcs (w, t) and (v, t), an arc (c_i, v) for $i \in \{1, \ldots, m\}$ for which C_i contains x, and arcs (c_j, w) for both $j \in \{1, \ldots, m\}$ for which C_j contains $\neg x$. The described gadget is also denoted by the shortcut given in Figure 4.3. The important property of this gadget is that—given arc capacities equal to 1—it can either route one unit of flow from v to w or one or two units of flow from w to v, but not both.

For each variable x of the third type we insert the same gadget as above, but connect it to the rest of the network the other way around. For the clause C_i which contains $\neg x$ we insert an arc (c_i, v), for the clauses C_j and

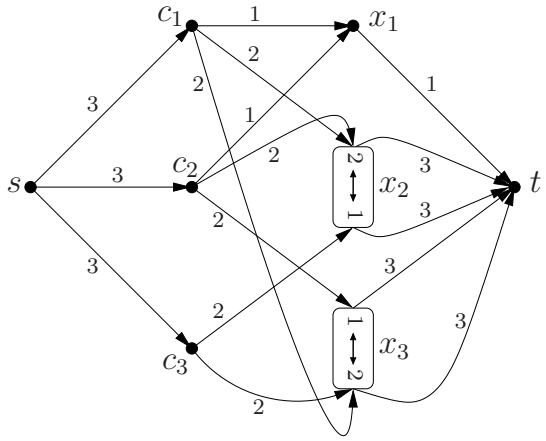

Figure 4.4: Reduction network for the SAT-instance $(\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3)$. All arcs entering or leaving a gadget are incident to the corresponding v or w respectively.

C_k containing x we add arcs (c_j, w) and (c_k, w). Additionally, we add again arcs (w, t) and (v, t).

After the specification of nodes and arcs we still need to define arc capacities. These are given as follows: Each arc in a gadget and all arcs $(v, t), (c_i, v)$ resulting from variables of the first type have capacity 1. All other arcs $(c_i, v), (c_i, w)$ have capacity 2 and all other arcs $(v, t), (w, t)$ capacity 3. The capacities of all arcs (s, c_i) is something between 1 and 4, depending on the capacities of arcs leaving c_i. In order to compute the capacity of arc (s, c_i), take the number of arcs of capacity 2 leaving c_i and add 1. Please see Figure 4.4 for an example of the reduction.

Now let $D = (V, A)$ with capacities $u : A \to \mathbb{R}^+$ be the reduction network for the considered instance $C = \bigwedge_{i=1}^{m} C_i$ of MAX 3SAT-3. Let C^* be the set of clauses that are satisfied in an optimal solution to this instance. Further, let L_3 be the set of variables that appear exactly three times in C.

Claim. The value of a maximum integral one-flow in D with capacity function u is exactly $3|L_3| + |C^*|$.

We first show that there exists an integral one-flow in D having value $3|L_3| + |C^*|$. Assume that we have an assignment for the variables in C that satisfies the clauses in C^*. Then send one unit of flow along each s-t-path not using arcs in a gadget or a node in V_1. This gives us $3|L_3|$ units of flow, because for each variable x in L_3 there are exactly three s-t-paths using v and w in the gadget of x, but no arc of it. Due to our construction of the gadgets we can additionally send one unit of flow for each satisfied clause through the gadget (or the node from V_1 respectively) which belongs to one of the variables by

which the clause is satisfied. Since this gives us another $|C^*|$ units of flow, it follows that there exists an integral one-flow of value $3|L_3| + |C^*|$.

It remains to show that there cannot be an integral one-flow of value greater than $3|L_3|+|C^*|$. Assume that we have an integral one-flow f of value $3|L_3|+k$, for some $k \in \mathbb{N}$. We show that then there must exist an assignment for the variables in C such that at least k clauses of C are satisfied. We know that f can send at most $3|L_3|$ units of flow that traverse neither a gadget nor a node from V_1. This implies immediately that at least k units of flow must be routed through gadgets or V_1. It remains to show that there are at least k clauses for which a unit of flow is routed from its corresponding node to t by traversing a gadget or a node in V_1. Then the flow implies an assignment for the variables in C. A variable is set TRUE if the inflow in the corresponding gadget (or in the node from V_1) comes from a clause in which it appears purely and FALSE otherwise. The number of satisfied clauses is then k.

If f routes at most one unit of flow for any clause from its corresponding node to t by traversing a gadget or a node in V_1, we are done. Thus, let us assume that there is a clause for which f routes $\ell \geq 2$ units of flow in such a way. Due to the capacities of the arcs (s, c_i), for $i = 1, \ldots, m$, it follows that at most $3|L_3| - (\ell - 1)$ units of flow can be sent without using gadgets or V_1, which means for the remaining clauses that they must send $k - 1$ units of flow through gadgets or V_1. By induction it follows that there must be at least k clauses for which at least one unit of flow is routed through a gadget or V_1.

Thus, we showed that the value of a maximum integral one-flow in D is at least $3|L_3| + |C^*|$ and that for any integral one-flow of value $3|L_3| + k$ we can find an assignment for the variables in C such that k clauses in C are satisfied. This implies that $3|L_3| + |C^*|$ is the value of any maximum integral one-flow and proves the claim.

Since the assignment of values to the variables in C can immediately be taken from the computed one-flow (see above), it only remains to prove that the number of paths in D is polynomially bounded in the size of D. Let X be the number of variables in C, then this follows immediately from the fact that $10X$ yields an upper bound for the number of paths in D. We can bound the number of paths in D by $10X$, because at each structure representing a clause (either a gadget or a node in V_1) we have at most three entering arcs. An arc entering a node in V_1 results in only one path to t. Arcs entering a node v of a gadget result in two paths to t—one through the gadget and a "direct" one. And arcs entering a node w of a gadget result in four paths to t—three through the gadget and a "direct" one. Since there is only one possibility to reach either of the considered arcs from s and three arcs (only

two of the same type) enter a gadget, we have at most ten paths per variable. With this insight the proof of Theorem 4.10 is complete. □

It follows from the strong NP-hardness of the decision problem 3SAT-3 that it is even strongly NP-hard to compute only the value of a maximum integral one-flow in a network. This result even holds if we restrict to networks in which the number of s-t-paths is polynomially bounded in the size of the network.

Remember from Section 4.3 that the integral max-1FP is still strongly NP-hard if we restrict it to acyclic networks.

We mentioned in the introduction that it is already NP-hard to compute only the value of a maximum one-flow. In general, this holds for the integral as well as for the fractional problem. For the integral problem we can state stronger results using the NP-hardness of the capacitated set packing decision problem and of the decision problem 3SAT-3.

Corollary 4.11. It is already NP-hard to determine the flow value of a maximum fractional/integral one-flow. In the integral case this is still true if we restrict to acyclic networks or to those in which the number of s-t-paths is polynomially bounded in the size of the network.

4.4.2 Integrality

We have already mentioned in Proposition 4.5 that the existence of a maximum one-flow where the flow value on each arc is integral does in general not imply the existence of an integral path decomposition where each path carries at most one unit of flow. Figure 4.5 shows an instance of the max-1FP in which the maximum one-flow yields an integral flow value on all arcs, although it is fractional. A similar network has been used in [9, 10] to prove the discrepancy of arcwise and pathwise integrality for the length-bounded flow problem.

The idea is to double the network from Figure 4.1. Then we can send 5.5 units from s to v along either chain of parallel arcs. These 11 units of flow are routed from v to t along the arc of capacity 11. Another unit of flow can be gained by sending half a unit of flow along each s-t-path using only arcs of capacity 4 from s to v plus the arc (v, t) of capacity 1. Note that the resulting flow saturates all arc capacities. This means that it is maximal while the total flow value on each arc is integral.

The considerations from Section 4.3 imply that a maximum integral one-flow from s to v has value 10. These 10 units of flow can be sent using the paths that use only arcs of capacity 4, the paths using only arcs of capacity

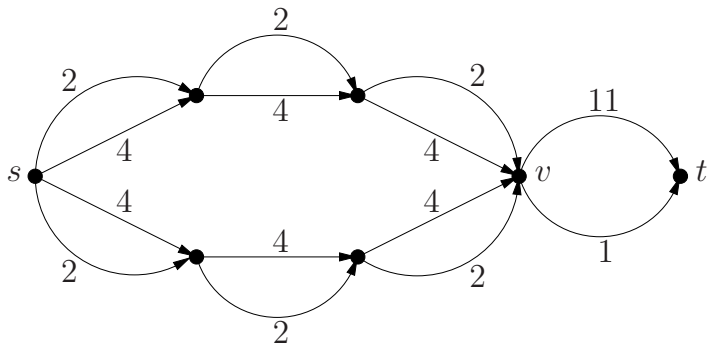

Figure 4.5: A network with a maximum one-flow that is arcwise integral, but pathwise fractional.

2 and those using exactly one arc of capacity 2. Then there is one capacity unit left on each arc of capacity 4. Thus, we can send 11 units of flow from s to t in total: We use the described maximum integral one-flow from s to v and forward this to t along the arc (v, t) of capacity 11. In addition, we use one path that uses only arcs of capacity 4 from s to v and forward this to t along the arc (v, t) of capacity 1.

4.5 An FPTAS for the Fractional Max-1FP

In this section we develop an FPTAS for the fractional max-1FP to prove the following theorem.

Theorem 4.12. For any $\epsilon > 0$ and any instance of the max-1FP with maximum flow value F^*, it is possible to compute a one-flow of value at least $(1 - \epsilon)F^*$ in time polynomial in the input size, ϵ^{-1}, and F^*.

First we show how to compute a one-flow of at least[2] a given value F that does not violate the arc capacities by more than a factor $(1 + \epsilon)$, for some $\epsilon > 0$, or to decide that no feasible flow of value F exists. A flow violating each arc capacity by at most a factor $(1 + \epsilon)$ is called $(1 + \epsilon)$-*approximate*. The computation of $(1 + \epsilon)$-approximate one-flows of given values is embedded in a geometric binary search later on. This proceeding yields the desired FPTAS.

4.5.1 Computing $(1 + \epsilon)$-Approximate One-Flows

To compute a $(1 + \epsilon)$-approximate one-flow of a given value, we use ideas by Plotkin, Shmoys, and Tardos [86]. They developed an appropriate algo-

[2]For better readability, we prospectively say that a flow has value F, although its value might as well be larger than that.

rithm for the fractional set packing problem. In this problem we have sets of capacitated elements and search for a packing, i.e., a family of sets such that each element appears in at most as many sets as its capacity permits. To compute $(1+\epsilon)$-approximate packings of a given size, Plotkin et al. use a Lagrange relaxation that penalizes the violation of the capacity constraints in the objective function. Iteratively, they choose reasonable Lagrange multipliers, compute a solution to the relaxed problem, and combine this with the current solution, until a $(1+\epsilon)$-approximate packing is found.

In the remainder of this section we adapt the approach in [86] to the fractional 1FP.

Let $\epsilon > 0$ and $F \geq 0$ be the value for which we want to find the mentioned one-flow. For a given one-flow x, we use $\lambda_x(a) := \sum_{P \in \mathcal{P}: a \in P} x_P/u(a)$ to denote the congestion on arc $a \in A$ and $\Lambda_x := \max_{a \in A} \lambda_x(a)$ to denote the overall congestion. Then a one-flow x is $(1+\epsilon)$-approximate if and only if $\Lambda_x \leq (1+\epsilon)$. For given arc costs $c : A \to \mathbb{R}_0^+$, we define $c_P := \sum_{a \in P} c(a)$ for all s-t-paths $P \in \mathcal{P}$. (The arc costs that are used later on in the algorithm are equivalent to Lagrange multipliers of a Lagrange relaxation that penalizes the violation of the arc capacities in the objective function.) We consider the following linear program for min-cost one-flows and use C to denote its minimum objective value corresponding to the cost function c.

$$\begin{aligned} \min \ & \sum_{P \in \mathcal{P}} c_P y_P & & \\ \text{s.t.} \ & \sum_{P \in \mathcal{P}} y_P \geq F & & \text{(min-cost LP)} \\ & 0 \leq y_P \leq 1 & \forall\ P \in \mathcal{P} \quad . & \end{aligned}$$

Lemma 4.13. Given a cost funtion $c : A \to \mathbb{R}_0^+$, (min-cost LP) can be solved in time polynomial in the input size and F.

Proof. In order to solve (min-cost LP) compute $\lceil F \rceil$ shortest paths according to the length function c. This can be done in time polynomial in the input size and F (see, e.g., [69]). An optimal solution to (min-cost LP) is given by assigning the flow value 1 to each of the $\lfloor F \rfloor$ shortest paths and additionally a flow value of $F - \lfloor F \rfloor$ to the $\lceil F \rceil$th shortest path. □

We say that a one-flow x fulfills the *approximation condition* (AC) if it holds, for a given cost function $c : A \to \mathbb{R}_0^+$, that

$$\sum_{P \in \mathcal{P}} c_P x_P - C \leq \epsilon \Big(\sum_{P \in \mathcal{P}} c_P x_P + \Lambda_x \sum_{a \in A} c(a) u(a) \Big) \quad .$$

The procedure IMPR-CONG((D,u),x,ϵ) described in Algorithm 2 returns a one-flow which is either $(1+6\epsilon)$-optimal or reduces the congestion of x by at least one half. We define a flow x to be $(1+\epsilon)$-*optimal*, for some $\epsilon > 0$, if $\Lambda_x \leq (1+\epsilon)\Lambda^*$ where Λ^* is the minimum congestion possible for a one-flow of value F in the underlying network, i.e., Λ^* is the minimum objective value of

$$\begin{aligned} &\min \Lambda \\ &\text{s.t.} \sum_{P\in\mathcal{P}} x_P \geq F \\ &\quad \sum_{\substack{P\in\mathcal{P}:\\ a\in P}} x_P \leq \Lambda u(a) && \forall\ a \in A \\ &\quad 0 \leq x_P \leq 1 && \forall\ P \in \mathcal{P} \quad . \end{aligned}$$

For minimum congestion problems, it is usually required that $\Lambda \geq 1$ so that each flow obeying all arc capacities is classified as optimal. But the additional requirement would complicate our notation and is therefore omitted. Nevertheless, we accept each solution with $\Lambda \leq 1$ as an optimum.

It holds for Λ^*, the corresponding flow x^*, and the minimum objective value C of (min-cost LP), that

$$\Lambda^* \sum_{a\in A} c(a)u(a) \geq \sum_{a\in A} c(a) \sum_{\substack{P\in\mathcal{P}:\\ a\in P}} x^*_P = \sum_{P\in\mathcal{P}} c_P x^*_P \geq C \quad . \tag{4.3}$$

First we show that IMPR-CONG((D,u),x,ϵ) matches its specification. For the following proofs we assume without loss of generality that the parameter ϵ is at most $1/6$.

Theorem 4.14. IMPR-CONG((D,u),x,ϵ) returns a one-flow which is either $(1+6\epsilon)$-optimal or reduces the congestion of x by at least one half.

Proof. This proof is similar to a proof by Plotkin, Shmoys, and Tardos [86]. We prove in Theorem 4.15 that IMPR-CONG((D,u),x,ϵ) terminates. Here, we assume that it outputs a flow x' with $\Lambda_{x'} \geq \Lambda_0/2$, because otherwise the congestion of x was reduced by at least one half. Then it holds that IMPR-CONG((D,u),x,ϵ) stopped due to the satisfication of (AC). We show that x' is $(1+6\epsilon)$-optimal in this case.

In the following we set $x'(\bar{a}) := \sum_{P\in\mathcal{P}:\bar{a}\in P} x'_P$, for any $\bar{a} \in A$. It holds that $\alpha \geq 2\Lambda_{x'}^{-1}\epsilon^{-1}\log(2m\epsilon^{-1})$. If we prove that for each $\bar{a} \in A$ it holds that $(1-\epsilon/2)\Lambda_{x'}u(\bar{a}) \leq x'(\bar{a})$ or $c(\bar{a})u(\bar{a}) \leq \frac{\epsilon}{2m}\sum_{a\in A} c(a)u(a)$ for the cost function c defined by x', we can conclude that $(1-\epsilon)\Lambda_{x'}\sum_{a\in A} c(a)u(a) \leq \sum_{P\in\mathcal{P}} c_P x'_P$.

Algorithm 2: IMPR-CONG((D,u),x,ϵ)

Input: A digraph $D=(V,A)$ with arc capacities $u: A \to \mathbb{R}^+$, a one-flow x of value F not obeying all arc capacities, $\epsilon > 0$.

Output: A one-flow of value F that is either $(1+6\epsilon)$-optimal or reduces the congestion of x by at least one half.

Set $\Lambda_0 := \Lambda_x$, $\alpha := 4\Lambda_0^{-1}\epsilon^{-1}\log(2m\epsilon^{-1})$, and $\sigma := \epsilon/(4\alpha F)$.
For all $a \in A$ let $c(a) := e^{\alpha\lambda_x(a)}/u(a)$.
Let y be the solution and C the objective value of (min-cost LP).
while $\Lambda_x \geq \Lambda_0/2$ *and* x *does not fulfill (AC)* **do**
  Set $x := (1-\sigma)x + \sigma y$.
  Set $c(a) := e^{\alpha\lambda_x(a)}/u(a)$.
  Compute a solution (y, C) for (min-cost LP) with the new values.
end
return x.

(We prove this later on.) Together with the fact that (AC) is satisfied by x' and c, this gives us that

$$\begin{aligned} C &\geq (1-\epsilon)\sum_{P\in\mathcal{P}} c_P x'_P - \epsilon\Lambda_{x'}\sum_{a\in A} c(a)u(a) \\ &\geq (1-\epsilon)^2\Lambda_{x'}\sum_{a\in A} c(a)u(a) - \epsilon\Lambda_{x'}\sum_{a\in A} c(a)u(a) \\ &\geq (1-3\epsilon)\Lambda_{x'}\sum_{a\in A} c(a)u(a) \qquad . \end{aligned}$$

Using (4.3) this implies for Λ^*—the minimum congestion possible for a one-flow of value F in the considered network—that

$$\Lambda_{x'} \leq \frac{C}{(1-3\epsilon)\sum_{a\in A} c(a)u(a)} \leq (1-3\epsilon)^{-1}\Lambda^* \leq (1+6\epsilon)\Lambda^*$$

and thus proves the theorem.

Let $\bar{a} \in A$. We prove now that $(1-\epsilon/2)\Lambda_{x'}u(\bar{a}) \leq x'(\bar{a})$ or $c(\bar{a})u(\bar{a}) \leq \frac{\epsilon}{2m}\sum_{a\in A} c(a)u(a)$.

By the definition of $\Lambda_{x'}$ and $c: A \to \mathbb{R}^+$ it holds that $\sum_{a\in A} c(a)u(a) \geq e^{\alpha\Lambda_{x'}}$. Now assume that $(1-\epsilon/2)\Lambda_{x'}u(\bar{a}) > x'(\bar{a})$, then it follows that $c(\bar{a})u(\bar{a}) = e^{\alpha x'(a)/u(a)} < e^{(1-\epsilon/2)\alpha\Lambda_{x'}}$. Thus,

$$\frac{c(\bar{a})u(\bar{a})}{\sum_{a\in A} c(a)u(a)} < \frac{e^{(1-\epsilon/2)\alpha\Lambda_{x'}}}{e^{\alpha\Lambda_{x'}}} = e^{-\epsilon\alpha\Lambda_{x'}/2} \leq e^{-\log(2m\epsilon^{-1})} = \frac{\epsilon}{2m}$$

which proves the claim.

Now we can conclude that $(1-\epsilon)\Lambda_{x'}\sum_{a\in A} c(a)u(a) \leq \sum_{P\in\mathcal{P}} c_P x'_P$. Let $A' := \{a \in A | (1-\epsilon/2)\Lambda_{x'}u(a) \leq x'(a)\}$. Then it holds that

$$\begin{aligned}\Lambda_{x'}\sum_{a\in A} c(a)u(a) &= \Lambda_{x'}\sum_{a\in A'} c(a)u(a) + \Lambda_{x'}\sum_{a\in A\setminus A'} c(a)u(a)\\ &\leq \frac{1}{1-\epsilon/2}\sum_{a\in A'} c(a)x'(a) + \Lambda_{x'}\sum_{a\in A\setminus A'}\frac{\epsilon}{2m}\sum_{a\in A} c(a)u(a)\\ &\leq \frac{1}{1-\epsilon/2}\sum_{P\in\mathcal{P}} c_P x'_P + \frac{\epsilon}{2}\Lambda_{x'}\sum_{a\in A} c(a)u(a)\end{aligned}$$

which implies

$$(1-\epsilon)\Lambda_{x'}\sum_{a\in A} c(a)u(a) \leq (1-\epsilon/2)^2\Lambda_{x'}\sum_{a\in A} c(a)u(a) \leq \sum_{P\in\mathcal{P}} c_P x'_P \quad .$$

□

Now let us turn to the analysis of the runtime of IMPR-CONG((D,u),x,ϵ).

Theorem 4.15. The procedure IMPR-CONG((D,u),x,ϵ) terminates after $O(\epsilon^{-3}F\log(m\epsilon^{-1}))$ iterations.

Proof. This proof is similar to a proof by Plotkin, Shmoys, and Tardos [86]. We consider IMPR-CONG((D,u),x,ϵ) and analyse the decrement of $C(A) := \sum_{a\in A} c(a)u(a)$ for the cost function $c : A \to \mathbb{R}^+$ throughout the execution. We show that the decrement of $C(A)$ due to a single iteration is at least $\epsilon^2\Lambda_0 F^{-1}C(A)$. Together with the fact that $e^{\alpha\Lambda_0/2} \leq C(A) \leq me^{\alpha\Lambda_0}$, until the last decrement of $C(A)$ is performed, the claim follows.

We start proving that the decrement of $C(A)$ due to a single iteration is at least $\epsilon^2\Lambda_0 F^{-1}C(A)$. In the following we denote the one-flow given at the beginning of the iteration by x and the one given at the end by x', i.e., $x' = (1-\sigma)x + \sigma y$, where y is an optimal solution of (min-cost LP) for the cost function c given by x. The cost function for x' is denoted as c' and $C'(A) := \sum_{a\in A} c'(a)u(a)$. We show that $C(A) - C'(A) \geq \frac{1}{2}\alpha\sigma\epsilon\Lambda_0 C(A)$ and thus $C(A) - C'(A) \geq \frac{1}{8}\epsilon^2\Lambda_0 F^{-1}C(A)$ (using the definition of σ).

Let $a \in A$. In the following we set $x(a) := \sum_{P\in\mathcal{P}:a\in P} x_P$ and $y(a) := \sum_{P\in\mathcal{P}:a\in P} y_P$. Since we assume that $u_{min} \geq 1$, it holds that $0 \leq x(a) \leq Fu(a)$ and $0 \leq y(a) \leq Fu(a)$ which implies $\alpha\sigma|x(a)-y(a)|/u(a) \leq \alpha\sigma F = \frac{\epsilon}{4} \leq \frac{1}{4}$.

Setting $\delta := \alpha\sigma(y(a)-x(a))/u(a)$ we obtain $|\delta| \leq 1/4$. Thus, a second-order Taylor series approximation gives us for any $r \in \mathbb{R}$ that $e^{r+\delta} \leq e^r +$

$\delta e^r + \frac{\epsilon}{2}|\delta|e^r$. This means for us that

$$\begin{aligned} c'(a) &= \frac{1}{u(a)} e^{\alpha\lambda_{x'}(a)} = \frac{1}{u(a)} e^{\alpha x'(a)/u(a)} \\ &= \frac{1}{u(a)} e^{\alpha(1-\sigma)x(a)/u(a)+\alpha\sigma y(a)/u(a)} = \frac{1}{u(a)} e^{\alpha x(a)/u(a)+\delta} \\ &\leq \frac{1}{u(a)} e^{\alpha\lambda_x(a)} + \frac{\alpha\sigma(y(a)-x(a))}{u^2(a)} e^{\alpha\lambda_x(a)} + \frac{\epsilon\alpha\sigma|y(a)-x(a)|}{2u^2(a)} e^{\alpha\lambda_x(a)} \\ &\leq c(a) + \frac{\alpha\sigma(y(a)-x(a))}{u(a)} c(a) + \frac{\epsilon\alpha\sigma(y(a)+x(a))}{2u(a)} c(a) \qquad . \end{aligned}$$

Now we can conclude for the decrement of $C(A)$ that

$$\begin{aligned} C(A) - C'(A) &= \sum_{a\in A} u(a)(c(a)-c'(a)) \\ &\geq \alpha\sigma \sum_{a\in A} c(a)(x(a)-y(a)) - \alpha\sigma\frac{\epsilon}{2} \sum_{a\in A} c(a)(x(a)+y(a)) \\ &= \alpha\sigma \sum_{P\in\mathcal{P}} c_P(x_P - y_P) - \alpha\sigma\frac{\epsilon}{2} \sum_{P\in\mathcal{P}} c_P(x_P + y_P) \\ &\geq \alpha\sigma(\sum_{P\in\mathcal{P}} c_P x_P - \sum_{P\in\mathcal{P}} c_P y_P - \epsilon \sum_{P\in\mathcal{P}} c_P x_P) \\ &\geq \alpha\sigma\epsilon\Lambda_x \sum_{a\in A} c(a)u(a) \\ &\geq \frac{1}{2}\alpha\sigma\epsilon\Lambda_0 C(A) \qquad , \end{aligned}$$

where we use the fact that (AC) is not fulfilled and that $\Lambda_x \geq \Lambda_0/2$ until IMPR-CONG((D,u),x,ϵ) stops. Using the definition of σ we have now that the decrement of $C(A)$ is at least $\frac{1}{8}\epsilon^2\Lambda_0 F^{-1} C(A)$.

In order to bound the number N of iterations we use the fact that $e^{\alpha\Lambda_0/2} \leq C(A) \leq me^{\alpha\Lambda_0}$ for all cost functions c considered throughout the execution of IMPR-CONG((D,u),x,ϵ), until the last decrement of $C(A)$ is fulfilled. We know that in each iteration $C(A)$ is at most multiplied with $(1-\frac{1}{8}\epsilon^2\Lambda_0 F^{-1})$. Thus, it holds that

$$(1 - \frac{\epsilon^2\Lambda_0}{8F})^{N-1} m e^{\alpha\Lambda_0} \geq e^{\alpha\Lambda_0/2}$$

which is equivalent to

$$(1 - \frac{\epsilon^2\Lambda_0}{8F})^{N-1} \geq m^{-1} e^{-\alpha\Lambda_0/2} \qquad .$$

Using that $\log(1-\beta) \leq -\beta$, for all $\beta \in (0,1)$, and that $\Lambda_0 > 1$ (since the initial flow x does not obey all arc capacities), it follows that

$$N - 1 \leq \frac{\log(m^{-1}e^{-\alpha\Lambda_0/2})}{\log(1-\frac{\epsilon^2\Lambda_0}{8F})} = -\frac{\log(me^{\alpha\Lambda_0/2})}{\log(1-\frac{\epsilon^2\Lambda_0}{8F})} = O(F\epsilon^{-2}\log(me^{\alpha\Lambda_0/2})) \quad .$$

Now we can use the definition of α and obtain

$$\begin{aligned} N &= O(\epsilon^{-2}F\log(m(2m\epsilon^{-1})^{2\epsilon^{-1}})) \\ &= O(\epsilon^{-2}F\log m + \epsilon^{-3}F\log(m\epsilon^{-1})) \\ &= O(\epsilon^{-3}F\log(m\epsilon^{-1})) \quad . \end{aligned}$$

This completes our proof. □

Lemma 4.16. Let $\epsilon, F > 0$ and let x be a one-flow (not necessarily obeying arc capacities) of value F in (D,u). Moreover, let the number of paths used in x be polynomial in the input size, ϵ^{-1}, and F. Then each iteration in IMPR-CONG((D,u),x,ϵ) can be done in time polynomial in the input size, ϵ^{-1}, and F.

Proof. The congestion Λ_x, α, σ, and the function $c : A \to \mathbb{R}^+$ can be computed in time polynomial in the input size.

Since IMPR-CONG((D,u),x,ϵ) terminates after $O(\epsilon^{-3}F\log(m\epsilon^{-1}))$ iterations, the number of paths used in x during the procedure keeps polynomial in the input size, ϵ^{-1}, and F. Thus, all updates of x can also be computed fast enough.

In Lemma 4.13 we proved that (min-cost LP) can be solved in time polynomial in the input size and F, so we are done. □

Theorem 4.17. Let $\epsilon, F > 0$. With a smart sequence of calls of IMPR-CONG it is possible to find a $(1+\epsilon)$-approximate one-flow of value F or to decide that no feasible one-flow of value F exists. Further, the number of calls of IMPR-CONG is polynomial in $\log(1/\epsilon)$ and $\log F$.

Proof. This proof is similar to a proof by Plotkin, Shmoys, and Tardos [86]. We differentiate between two cases—the one that $\epsilon \geq 1$ and the one that $\epsilon < 1$.

Let us start with the one that $\epsilon \geq 1$. In this case we call IMPR-CONG((D,u),x,1/6) for some arbitrary one-flow x (not necessarily obeying arc capacities) in (D,u) of value F. As long as the output flow x' is neither 2-optimal nor $\Lambda_{x'} \leq 2$, we call IMPR-CONG((D,u),x',1/6) with the respective output flow x' of the previous call. If finally $\Lambda_{x'} \leq 2$, then x' is

2-approximate and therefore, in particular, $(1+\epsilon)$-approximate. Otherwise x' is 2-optimal and $\Lambda_{x'} > 2$. Thus, no feasible one-flow of value F exists and we are also done.

Now we turn to the case that $\epsilon < 1$. First we proceed as above and check if the final 2-approximate flow x' is also $(1+\epsilon)$-approximate. If not, we call IMPR-CONG$((D,u),x',\tilde{\epsilon})$ as long as $1+\epsilon < \Lambda_{x'} \leq 1+6\tilde{\epsilon}$, i.e., x' is neither $(1+\epsilon)$-approximate nor $\Lambda_{x'} > 1+6\tilde{\epsilon}$. Here, x' is always the output flow of the previous call and $\tilde{\epsilon}$ is half of the "approximation parameter" $\tilde{\epsilon}$ of the previous call, i.e., we start with $\tilde{\epsilon} = 1/12$. If we do not stop due to the satisfaction of $\Lambda_{x'} > 1+6\tilde{\epsilon}$, we stop at the latest when $\tilde{\epsilon} \leq \epsilon/6$, because then $\Lambda_{x'} \leq 1+6\tilde{\epsilon} \leq 1+\epsilon$. If we prove that no feasible one-flow exists, if $\Lambda_{x'} > 1+6\tilde{\epsilon}$, we are done.

It suffices to show that if $\Lambda_{x'} > 1+6\tilde{\epsilon}$, then x' is $(1+6\tilde{\epsilon})$-optimal. Assume that x' is not $(1+6\tilde{\epsilon})$-optimal, then its congestion would have been halved since the last call of IMPR-CONG and x' would therefore be a feasible one-flow, because it holds for the $\bar{x}$ and $\bar{\epsilon}$ of the previous call that $\Lambda_{\bar{x}} \leq 1+6\bar{\epsilon} \leq 2$. Thus, $\Lambda_{x'} \leq 1 < 1+6\tilde{\epsilon}$—a contradiction.

In the first case ($\epsilon \geq 1$) we call IMPR-CONG at most $\lfloor \log \Lambda_x \rfloor \leq \lfloor \log F \rfloor$ times, because the procedure either returns a 2-optimal flow (then we stop the calls) or it reduces the congestion by at least one half. For the second case we need at most $O(\log(1/\epsilon))$ in addition, because we stop at the latest when $\tilde{\epsilon} \leq \epsilon/6$ and we halve $\tilde{\epsilon}$ in each step. Thus, the number of calls of IMPR-CONG is polynomial in $\log(1/\epsilon)$ and $\log F$ in both cases. □

With the insight from Theorem 4.17 we know that we need only some time that is polynomial in the input size, ϵ^{-1}, and F to find a $(1+\epsilon)$-approximate one-flow of value F or decide that no feasible one-flow of that value exists. This holds, because the first call of IMPR-CONG can be done with a one-flow (not obeying all arc capacities) using $\lceil F \rceil$ paths. Combining Theorem 4.15, Lemma 4.16, and Theorem 4.17 with the fact that all calls of IMPR-CONG (following the first one) are done with the flow resulting from the previous call and some constant $\tilde{\epsilon}$ or one that is larger than $\epsilon/12$, we obtain the mentioned runtime.

4.5.2 An FPTAS

To obtain a one-flow for a given instance of the max-1FP that obeys all arc capacities and approximates the maximum flow value F^* within a factor $(1-\epsilon)$, we embed the algorithm from Section 4.5.1 in a binary search to find a value $\bar{F}$ with $\bar{F} \leq F^* \leq (1+\tilde{\epsilon})\bar{F}$, for $\tilde{\epsilon} = (1-\epsilon)^{-1/2} - 1$. Then the desired flow is obtained by computing a $(1+\tilde{\epsilon})$-approximate one-flow of value

$\bar{F}$ and dividing all its flow values by $(1+\tilde{\epsilon})$. The resulting flow has value $(1+\tilde{\epsilon})^{-1}\bar{F} \geq (1-\epsilon)F^*$.

The binary search works as follows. For increasing values of $\ell \in \mathbb{N}$ (starting with $\ell = 0$) we use the algorithm from Section 4.5.1 to compute a 2-approximate one-flow of value 2^ℓ or decide that no feasible one-flow of that value exists. We stop with the first value $\bar{\ell}$ for ℓ for which the answer is that no feasible one-flow of value $\bar{F} := 2^{\bar{\ell}}$ exists. This takes us $O(\log F^*)$ steps. It follows that $\bar{F}/4 \leq F^* < \bar{F}$. Using geometric binary search as introduced in [50], we obtain $\bar{F}$ with $\bar{F} \leq F^* \leq (1+\tilde{\epsilon})\bar{F}$ within $O(\log(1/\tilde{\epsilon})) = O(\log(1/\epsilon))$ steps. This is done as follows. We set our upper bound for F^* to $B_u := \bar{F}$ and our lower bound to $B_\ell := \bar{F}/4$. Then we iteratively compute a $(1+\tilde{\epsilon}/8)$-approximate flow of value $\sqrt{B_\ell B_u}$ or decide that no feasible one-flow of that value exists. In the first case we update the lower bound to $B_\ell := \sqrt{B_\ell B_u}/(1+\tilde{\epsilon}/8)$, in the second case the upper bound to $B_u := \sqrt{B_\ell B_u}$. We stop when $B_u/B_\ell \leq 1+\tilde{\epsilon}$. Since after N iterations it holds that

$$\frac{B_u}{B_\ell} \leq (1+\frac{\tilde{\epsilon}}{8})^2 \left(\frac{F}{F/4}\right)^{\frac{1}{2^N}} \leq (1+\frac{\tilde{\epsilon}}{2})4^{\frac{1}{2^N}} \quad , \tag{4.4}$$

we obtain $B_u/B_\ell \leq 1+\tilde{\epsilon}$ after $N \geq \log(2/\log\frac{1+\tilde{\epsilon}}{1+\tilde{\epsilon}/2}) = O(\log(1/\tilde{\epsilon}))$ iterations.

Thus, we used $O(\log F^* + \log(1/\epsilon))$ steps in total in order to obtain $\bar{F}$ which is set to the final lower bound.

The algorithm is polynomial in input plus output size and ϵ^{-1}, because we have $O(\log F^* + \log(1/\epsilon))$ steps in which we compute an approximate one-flow. The time to compute $\lceil 2^\ell \rceil$ shortest paths is polynomial in the input size and F^* in each step, because $\lceil 2^\ell \rceil = O(F^*)$ for all considered values of ℓ. Since we need at least $(1-\epsilon)F^*$ paths to present a one-flow of value $(1-\epsilon)F^*$, F^* is polynomial in the output size.

4.6 Approximating the Integral Max-1FP

In this section we present approximation results for the integral max-1FP. We start by giving constant factor approximations for instances in which all arc capacities or the value of a maximum fractional one-flow are large. Afterwards we present a randomized $O(\log m)$-approximation algorithm for the general problem. We assume that all arc capacities are integral.

4.6.1 Constant Factor Approximations for Special Cases

For the composition of a constant factor approximation algorithm for instances in which the value of a maximum fractional one-flow is large, we first prove that the additive integrality gap of the max-1FP is less than m. This result follows from basic linear programming theory.

Proposition 4.18. The difference of the value F_F^* of a maximum fractional one-flow and the value F_I^* of a maximum integral one-flow is less than m.

Proof. Consider an optimal basic solution x to the maximum one-flow LP given in Section 4.2 and let $\mathcal{P}_F$ be the set of paths P for which x_P is fractional. It suffices to show that $|\mathcal{P}_F| \leq m$, because it then follows that we lose less than m units of flow by dropping the fractional part of an optimal basic solution. The remaining part yields an integral solution.

Let $\mathcal{A} = (\mathcal{A}_P)_{P \in \mathcal{P}_F}$ be the matrix given by the inequalities in (4.1), i.e., $\mathcal{A}_P \in \{0,1\}^A$ is the vector that has 1-entries for all $a \in P$ and 0-entries elsewhere. Further let $e_P \in \{0,1\}^{\mathcal{P}}$ be the vector that has a single 1-entry for $P \in \mathcal{P}$ and 0-entries elsewhere. Since x_P is fractional for $P \in \mathcal{P}_F$, the slack variable of the inequality $x_P \leq 1$ is positive. All vectors corresponding to positive variables in a basic solution must be linearly independent. It follows that the vectors $(\mathcal{A}_P, e_P)$, $(0, e_P) \in \{0,1\}^{A \cup \mathcal{P}}$ with $P \in \mathcal{P}_F$ are linearly indepedent. This implies immediately that the vectors $\mathcal{A}_P$ with $P \in \mathcal{P}_F$ must also be linearly independent. Since the dimension of these vectors is m, it follows that $|\mathcal{P}_F| \leq m$. □

Combining the insights from Proposition 4.18 and its proof with the FPTAS from the last section yields the following result.

Corollary 4.19. For any $\epsilon > 0$, an integral one-flow of value $F_I > (1 - \epsilon)F_F^* - m$ (where F_F^* is the value of an optimal fractional one-flow) can be computed in time polynomial in the input size, ϵ^{-1}, and F_I.

Proof. Consider any instance of the max-1FP. Let F_F^* be the value of a maximum fractional one-flow. By Theorem 4.12 we can compute a fractional one-flow $(\bar{x}_P)_{P \in \mathcal{P}}$ of value at least $(1 - \epsilon)F_F^*$, for any $\epsilon > 0$, in time polynomial in the input size, ϵ^{-1}, and F_F^*. Based on this, we compute another one-flow of the same or larger value. Consider the system of linear inequalities constructed in the following. We only take the set of paths $\bar{\mathcal{P}} := \{P \in \mathcal{P} \mid \bar{x}_P > 0\}$ with positive flow values into account. Then

the size of the system

$$\begin{aligned}
&\sum_{P\in\bar{\mathcal{P}}} x_P \geq \sum_{P\in\bar{\mathcal{P}}} \bar{x}_P && \\
&\sum_{\substack{P\in\bar{\mathcal{P}}:\\ a\in P}} x_P \leq u(a) && \forall\ a \in A \\
&0 \leq x_P \leq 1 && \forall\ P \in \bar{\mathcal{P}}
\end{aligned}$$

is polynomial in the input size of the original problem, ϵ^{-1}, and F_F^*. It follows that a basic solution to this system can be computed in polynomial time. The flow value of this solution is at least $(1-\epsilon)F_F^*$. For the fractional part of this basic solution, we can argue as in the proof of Proposition 4.18 that there are at most m paths carrying a fractional amount of flow. (Note that all vectors only have an additional 1 in the first entry.) Thus, we obtain an integral one-flow of value $F_I > (1-\epsilon)F_F^* - m$ when dropping the fractional part of the considered basic solution. Since $F_F^* < (1-\epsilon)^{-1}(F_I + m)$, the described procedure can be implemented to run in time polynomial in the input size, ϵ^{-1}, and F_I. □

As an immediate consequence, we obtain the following approximation result for the integral max-1FP.

Theorem 4.20. There exists a constant factor approximation algorithm for the integral max-1FP whose runtime is polynomial in input plus ouput size, if we restrict to instances whose maximum fractional flow value is larger than some constant $c > 1$ times their numbers of arcs.

In the remainder of this section we only consider instances of the max-1FP whose minimum arc capacity is $\Omega(\log m)$. Starting with a nearly optimal fractional solution we use randomized rounding to transform the fractional one-flow into an integral one. Randomized rounding was first introduced by Raghavan and Thompson [88]. We adapt a revision by Kleinberg [61] to the problem considered here. This yields a constant factor approximation with high probability.

For a given $\mu \in [0,1]$ and a nearly optimal fractional one-flow $(x_P)_{P\in\mathcal{P}}$, randomized rounding routes one unit of flow along path $P \in \mathcal{P}$ with probability μx_P. Obviously, this produces an integral one-flow which does not necessarily obey all arc capacities. With an analysis similar to one by Kleinberg [61] we show that the probability of violation of any arc capacity is at most $1/m$. For this proof we need the following lemma [61].

Lemma 4.21. Let $\mu \in (0,1]$ and $\Psi_1, \ldots, \Psi_N$ be completely independent Bernoulli trials with $E[\Psi_i] = \mu p_i$, for all $i \in \{1, \ldots, N\}$ and some $p_i \in [0,1]$. Then it holds for $\Psi := \sum_{i=1}^{N} \Psi_i$ and $p := \sum_{i=1}^{N} p_i$ that

$$Pr[\Psi > p] < (e\mu)^p \qquad .$$

Using Lemma 4.21, we prove that, for thoroughly chosen μ, the probability for a violation of any arc capacity is low.

Theorem 4.22. If $u_{min} \geq c \log m$, for some constant $c \in \mathbb{R}^+$, and $\mu := e^{-1}4^{-1/c}$, the probability, that any capacity constraint is violated after randomized rounding, will be less than $1/m$.

Proof. This proof is similar to one given by Kleinberg [61].

We assume to have a capacity function u and μ as given in the theorem. If we show for each arc that after randomized rounding its capacity is violated with probability less than $1/m^2$, we are done. Hence, consider any arc $a = (v, w) \in A$.

Note that randomized rounding works independently from the fact that all paths share common start and end nodes. Thus, we may assume without loss of generality that the flow along a in the fractional one-flow equals $u(a)$. Otherwise we could add artificial paths with corresponding flow across a.

For all $P \in \mathcal{P}$, let Ψ_P be the random 0/1-variable that indicates if P sends one unit of flow across a in the computed integral one-flow. Further, let $x_P(a)$ be 0, if $a \notin P$, and x_P otherwise. Then $E[\Psi_P] = \mu x_P(a)$ and $\sum_{P \in \mathcal{P}} x_P(a) = u(a)$. Using Lemma 4.21 it follows for $\Psi := \sum_{P \in \mathcal{P}} \Psi_P$ that

$$Pr[\Psi > u(a)] < (e\mu)^{u(a)} = \left(\frac{1}{4^{1/c}}\right)^{u(a)} \leq \frac{1}{4^{\log m}} = \frac{1}{m^2} \qquad . \tag{4.5}$$

Since Ψ is the random variable giving the number of paths that send one unit of flow across a in the computed integral one-flow, the proof is complete. □

It remains to prove, that randomized rounding can be used to find a constant factor approximation with high probability.

Theorem 4.23. Let $k \in \mathbb{N}$. For any instance of the integral max-1FP with $u_{min} = \Omega(\log m)$, randomized rounding needs $O(k)$ iterations to find a constant factor approximation with probability at least $1 - 2^{-k}$.

Proof. We may assume without loss of generality that $m > 2\mu^{-1} - 1$, otherwise we can efficiently solve the problem exactly.

Let v_I be the random variable indicating the amount of flow that is routed by the integral one-flow obtained from one application of randomized rounding. It follows that $E[v_I]$ is exactly μ times the amount of flow v_F that is routed by the fractional flow. Thus, $E[v_I]$ is only a constant factor smaller than the optimal value of an integral one-flow.

Let p be the probability for the event, that v_I is at least $\frac{\mu}{2}v_F$. Since $E[v_I] = \mu v_F$, it holds that

$$(1-p)\frac{\mu}{2}v_F + p \cdot v_F \geq \mu v_F,$$

which is equivalent to $p \geq \frac{\mu}{2-\mu}$.

Thus, v_I is at most some constant factor away from the value of an optimal one-flow with probability at least $\frac{\mu}{2-\mu}$.

It can still happen that the computed one-flow is infeasible. A feasible constant factor approximation is obtained with probability at least $q := \frac{\mu}{2-\mu} - \frac{1}{m} > 0$. For some constant $\ell \in \mathbb{N}$ that only depends on μ, it holds that $(1-q)^\ell < 1/2$, which means that the probability of a flop will be less than $1/2$ after ℓ iterations of the algorithm. Thus, we get that after ℓk iterations for some $k \in \mathbb{N}$ the probability of success will be at least $1-(1-q)^{\ell k} > 1-2^{-k}$, which completes the proof. □

4.6.2 An $O(\log m)$-Approximation

In order to obtain a randomized $O(\log m)$-approximation algorithm for arbitrary instances of the integral max-1FP, we use the computation of an approximate maximum integral one-flow in a modified network by giving a special treatment to those arcs whose capacity is smaller than $\log m$. In order to simplify the following construction, we first introduce further terminology. We call an arc whose capacity is at least $\log m$ *thick*; all other arcs are *thin*. A path is called *thick* if all its arcs are thick; otherwise it is called *thin*.

For a given instance of the max-1FP, we compute an approximate solution to the fractional problem using the FPTAS from Section 4.5 for some constant $\epsilon > 0$. If the total flow value along thick paths is at least half of the total flow value (and thus at most some constant factor smaller than the maximum integral flow value), we can use randomized rounding as explained above in order to obtain a constant factor approximation. Otherwise we can use the algorithm described in the following which computes an $O(\log m)$-approximation from the flow that is routed along thin paths.

The algorithm works as follows. First we delete the flow routed along thick paths. From now on we consider only the part of the underlying graph which is used by thin paths. For each thin arc (v, w) insert a new node $\tilde{v}$, delete the

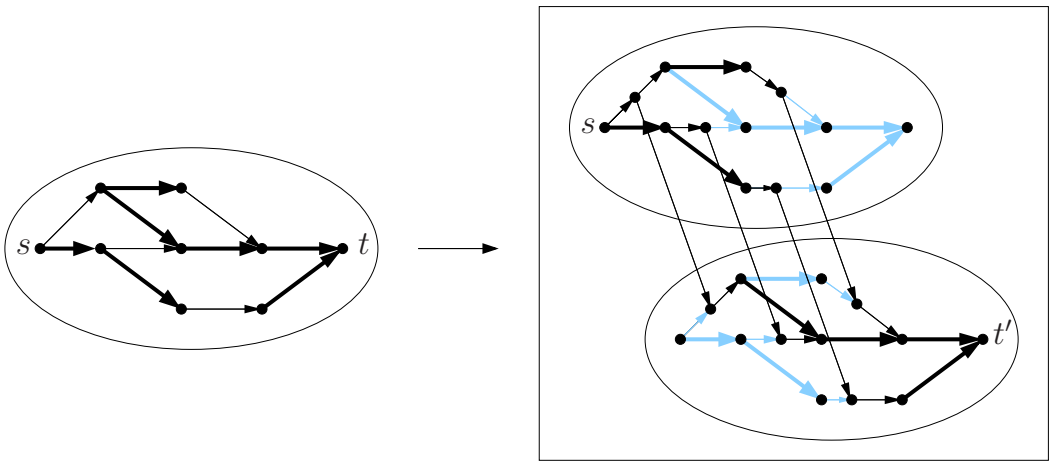

Figure 4.6: Rerouting of the original flow on the left in the 2-layer graph on the right. Gray arcs are not used by the final flow.

arc (v, w), insert the arcs $(v, \tilde{v})$ and $(\tilde{v}, w)$ and assign the capacity of (v, w) to them. (The flow is adjusted adequately using arcs $(v, \tilde{v})$ and $(\tilde{v}, w)$ instead of (v, w).) The graph of the resulting network is denoted by $D = (V, A)$, the set of newly inserted nodes by U. Next, we make a copy $D' = (V', A')$ of D. From each node in U we insert an arc to its copy in D'. The resulting graph is denoted by $\bar{D} = (\bar{V}, \bar{A})$.

We define v' to be the clone of $v \in V$ in V' and a' to be the clone of $a \in A$ in A'. An arc connecting a node $u \in U$ with u' is denoted as a_u. For $u \in U$ that was inserted to split an arc a of the original digraph, the capacity of a_u is the same as that of a.

We modify the considered fractional flow by rerouting all its paths from D to D' along the last thin arc at which a rerouting is possible. More detailed, this works as follows. Consider any path P that is used in the original fractional flow and let $(v, w) \in P$ be the last thin arc on P. (This arc does not exist in D.) Then the analogon to P in $\bar{D}$ uses the adjusted path P in D until it reaches v, then uses $(v, \tilde{v})$ and is rerouted to D' along $a_{\tilde{v}}$. In D' the new path uses the arc from $\tilde{v}'$ to w' and then the arcs corresponding to the ones P used in D after (v, w). (See Figure 4.6 for an illustration.) Let the resulting s-t'-flow be denoted by $\bar{x}$. Note that the value $|\bar{x}|$ of $\bar{x}$ is still only some constant factor smaller than the value of a maximum one-flow in the original network.

We choose integral capacities $u(a)$ and $u(a')$ for a thick arc $a \in A$ and its clone a' as follows. If $\lceil \bar{x}(a) \rceil + \lceil \bar{x}(a') \rceil$ is not larger than the original capacity of a, we set $u(a) = \lceil \bar{x}(a) \rceil$ and $u(a') = \lceil \bar{x}(a') \rceil$. Otherwise, we choose the capacities by rounding the smaller flow value up and the larger one down. (The sum of the resulting values is not larger than the original capacity of

a, because this was assumed to be integral.) For all thin arcs $a \in \bar{A}$, the capacity $u(a)$ is set to 1.

Lemma 4.24. For each arc $a \in \bar{A}$, it holds that $u(a) > \bar{x}(a)/\log m$.

Proof. First let $a \in A$ be a thick arc. If $\lceil \bar{x}(a) \rceil + \lceil \bar{x}(a') \rceil$ is not larger than the original capacity of a, it obviously holds that $u(a) \geq \bar{x}(a)$ and $u(a') \geq \bar{x}(a')$. For the other case, we assume without loss of generality that $\bar{x}(a) \leq \bar{x}(a')$. Then, $u(a) = \lceil \bar{x}(a) \rceil \geq \bar{x}(a)$ and $u(a') = \lfloor \bar{x}(a') \rfloor$ with $\bar{x}(a') \geq 1$, because otherwise $\lceil \bar{x}(a) \rceil + \lceil \bar{x}(a') \rceil \leq 2$ which is less than the original capacity of a.[3] It follows that $u(a') > \bar{x}(a')/2$.

Now let $a \in \bar{A}$ be a thin arc. Then, $u(a) = 1 > \bar{x}(a)/\log m$, because the original capacity of a was less than $\log m$. □

It follows immediately from Lemma 4.24 that a (usual) maximum s-t'-flow in $\bar{D}$ with capacities u has flow value at least $|\bar{x}|/\log m$, because $\bar{x}/\log m$ is a feasible s-t'-flow in this network. By network flow theory, an integral maximum s-t'-flow can be computed in polynomial time. Since the value of this flow is at least $|\bar{x}|/\log m$, it is only by a factor $O(\log m)$ smaller than the value of a maximum one-flow in the original network. If we reroute this flow to D, i.e., do not let it switch from D to D' and let it use the corresponding arcs in D instead, it is still feasible, because the sum of arc capacities $u(a)$ and $u(a')$ is at most the original capacity of $a \in A$. Further, the resulting flow does not send more than one unit of flow along each path, because each path uses at least one thin arc (otherwise it would not have been able to get from D to D') whose capacity is now 1. This gives us the following result.

Theorem 4.25. For any $k \in \mathbb{N}$, there exists an algorithm for the integral max-1FP that computes an $O(\log m)$-approximation with probability at least $1 - 2^{-k}$. The runtime of this algorithm is polynomial in input plus output size and k.

Additionally, it follows immediately from our construction that the (multiplicative) integrality gap of the max-1FP is $O(\log m)$.

4.7 Multicommodity One-Flows

In this section we consider the case that we have several source-sink-pairs—the multicommodity case. As before we always have a digraph $D = (V, A)$ with arc capacities $u : A \to \mathbb{R}^+$. Instead of a single source-sink-pair we now have *requests* $(s_i, t_i) \in V \times V$, for $i = 1, \ldots, K$, where $K \in \mathbb{N}$ denotes the

[3] The case $m \leq 4$ can be solved efficiently and is neglected here.

number of requests. We also call (s_i, t_i) the ith *commodity*. Again we assume that the minimal arc capacity u_{min} is at least 1.

We can think of different optimization problems for the multicommodity 1FP. On the one hand, we can simply maximize the total flow that is sent through the network, on the other hand, we can introduce a demand for each commodity which has to be fulfilled. In the second case it seems to be reasonable to minimize the congestion of the one-flow that is used in order to satisfy all demands. We consider both problems. For both problems we generalize the FPTAS from Section 4.5 for fractional one-flows. For the integral maximization problem we present a randomized $O(\sqrt{m})$-approximation algorithm in Section 4.7.1. A result by Guruswami et al. [48] shows that it is NP-hard to approximate the problem within a factor $O(m^{1/2-\epsilon})$, for any $\epsilon > 0$. For the integral minimum congestion problem we present an $O(\log m / \log\log m)$-approximation algorithm in Section 4.7.2.

In the following we use $\mathcal{P}_i$ $(i = 1, \ldots, K)$ to denote the set of s_i-t_i-paths in D and define $\mathcal{P} := \bigcup_{i=1}^{K} \mathcal{P}_i$.

4.7.1 Maximum Multicommodity One-Flows

With the definitions given above we can describe the *maximum multicommodity one-flow problem (max-mc-1FP)* by the linear program in Section 4.2. Since solving the min-cost LP from Section 4.5 can be adapted to the new situation easily by computing the $\lceil F \rceil$ shortest paths for each commodity and then choosing the $\lceil F \rceil$ overall shortest paths, the FPTAS can be used for the max-mc-1FP without any changes.

Theorem 4.26. For any $\epsilon > 0$ and any instance of the max-mc-1FP with maximum flow value F^*, it is possible to compute a maximum multicommodity one-flow of value $(1-\epsilon)F^*$ in time polynomial in the input size, ϵ^{-1}, and F^*.

We turn to integral multicommodity one-flows. Also in the multicommodity case Proposition 4.18 holds. This gives us a constant factor approximation algorithm whose runtime is polynomial in input plus ouput size if we restrict to instances whose maximum fractional flow value is larger than some constant $c > 1$ times its number of arcs.

Further, we can also apply randomized rounding as in Section 4.6.1 with a suitably chosen μ. In the case that $u_{min} = \Omega(\log m)$, we obtain again a constant factor approximation with high probability. For arbitrary arc capacities, a more detailed analysis (see [13]) of the randomized rounding

process yields the conclusion that we still obtain a randomized $O(\sqrt{m})$-approximation algorithm.[4]

Theorem 4.27. There exists a randomized $O(\sqrt{m})$-approximation algorithm for the integral max-mc-1FP. The runtime of this algorithm is polynomial in input plus output size.

This result hits the lower bound of approximability, unless P = NP. A result by Guruswami et al. [48] provides the following.

Theorem 4.28. For any $\epsilon > 0$, it is NP-hard to approximate the integral max-mc-1FP within a factor $O(m^{1/2-\epsilon})$. We cannot even find an algorithm with this approximation ratio whose runtime is polynomial in the input size and the maximum flow value.

To prove this claim we use an idea by Guruswami et al. [48] to build a reduction of the NP-complete 2DIRPATH problem. 2DIRPATH is the following problem.

> **2DIRPATH**
>
> GIVEN: A digraph $D = (V, A)$ with four specific mutually distinct nodes $s_1, t_1 \in V$ and $s_2, t_2 \in V$.
>
> QUESTION: Are there arc disjoint s_1-t_1- and s_2-t_2-paths in D?

Proof of Theorem 4.28. Consider any $\epsilon \in (0, 1/2)$ and set $\delta := 4\epsilon/(1-2\epsilon)$. Let $H = (\tilde{V}, \tilde{A})$ be an instance of the 2DIRPATH problem with specific nodes $\tilde{s}_1,\ \tilde{t}_1$ and $\tilde{s}_2,\ \tilde{t}_2$. Let $M := |\tilde{A}|$ denote the number of arcs in H.

We show that it is possible to solve the 2DIRPATH problem on H in polynomial time by using an $O(m^{1/2-\epsilon})$-approximation for a special instance of the integral max-mc-1FP.

Assume without loss of generality that there are $\tilde{s}_1$-$\tilde{t}_1$- and $\tilde{s}_2$-$\tilde{t}_2$-paths in H, because this can be checked in polynomial time and we are done if one of the required paths is missing.

Now we build an instance of the integral max-mc-1FP as follows. We insert $K = \omega(M^{1/\delta})$ requests $(s_1, t_1), \ldots, (s_K, t_K)$ and $(K^2 - K)/2$ copies of H, denoted by H_{ij}, for $i = 1, \ldots, K-1$, $j = i+1, \ldots, K$. Additionally we add arcs

- from s_1 to $\tilde{s}_1$ of H_{12},

[4] In this case μ is chosen as follows. If the value of the underlying fractional one-flow is given by F, we set $\mu = c \cdot \min\{F/m, 1\}$, for a constant c such that $8\mu^2 < 1$ and $(1-8\mu^2)^m > e^{-F(1-1/e)\mu/16}$. Please see [13] for further details.

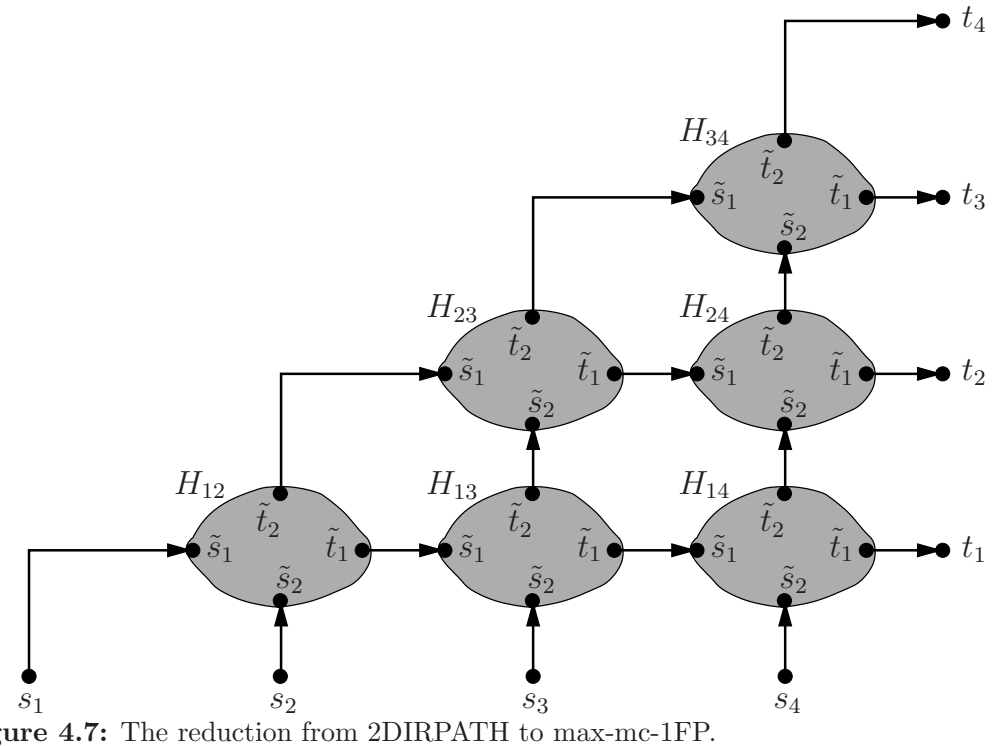

Figure 4.7: The reduction from 2DIRPATH to max-mc-1FP.

- from s_j to $\tilde{s}_2$ of H_{1j}, for all $j = 2, \ldots, K$,
- from $\tilde{t}_2$ of $H_{K-1,K}$ to t_K,
- from $\tilde{t}_1$ of H_{iK} to t_i, for all $i = 1, \ldots, K-1$,
- from $\tilde{t}_1$ of H_{ij} to $\tilde{s}_1$ of $H_{i,j+1}$, for all $i = 1, \ldots, K-2$, $j = i+1, \ldots, K-1$,
- from $\tilde{t}_2$ of H_{ij} to $\tilde{s}_2$ of $H_{i+1,j}$, for all $i = 1, \ldots, K-2$, $j = i+2, \ldots, K$,
- and from $\tilde{t}_2$ of $H_{i,i+1}$ to $\tilde{s}_1$ of $H_{i+1,i+2}$, for all $i = 1, \ldots, K-2$.

(An illustration of the resulting graph for $K = 4$ is given in Figure 4.7.) All arc capacities are set to 1.

The resulting graph consists of $m = O(K^2 M) = o(K^{2+\delta})$ arcs. Here, solving the integral max-mc-1FP is the same as computing as many arc-disjoint paths as possible, such that every two paths connect different requests. By construction we can connect either all requests simultaneously (if the instance of 2DIRPATH is a YES-instance) or only exactly one (if the instance of 2DIRPATH is a NO-instance).

Now assume that we have an $O(m^{1/2-\epsilon})$-approximation for the underlying integral max-mc-1FP. This one-flow connects at least two requests, if and only if it is theoretically possible to connect all of them. (If we obtained

a flow connecting only one request, although all of them can be connected simultaneously, we would have only a K-approximation. This gives a contradiction, because $K = \omega(m^{1/(2+\delta)}) = \omega(m^{1/2-\epsilon})$.) Thus, we can use the $O(m^{1/2-\epsilon})$-approximation to decide whether H is a YES- or a NO-instance of 2DIRPATH, because H is a YES-instance, if and only if all requests can be connected simultaneously. □

4.7.2 Minimizing Congestion

Now we assume that each request (s_i, t_i) $(i = 1, \ldots, K)$ has a corresponding positive demand d_i which has to be satisfied by a one-flow. We use $d_{max} := \max_{i=1,\ldots,K} d_i$ to denote the maximum demand. As usually performed for multicommodity flow problems, we look for a solution of minimum congestion. This problem (in the following denoted as *min-cong-1FP*) can be coded as follows.

$$
\begin{aligned}
&\min \lambda \\
&\text{s.t.} \sum_{P \in \mathcal{P}_i} x_P \geq d_i && \forall\ i = 1, \ldots, K \\
&\sum_{\substack{P \in \mathcal{P}: \\ a \in P}} x_P \leq \lambda u(a) && \forall\ a \in A \\
&\lambda \geq 1 \\
&0 \leq x_P \leq 1 && \forall\ P \in \mathcal{P}
\end{aligned}
$$

In order to find an FPTAS for the fractional min-cong-1FP we mainly use the procedure IMPR-CONG from Section 4.5.1. The corresponding (min-cost LP) is the following.

$$
\begin{aligned}
&\min \sum_{P \in \mathcal{P}} c_P y_P \\
&\text{s.t.} \sum_{P \in \mathcal{P}_i} y_P \geq d_i && \forall i = 1, \ldots, K && \text{(min-cost LP 2)} \\
&0 \leq y_P \leq 1 && \forall\ P \in \mathcal{P}
\end{aligned}
$$

Given a cost funtion $c : A \to \mathbb{R}_0^+$, (min-cost LP 2) can be solved in time polynomial in the input size and d_{max}. This is done by computing $\lceil d_i \rceil$ shortest paths, for all $i = 1, \ldots, K$. IMPR-CONG((D, u),x,ϵ) can be taken without any changes and returns a one-flow satisfying all demands whose

congestion is either a factor $(1 + 6\epsilon)$ away from optimum or is at most half the congestion of x.

Let $D := \sum_{i=1}^{K} d_i$. Then the congestion of any one-flow is at most D. This means that after at most $\lceil \log D \rceil$ runs of IMPR-CONG((D, u),x,$\epsilon/6$), each with the flow x returned by the previous run, we either obtain a flow x whose congestion is only a factor $(1 + \epsilon)$ away from optimum or is 1 and thus optimal. This insight yields our FPTAS.

Theorem 4.29. For any $\epsilon > 0$ and any instance of the min-cong-1FP with minimum congestion λ^*, it is possible to compute a multicommodity one-flow of congestion at most $(1+\epsilon)\lambda^*$ in time polynomial in the input size, ϵ^{-1}, and d_{max}.

It remains to give an approximation for the integral min-cong-1FP. For this case we again apply Raghavan and Thompson's [88] randomized rounding method to a nearly optimal fractional solution. In analogy to the unsplittable flow problem this method yields an $O(\log m / \log \log m)$-approximation algorithm.

Theorem 4.30. Applying randomized rounding to a nearly optimal fractional one-flow yields an $O(\log m / \log \log m)$-approximation to the min-cong-1FP with probability greater than 0.

The randomized rounding method in the minimum congestion version can be derandomized using the method of conditional probabilities [81, 87]. This gives us a deterministic $O(\log m / \log \log m)$-approximation algorithm for the min-cong-1FP.

Chapter 5

Abstract Flows

5.1 Introduction

The form of contents in this chapter differs from the ones in Chapters 2 to 4. We do not present fundamental new ideas, but rather give a survey of the problems considered so far in the context of abstract flows. In this respect, this chapter also summarizes the preceding ones.

Abstract flows are a new direction in network flow theory. They were introduced by Hoffman [52] who observed that the original proof of the Max-Flow-Min-Cut Theorem by Ford and Fulkerson [35] uses very few assumptions about the network. In this context, Hoffman departed from the usual structure of graphs and introduced an abstract model for network flows. For abstract flow problems we are given a ground set E of "edges"[1] with capacities $u : E \to \mathbb{R}^+$ and a family $\mathcal{P}$ of ordered subsets of E—the "paths". The order of the elements in $P \in \mathcal{P}$ is denoted as $<_P$. It is assumed that any two paths $P, Q \in \mathcal{P}$ which share an edge $e \in P \cap Q$ fulfill the requirement that there is a path contained in the element subset $(P, e, Q) := \{f \in P \mid f \leq_P e\} \cup \{f \in Q \mid f \geq_Q e\}$ (and also one contained in (Q, e, P)). In the following we refer to this requirement as (PR). But note that (PR) is not fulfilled in the general case of standard multicommodity flow problems.

With given edges E and paths $\mathcal{P}$ the linear program for the abstract maximum flow problem looks exactly like the one we are used to from standard maximum flows.

[1] Although we want to keep in mind that the considerations made in this chapter merely abstract the usual studies of flow, we resort to the notion "edges" instead of "arcs". In standard flow theory the notion "edge" is used if an arc is undirected. Here we consider some abstract structures of arcs which particularly do not have a direction. Hence, we prefer the notion "edges" which is also used in [78].

$$\begin{aligned} \max \ & \sum_{P \in \mathcal{P}} x_P & & \\ \text{s.t.} \ & \sum_{\substack{P \in \mathcal{P}: \\ e \in P}} x_P \leq u(e) & & \forall\ e \in E \\ & x_P \geq 0 & & \forall\ P \in \mathcal{P} \end{aligned}$$

The main result given by Hoffman [52] is an analogon to the Max-Flow-Min-Cut Theorem proving that, for integral edge capacities, there always exist integral optimal solutions to the given LP and its dual. Or in other words, there always exist an integral maximum flow and a(n) (integral) minimum cut, where a *cut* is a subset $C \subseteq E$ of edges such that $C \cap P \neq \emptyset$, for all $P \in \mathcal{P}$. The *capacity of a cut* C is defined as $u(C) := \sum_{e \in C} u(e)$. A *minimum cut* is one of minimum capacity.

Unfortunately the proof in [52] is not constructive. This fact left the search for a polynomial algorithm solving the (integral) abstract maximum flow problem. This algorithm was found by McCormick [78]. He allows $|E|$, but not $|\mathcal{P}|$, to appear in a polynomial runtime—a regulation that we want to adopt, because it is consistent with most applications of the abstract model. To deal with abstract paths, naturally the question arises how these can be represented. McCormick uses an oracle that—given a subset $F \subseteq E$—returns either a path (with its order) whose elements are all in F or the statement that no such path exists. But one could also think of other types of oracles, e.g., we could have one that returns a shortest path, for any length function on the edges, contained in a given subset of edges. If not stated differently, we assume to have such a shortest path oracle.

In the following the runtime of an algorithm is called *polynomial* if its number of steps is polynomial in the input size. Here, each oracle call is also considered as a single step.

The algorithm in [78] uses the idea to augment flow along an augmenting path. Since such a path uses backward edges in general, we need a translation of an augmenting path to the abstract model. This translation results in finding a set of paths along which flow is decreased or increased. The paths in this set overlap such that the change in the flow value sums up to 0 for most edges. The remaining edges build something that can be regarded as an augmenting path. (See Figure 5.1 to get a rough idea of what the set of paths looks like. The figure is drawn in the style of a figure in [78].) Finally, McCormick embeds the augmentation of flow in a standard scaling algorithm to obtain a maximum flow as well as a minimum cut in polynomial time.

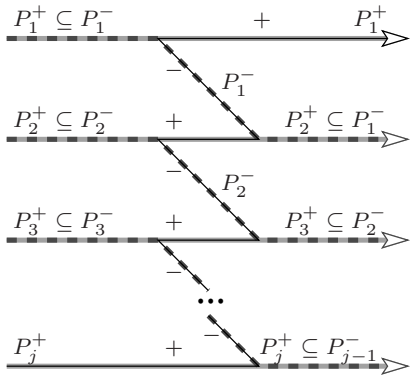

Figure 5.1: A set of paths along which flow is changed in McCormick's algorithm. Solid paths in light gray are those along which flow is increased. Dashed paths in dark gray are those along which flow is decreased. The solid black line indicates the "augmenting path".

In the following, we use $x(e) := \sum_{P \in \mathcal{P}: e \in P} x_P$ to define the flow along an edge $e \in E$. To underline that a flow obeys all edge capacities, we say that it is *feasible*. If nothing else is declared, we use $m := |E|$ to denote the number of edges.

Organization of this Chapter

We consider the problems discussed in the previous chapters in the context of abstract flows. We start with unsplittable flows in Section 5.2 and consider the multicommodity case as well as the single soure unsplittable flow problem. First of all, the analogous standard flow problems are transfered to the abstract model. The transformation already indicates that the known connection between the number of sources and the approximation factor carries forward to abstract flows. In Section 5.3 we consider the abstract k-splittable flow problem. The chapter is concluded by the study of abstract one-flows in Section 5.4.

5.2 Abstract Unsplittable Flows

To define an abstract unsplittable flow it is necessary to state which part of flow is to be sent unsplittably. In order to end up with a model related to the standard unsplittable flow problem (UFP), we partition the family $\mathcal{P}$ of paths into disjoint subsets $\mathcal{P}_1, \ldots, \mathcal{P}_K$, for some $K \in \mathbb{N}$. These subsets form the analogon to a request. If we use the notion *commodity* we mean the flow that is routed within only one of the set of paths. So *commodity* i stands for the flow that is to be routed within $\mathcal{P}_i$, for $i \in \{1, \ldots, K\}$. It sounds reasonable to require that each $\mathcal{P}_i$ fulfills (PR). But for our studies here, this

requirement is not needed in general.

Since all we have to represent our abstract paths is an oracle, the representation of $\mathcal{P}_1, \ldots, \mathcal{P}_K$ must be integrated in this construction. We assume that—given a subset of edges $F \subseteq E$ and a commodity $i \in \{1, \ldots, K\}$—the oracle is able to return a shortest path from $\mathcal{P}_i$ whose edges are in F or the information that no such path exists.

We first consider the case that (PR) is not satisfied by $\mathcal{P}$. This relates to standard multicommodity flows. If some certain specification of (PR), which refers to the single source UFP, is met, the problem becomes somewhat easier. This specification and the results for this case are discussed in Section 5.2.2.

Disregarding the satisfaction of (PR), the problem specification of both problems is the same. In addition to the standard input for abstract flows, we are given the mentioned sets $\mathcal{P}_1, \ldots, \mathcal{P}_K$ building a partition of $\mathcal{P}$ with positive demands d_i, for $i = 1, \ldots, K$. A feasible solution to the abstract unsplittable flow problem is a family of paths $\{P_1, \ldots, P_K\}$ where $P_i \in \mathcal{P}_i$, for all $i \in \{1, \ldots, K\}$, such that

$$\sum_{\substack{i \in \{1,\ldots,K\}: \\ e \in P_i}} d_i \leq u(e) \qquad \text{for all } e \in E.$$

Since in general it is not possible to find such a feasible solution, we consider the optimization problems to maximize the satisfied demand or to minimize the congestion. These problems are defined in analogy with the ones of the standard UFP.

To emphasize that a flow is not necessarily unsplittable, we call it *fractional*.

5.2.1 The Abstract Multicommodity UFP

In fact, most algorithms for the standard (multicommodity) UFP that are based on its path formulation are transmittable to the corresponding abstract problem—the one with sets $\bigcup_{i=1}^{K} \mathcal{P}_i = \mathcal{P}$ which do not necessarily satisfy (PR). Here, we present an extract of these algorithms.

Among the most important methods is the one of randomized rounding that was introduced by Raghavan and Thompson [88, 87]. To get a basis for the rounding, one has to have a fractional solution. Here, this is a (nearly) optimal fractional abstract flow. For both optimization problems mentioned above (the one to maximize the satisfied demand and the one to minimize the congestion), the fractional problem can be solved in polynomial time by linear programming.

The LP for the congestion minimization problem is the following.

$$\begin{aligned}
&\min \alpha \\
&\text{s.t.} \sum_{\substack{P\in\mathcal{P}:\\ e\in P}} x_P \le \alpha u(e) && \forall\, e \in E \\
&\sum_{P\in\mathcal{P}_i} x_P = d_i && \forall\, i = 1,\dots,K \\
&\alpha \ge 1 \\
&x_P \ge 0 && \forall\, P \in \mathcal{P}
\end{aligned}$$

Its dual is given by

$$\begin{aligned}
&\max \sum_{i=1}^{K} d_i y_i + z \\
&\text{s.t.} \sum_{e\in P} \ell(e) \ge y_i && \forall\, i = 1,\dots,K,\ P \in \mathcal{P}_i \\
&\sum_{e\in E} \ell(e)u(e) + z = 1 \\
&\ell(e) \ge 0 && \forall\, e \in E \\
&z \ge 0 \qquad .
\end{aligned}$$

The dual can be separated in polynomial time by shortest path computations. Given values $\ell(e) \ge 0$, for all $e \in E$, y_i, for $i = 1,\dots,K$, and $z \ge 0$, one can compute $\sum_{e\in E} \ell(e)u(e)+z$ in polynomial time and then simply has to use the oracle to check whether the shortest path in $\mathcal{P}_i$ (according to ℓ) has length at least y_i. Grötschel, Lovasz, and Schrijver [47] prove that optimization is just as hard as separation. It follows from the theory of linear programming that we can also compute an optimal fractional flow in polynomial time.

For the problem of maximizing the satisfied demand the following LP arises.

$$\begin{aligned}
&\max \sum_{P\in\mathcal{P}} x_P \\
&\text{s.t.} \sum_{\substack{P\in\mathcal{P}:\\ e\in P}} x_P \le u(e) && \forall\, e \in E \\
&x_P \ge 0 && \forall\, P \in \mathcal{P}
\end{aligned}$$

Its dual is given by

$$\begin{aligned} \min \ & \sum_{e \in E} u(e)\ell(e) & \\ \text{s.t.} \ & \sum_{e \in P} \ell(e) \geq 1 & \forall\ P \in \mathcal{P} \\ & \ell(e) \geq 0 & \forall\ e \in E \end{aligned}$$

Again, the dual can be separated in polynomial time. Given values $\ell(e) \geq 0$, for all $e \in E$, it can be decided by K shortest path computations, i.e., oracle calls, whether the values form a feasible solution. Thus, it again follows that the primal LP can be solved in polynomial time.

To obtain a flow that routes no more than d_i units of commodity $i \in \{1, \ldots, K\}$, we can expand all paths in $\mathcal{P}_i$ by a new edge e_i and give this edge a capacity of d_i.

Even the FPTASs by Garg and Könemann [40] and Fleischer [33] for the considered optimization problems for the standard multicommodity flow problem can be transfered to the abstract setting. Thus, it can be avoided to solve linear programs, if one is content with a nearly optimal flow as well.

Using an optimal fractional solution, randomized rounding yields an $O(\log m / \log\log m)$-approximation algorithm for the optimization problem to minimize the congestion of an abstract unsplittable flow which satisfies all demands. Of course, the results from Section 2.3, which show that the approximation ratio of randomized rounding is also $\Omega(\log m / \log\log m)$, transmit to abstract flows as well, because these form only a generalization of standard flows.

In the case of maximizing the totally satisfied demand in which all edge capacities must be obeyed, we can apply Kleinberg's [61] revision of Raghavan and Thompson's [88, 87] randomized rounding technique for the high-capacity UFP, where all edge capacities are in $\Omega(\log m)$. For any $k \in \mathbb{N}$, this can provide a constant factor approximation with probability $1 - 2^{-k}$. For the general case, there exists a greedy algorithm by Kolman and Scheideler [67] that yields $O(\sqrt{m})$-approximations and is transmittable to the abstract setting without any changes. Here, we can also transmit the inapproximability statement of Guruswami et al. [48] which says that, for any $\epsilon > 0$, it is NP-hard to approximate the underlying optimization problem within a factor $O(m^{1/2-\epsilon})$.

5.2.2 The Abstract Single Source UFP

We translate the (multicommodity) single source unsplittable flow problem to the abstract setting in the following way. We assume that for any two paths $P_i \in \mathcal{P}_i, P_j \in \mathcal{P}_j$ (for some $i,\ j \in \{1,\dots,K\}$) which share an edge $e \in P_i \cap P_j$ there is a path $P_j' \in \mathcal{P}_j$ with $P_j' \subseteq (P_i, e, P_j)$ and a path $P_i' \in \mathcal{P}_i$ with $P_i' \subseteq (P_j, e, P_i)$. In the following we refer to this property as (PR2).

Taking advantage of (PR2), which is even stronger than (PR), the algorithm by McCormick [78] can be used in order to find an abstract maximum flow which does not send more than d_i units along paths in $\mathcal{P}_i$, for each $i \in \{1,\dots,K\}$. This property of the flow can be obtained by an expansion of all paths in $\mathcal{P}_i$ by a new edge e_i with capacity d_i. Since this expansion does not destroy the fullfilment of (PR2), we can apply McCormick's algorithm and get even an integral maximum flow sending at most d_i units of flow along paths in $\mathcal{P}_i$. The integrality is needed when transfering the results for the single source UFP obtained by Skutella [95] to the abstract setting.

We pursue the objective to send all demands and find an unsplittable flow of minimum congestion. Considering a fractional flow x_0 satisfying all demands, the results by Skutella [95] provide an algorithm computing an unsplittable flow that satisfies all demands and sends at most $2x_0(e) + d_{max}$ units of flow along edge $e \in E$ where $d_{max} := \max_{i=1,\dots,K} d_i$.

To describe the algorithm we first assume that, for all $i,\ j \in \{1,\dots,K\}$, either d_i is a factor of d_j, or vice versa. Let $d_1,\dots,d_L$ (for some $L \le K$) be the different values of the demands in increasing order. For $i = 1,\dots,L$, the algorithm sets the capacity $u(e)$, for $e \in E$, to $x_{i-1}(e)$ rounded up to the nearest multiple of d_i. Then it computes a feasible d_i-integral flow satisfying all demands. (As elaborated above, we can use McCormick's algorithm in order to find such a flow.) Since now all commodities with demand d_i are sent along a single path, we store these paths in our solution and omit the corresponding commodities and the corresponding flow from further consideration.

To approximate the general case, for all $i \in \{1,\dots,K\}$, demand d_i is rounded down to $d_i' := d_{min} \cdot 2^{\lfloor \log(d_i/d_{min}) \rfloor}$, where $d_{min} := \min_{i=1,\dots,K} d_i$. The initial flow x_0 is decreased accordingly. Then the algorithm above is applied and finally the original demand is sent along each path sending flow in the algorithm's output. Skutella [95] proves that this algorithm yields an unsplittable flow sending at most $2x_0(e) + d_{max}$ units of flow along edge $e \in E$. If the initial flow x_0 is feasible and $d_{max} \le u_{min} := \min_{e \in E} u(e)$, this will come up to a congestion of 3.

Theorem 5.1. Given a fractional flow x which sends d_i units of commodity

$i \in \{1, \ldots, K\}$, we have a polynomial time algorithm that computes an unsplittable flow routing the same amount while sending at most $2x(e)+d_{max}$ units of flow along edge $e \in E$.

This result immediately suggests a 3-approximation algorithm for the congestion minimization problem, since the congestion of any initial fractional flow is increased by at most a factor 3. If we choose an optimal initial flow, the resulting unsplittable flow will be at most a factor 3 away from optimum, because the congestion of an optimal fractional flow yields a lower bound for the one of an optimal unsplittable flow. An optimal fractional flow can be found by linear programming (see Section 5.2.1). If we want to avoid to solve linear programs, it will still be possible to use the FPTAS by Garg and Könemann [40] and thus obtain a $(3 + \epsilon)$-approximation algorithm, for any $\epsilon > 0$.

Theorem 5.2. If $d_{max} \leq u_{min}$, there is a 3-approximation algorithm for the minimum congestion version of the abstract single source unsplittable flow problem.

The results mentioned above will offer a 2-approximation algorithm for the same problem, if for all $i,\ j \in \{1, \ldots, K\}$ either d_i is a factor of d_j, or vice versa (see [95]).

5.3 Abstract k-Splittable Flows

Again we study two different versions of the problem under consideration. These are the abstract single commodity k-splittable flow problem (k-SFP) and the abstract multicommodity k-SFP. The relation between the standard single commodity k-SFP and the abstract one is that we assume in the abstract setting that (PR) is fulfilled for the paths in $\mathcal{P}$. In the multicommodity case, we are again given a partition of $\mathcal{P}$ and do not require that (PR) is obeyed.

In addition to the set of edges E with its capacity function u and the family of paths $\mathcal{P}$ with its partition $\mathcal{P}_1, \ldots, \mathcal{P}_K$ and the demands $d_1, \ldots d_K$, an instance of the abstract multicommodity k-SFP consists of integral numbers k_i, for $i = 1, \ldots, K$, that specify how many paths may be used to route commodity i. The task is to find paths $P_1^i, \ldots, P_{k_i}^i$, for each commodity i, with corresponding nonnegative flow values $f_1^i, \ldots, f_{k_i}^i$ such that we meet the usual requirements of the k-SFP:

$$\sum_{j=1}^{k_i} f_j^i \;=\; d_i \text{ for } i = 1, \ldots, K, \quad \text{and} \quad \sum_{i=1}^{K} \sum_{\substack{j=1,\ldots,k_i: \\ e \in P_j^i}} f_j^i \;\leq\; u(e) \text{ for all } e \in E.$$

We do not require that the paths $P_1^i, \ldots, P_{k_i}^i$ are distinct, for $i = 1, \ldots, K$. Furthermore, we allow a path to have flow value 0. In other words, we may use less than k_i paths for commodity i.

The single commodity case of the k-SFP is the one in which we have only one commodity, i.e., $K = 1$. Further, we require that $\mathcal{P}$ obeys (PR). In this case, we omit the "i-index", when denoting the input.

Since all other notions are also used as defined in Section 3.2, we desist from further explanations here and refer to that section.

5.3.1 The Abstract Maximum Single Commodity k-SFP

Since in this section we only study the case that we have exactly one commodity, we omit the term "single commodity" for brevity of presentation. Further, we neglect the given demand, because our objective here is to find a k-splittable flow of maximum value.

Realize that in the abstract setting we can still assume without loss of generality that $k \leq m$. This holds, because for each abstract flow x there is (another) flow of the same value that uses at most m paths and sends no more than $x(e)$ units of flow along $e \in E$. This follows immediately from the theory of linear programming, because we have only m constraints in the LP formulation of the abstract maximum flow problem—besides nonnegativity constraints—or, in other words, each basic solution to the problem has at most m non-zero entries.

We start considering the problem to find a maximum abstract uniform exactly-k-splittable flow. Since the results by Baier, Köhler, and Skutella [11] are mainly based on standard integral maximum flow computations, we can transmit them to the abstract setting by involving McCormick's algorithm. If the maximum flow value D was known, our problem would reduce to finding a (D/k)-integral flow of value D. This problem can simply be solved by McCormick's algorithm. Unfortunately, we do not know the maximum flow value. But realize that in a maximum uniform exactly-k-splittable flow at least one edge $e \in E$ must be saturated. Let us say that e is used by $i \in \{1, \ldots, k\}$ paths, then $D/k = u(e)/i$. Thus, at most $|E|k$ values are possible for D/k and we need to compute just as many integral maximum flows using McCormick's algorithm.

To find a maximum abstract uniform k-splittable flow, i.e., one that spreads flow evenly among at most (not necessarily exactly) k paths, one simply has to find the maximum flow of all uniform exactly-i-splittable ones, for $i = 1, \ldots, k$.

Theorem 5.3. A maximum abstract uniform single commodity k-splittable

flow can be computed in polynomial time.

We can also transmit the Max-Flow-Min-Cut Theorem given in [11] to the abstract setting. As already defined, a cut is given by a set of edges C such that $P \cap C \neq \emptyset$, for all $P \in \mathcal{P}$. Baier et al. [11] define the *k-uniform cut capacity* $u_k(C)$ as the maximum volume of a packing of k identically sized packages into bins having sizes equal to the capacities of the edges in C. This definition can exactly be adopted to abstract networks.

Theorem 5.4. The minimum k-uniform cut capacity equals the value of a maximum abstract uniform single commodity exactly-k-splittable flow.

Proof. The proof is analogous to the one given in [11]. Let c^* be the minimum k-uniform cut capacity of an underlying instance for the considered problem. It is obvious that c^* yields an upper bound for the value v^* of a maximum abstract uniform exactly k-splittable flow.

We have to prove that $v^* \geq c^*$. To do so, we round all edge capacities down to the nearest multiple of c^*/k. This can only decrease v^*. The capacity of a standard minimum cut is now at least c^* and by McCormick's algorithm we can find a (c^*/k)-integral flow of value c^*. This flow is routed on at most k paths. Thus, the theorem is proven. □

Baier et al. [11] prove that the value of a maximum uniform exactly-k-splittable flow is at least half of the value of a maximum k-splittable flow. Since the proof can exactly be transmitted to the abstract setting, we obtain the following result.

Theorem 5.5. There exists a 2-approximation algorithm for the maximum abstract single commodity k-splittable flow problem.

5.3.2 The Abstract Multicommodity k-SFP

Again, we consider the optimization problem to find a solution of minimum congestion. To approximate optimal solutions to the standard multicommodity k-SFP, Baier et al. [11] use approximation algorithms for the UFP. We still assume without loss of generality that $k_i \leq m$, for all $i \in \{1, \ldots, K\}$. It follows that we can generalize the result in [11], which has already been mentioned in Section 3.1, that any ρ-approximation algorithm for the UFP yields a ρ-approximation algorithm for the uniform exactly-k-SFP to the abstract model. Thus, we obtain the following theorem by applying the fact that in the multicommodity case the minimum congestion of an optimal uniform exactly-k-splittable flow is at most twice as large as the minimum congestion of an optimal k-splittable flow (see also [11]).

Theorem 5.6. Any ρ-approximation algorithm for the minimum congestion version of the abstract unsplittable flow problem yields a 2ρ-approximation algorithm for the minimum congestion version of the abstract k-spittable flow problem.

Since our results in Section 3.3 are also based on solving the UFP, they can be transmitted to the abstract setting without any changes. Thus, we immediately obtain approximation algorithms for the minimum congestion version of the abstract k-SFP with path capacities. These approximate the underlying problems within a factor 2ρ where ρ is any approximation factor for the minimum congestion version of the abstract UFP.

5.4 Abstract One-Flows

Here we consider abstract one-flows. Given a set E of edges and a family $\mathcal{P}$ of abstract paths, the problem definition is the same as for the basic abstract flow problem given in the introduction of this chapter except that every path may send at most one unit of flow. Thus, the corresponding linear program looks like the one above, only the last constraint changes to "$0 \leq x_P \leq 1 \; \forall \; P \in \mathcal{P}$".

Our considerations here are based on a new type of oracle. We assume that this oracle is able to return a shortest path subject to the constraint that certain edges are used. That means, given a set $F \subseteq E$ of edges and obligatory edges $O \subseteq F$, the oracle returns a shortest path $P \in \mathcal{P}$ with $O \subseteq P \subseteq F$ or the information that no such path exists. This oracle enables us to solve the k shortest path problem in our abstract setting, see Lawler [69]. Lawler starts by computing a shortest path that is represented by 0/1-variables for the edges depending on whether an edge is used by the path or not. Afterwards, he iteratively fixes edge variables so that paths chosen so far are excluded from further consideration. Lawler also shows how the task of the oracle can be solved for standard networks in time polynomial in the input size and k.

All complexity results from Sections 4.4 and 4.7 adapt to abstract one-flows, since the standard 1FP is a special case of the abstract 1FP.

From the algorithmic point of view, we can also use most of the results from Chapter 4. Since there are no changes in the linear program from the one discussed in Section 4.5, we can adapt the FPTAS stated there.

For the integral abstract 1FP we obtain the same results as in Sections 4.6.1 and 4.7, because those are based on the LP formulations of the 1FP. The main results are a randomized $O(\sqrt{m})$-approximation algorithm for the abstract maximum one-flow problem and a randomized $O(\log m)$-

approximation algorithm for the abstract minimum congestion one-flow problem.

Unfortunately, we cannot obtain a randomized $O(\log m)$-approximation algorithm for the integral abstract maximum one-flow problem using the results from Section 4.6.2. It would be possible to adopt the 2-layer system as in Section 4.6.2, but the resulting path system would not necessarily fulfill (PR). Thus, we could not compute the necessary integral flow in polynomial time.

Bibliography

[1] A. Aggarwal, A. Bar-Noy, D. Coppersmith, R. Ramaswami, B. Schieber, and M. Sudan, *Efficient routing in optical networks*, Journal of the ACM **43**, no. 6 (1996), pp. 973–1001. [17]

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*, Prentice-Hall, Englewood Cliffs, 1993. [7, 59]

[3] M. Andrews and L. Zhang, *Hardness of the undirected congestion minimization problem*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 284–293. [16, 60]

[4] Y. Aumann and Y. Rabani, *Improved bounds for all optical routing*, in Proceedings of the 6th Annual ACM-SIAM Symposium of Discrete Algorithms, 1995, pp. 567–576. [17]

[5] Y. Azar and O. Regev, *Strongly polynomial algorithms for the unsplittable flow problem*, in Proceedings of the 8th Conference on Integer Programming and Combinatorial Optimization, 2001, pp. 15–29. [16, 17]

[6] A. Bagchi, *Efficient Strategies for Topics in Internet Algorithmics*, PhD thesis, The Johns Hopkins University, October 2002. [39]

[7] A. Bagchi, A. Chaudary, C. Scheideler, and P. Kolman, *Algorithms for fault-tolerant routing in circuit switched networks*, in Fourteenth ACM Symposium on Parallel Algorithms and Architectures, 2002. [39]

[8] A. Bagchi, A. Chaudhary, and P. Kolman, *Short length Menger's theorem and reliable optical routing*, in Proceedings of the 15th Annual Symposium on Parallel Algorithms and Architectures, 2003, pp. 246–247. [17]

[9] G. Baier, *Flows with Path Restrictions*, PhD thesis, TU Berlin, 2003. [40, 42, 51, 54, 75]

[10] G. Baier, T. Erlebach, A. Hall, E. Köhler, H. Schilling, and M. Skutella, *Length-bounded cuts and flows*, in Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, 2006, pp. 679–690. [40, 75]

[11] G. Baier, E. Köhler, and M. Skutella, *On the k-splittable flow problem*, Algorithmica **42** (2005), pp. 231–248. [37, 38, 39, 42, 44, 47, 48, 51, 54, 57, 105, 106]

[12] L. BARTHOLDI, *Counting paths in graphs*, L'Enseignement Mathématique **45** (1999), pp. 83–131. [60]

[13] A. BAVEJA AND A. SRINIVASAN, *Approximation algorithms for disjoint paths and related routing and packing problems*, Mathematics of Operations Research **25** (2000), pp. 255–280. [16, 17, 60, 91, 92]

[14] R. E. BELLMAN, *On a routing problem*, Quarterly of Applied Mathematics **16** (1958), pp. 87–90. [8]

[15] D. BERTSIMAS AND J. TSITSIKLIS, *Introduction to Linear Optimization*, Athena Scientific, 1997. [22]

[16] A. BLEY, *On the complexity of vertex-disjoint length-restricted path problems*, Computational Complexity **12** (2004), pp. 131–149. [40]

[17] U. BRANDES, G. NEYER, AND D. WAGNER, *Edge-disjoint paths in planar graphs with short total length*, Technical Report 19, Universität Konstanz, 1996. [40]

[18] R. E. BURKARD, K. DLASKA, AND B. KLINZ, *The quickest flow problem*, ZOR – Methods and Models of Operations Research **37** (1993), pp. 31–58. [40]

[19] R. G. BUSACKER AND P. J. GOWEN, *A procedure for determining a family of minimum-cost network flow patterns*, ORO Technical Paper 15, Operational Research Office, Johns Hopkins University, Baltimore, 1961. [11]

[20] C. CHEKURI AND S. KHANNA, *Edge disjoint paths revisited*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, 2003. [17, 60]

[21] G.-H. CHEN AND Y.-C. HUNG, *Algorithms for the constrained quickest path problem and the enumeration of quickest paths*, Computers and Operations Research **21** (1994), pp. 113–118. [60]

[22] Y. L. CHEN, *An algorithm for finding the k quickest paths in a network*, Computers and Operations Research **20** (1993), pp. 59–65. [60]

[23] Y. L. CHEN, *Finding the k quickest simple paths in a network*, Information Processing Letters **50** (1994), pp. 89–92. [60]

[24] J. CHUZHOY AND J. NAOR, *New hardness results for congestion minimization and machine scheduling*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, 2004, pp. 28–34. [16]

[25] E. W. DIJKSTRA, *A note on two problems in connection with graphs*, Numerische Mathematik **1** (1959), pp. 269–271. [8, 61]

[26] Y. Dinitz, N. Garg, and M. X. Goemans, *On the single source unsplittable flow problem*, Combinatorica **19** (1999), pp. 17–41. [4, 17, 23]

[27] S. E. Dreyfus, *An appraisal of some shortest path algorithms*, Operations Research **17** (1969), pp. 395–412. [61]

[28] J. Edmonds and R. M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, Journal of the ACM **19** (1972), pp. 248–264. [10, 11]

[29] D. Eppstein, *Finding the* k *shortest paths*, SIAM Journal on Computing **28** (1998), pp. 652–673. [60, 61]

[30] G. Exoo, *On line disjoint paths of bounded length*, Discrete Mathematics **44** (1983), pp. 317–318. [40]

[31] L. Fleischer and M. Skutella, *The quickest multicommodity flow problem*, in Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization, 2002, pp. 36–53. [41, 55, 56]

[32] L. Fleischer and É. Tardos, *Efficient continuous-time dynamic network flow algorithms*, Operations Research Letters **23** (1998), pp. 71–80. [41]

[33] L. K. Fleischer, *Approximating fractional multicommodity flow indepedent of the number of commodities*, SIAM Journal on Discrete Mathematics **13**, no. 4 (2000), pp. 505–520. [13, 60, 102]

[34] L. R. Ford, *Network flow theory*, Paper P-923, The Rand Corporation, Santa Monica, California, 1956. [8]

[35] L. R. Ford and D. R. Fulkerson, *Maximal flow through a network*, Canadian Journal of Mathematics **8** (1956), pp. 399–404. [2, 10, 59, 97]

[36] L. R. Ford and D. R. Fulkerson, *Constructing maximal dynamic flows from static flows*, Operations Research **6** (1958), pp. 419–433. [13, 40]

[37] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princenton University Press, Princenton, NJ, 1962. [10, 40]

[38] B. L. Fox, *k-th shortest paths and applications to the probabilistic networks*, ORSA/TIMS Joint National Meeting **23** (1975), p. B263. [61]

[39] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP–Completeness*, Freeman, San Francisco, 1979. [7, 15, 18]

[40] N. Garg and J. Könemann, *Faster and simpler algorithms for multicommodity flow and other fractional packing problems*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1998, pp. 300–309. [13, 60, 102, 104]

[41] I. GESSEL, *Counting paths in young's lattice*, Journal of Statistical Planning and Inference **34** (1993), pp. 125–134. [60]

[42] A. V. GOLDBERG AND R. E. TARJAN, *Finding minimum-cost circulations by cancelling negative cycles*, Journal of the ACM **36** (1989), pp. 873–886. [11]

[43] M. GRIGORIADIS AND L. G. KHACHIYAN, *An exponential-function reduction method for block-angular convex programs*, Networks **26**, no. 1.2 (1995), pp. 59–68. [60]

[44] M. GRIGORIADIS AND L. G. KHACHIYAN, *A sublinear-time randomized approximation algorithm for matrix games*, Operations Research Letters **18**, no. 2 (1995), pp. 53–58. [60]

[45] M. GRIGORIADIS AND L. G. KHACHIYAN, *Approximate minimum-cost multicommodity flows in $o(\epsilon^{-2}knm)$ time*, Mathematical Programming **75** (1996), pp. 477–482. [60]

[46] M. GRIGORIADIS AND L. G. KHACHIYAN, *Coordination complexity of parallel price-directive decomposition*, Mathematics of Operations Research **21** (1996), pp. 321–340. [60]

[47] M. GRÖTSCHEL, L. LOVASZ, AND A. SCHRIJVER, *Geometric Algorithms and Combinatorial Optimization*, Springer, Berlin, 1987. [13, 101]

[48] V. GURUSWAMI, S. KHANNA, R. RAJARAMAN, B. SHEPHERD, AND M. YANNAKAKIS, *Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, 1999, pp. 19–28. [16, 40, 60, 91, 92, 102]

[49] A. HALL, S. HIPPLER, AND M. SKUTELLA, *Multicommodity flows over time: Efficient algorithms and complexity*, in Proceedings of the 30th International Colloquium on Automata, Languages and Programming, 2003, pp. 397–409. [41]

[50] R. HASSIN, *Approximation schemes for the restricted shortest path problem*, Mathematics of Operations Research **17**, no. 1 (1992), pp. 36–42. [84]

[51] D. S. HOCHBAUM, editor. *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, Boston, Massachusetts, 1996. [7]

[52] A. J. HOFFMAN, *A generalization of max flow-min cut*, Mathematical Programming **6** (1974), pp. 352–359. [97, 98]

[53] B. HOPPE, *Efficient dynamic network flow algorithms*, PhD thesis, Cornell University, 1995. [41]

[54] B. Hoppe and É. Tardos, *The quickest transshipment problem*, Mathematics of Operations Research **25** (2000), pp. 36–62. [41]

[55] M. Iri, *A new method for solving transportation-network problems*, Journal of the Operations Research Society of Japan **3** (1960), pp. 27–87. [11]

[56] A. Itai, Y. Perl, and Y. Shiloach, *The complexity of finding maximum disjoint paths with length constraints*, Networks **12** (1982), pp. 277–286. [40]

[57] W. S. Jewell, *Optimal flow through networks*, Interim Technical Report 8, MIT, 1958. [11]

[58] M. Klein, *A primal method for minimum cost flows with applications to the assignment and transportation problems*, Management Science **14** (1967), pp. 205–220. [11]

[59] P. Klein, S. A. Plotkin, D. B. Shmoys, and E. Tardos, *Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts*, SIAM Journal of Computing **23**, no. 3 (1994), pp. 466–487. [60]

[60] J. Kleinberg and R. Rubinfeld, *Short paths in expander graphs*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, 1996, pp. 86–95. [16]

[61] J. M. Kleinberg, *Approximation Algorithms for Disjoint Path Problems*, PhD thesis, Massachusetts Institute of Technology, May 1996. [15, 17, 18, 60, 86, 87, 102]

[62] J. M. Kleinberg, *Single-source unsplittable flow*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, 1996, pp. 68–77. [16, 17]

[63] R. Koch, M. Skutella, and I. Spenke, *Approximation and complexity of k-splittable flows*, in Proceedings of the 3rd International Workshop on Approximation and Online Algorithms, 2006, pp. 244–257. [39]

[64] S. G. Kolliopoulos, *Edge-disjoint paths and unsplittable flow.* Draft chapter from the forthcoming "Handbook of Approximation Algorithms and Metaheuristics", edited by T. F. Gonzalez, 2005. [60]

[65] S. G. Kolliopoulos and C. Stein, *Approximation algorithms for single-source unsplittable flow*, SIAM Journal on Computing **31** (2002), pp. 919–946. [17]

[66] S. G. Kolliopoulos and C. Stein, *Approximating disjoint-path problems using packing integer programs*, Mathematical Programming Series A **99** (2004), pp. 63–87. [60]

[67] P. KOLMAN AND C. SCHEIDELER, *Improved bounds for the unsplittable flow problem*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 184–193. [16, 40, 102]

[68] B. KORTE AND J. VYGEN, *Combinatorial Optimization. Theory and Algorithms*, Springer, Berlin, 2000. [7]

[69] E. L. LAWLER, *A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem*, Management Science **18** (1972), pp. 401–405. [60, 61, 77, 107]

[70] T. LEIGHTON, F. MAKEDON, S. A. PLOTKIN, C. STEIN, AND E. TARDOS, *Fast approximation algorithms for multicommodity flow problems*, Journal of Computer and System Sciences **50**, no. 2 (1995), pp. 228–243. [60]

[71] T. LEIGHTON, S. RAO, AND A. SRINIVASAN, *Multicommodity flow and circuit switching*, in Proceedings of the 31st Hawaii International Conference on System Sciences, 1998, pp. 459–465. [16, 17, 19, 20]

[72] L. LOVÁSZ, V. NEUMANN-LARA, AND M. PLUMMER, *Mengerian theorems for paths of bounded length*, Periodica Mathematica Hungarica **9** (1978), pp. 269–276. [40]

[73] J. F. LUCAS, *Paths and pascal numbers*, Two-Year College Mathematics Journal **14**, no. 4 (1983), pp. 329–341. [59]

[74] A. R. MAHJOUB AND S. T. MCCORMICK, *The complexity of max flow and min cut with bounded-length paths.* Unpublished manuscript, 2003. [40]

[75] M. MARTENS AND M. SKUTELLA, *Flows on few paths: Algorithms and lower bounds*, in Proceedings of the 12th Annual European Symposium on Algorithms, 2004, pp. 520–531. [18, 41]

[76] M. MARTENS AND M. SKUTELLA, *Flows on few paths: Algorithms and lower bounds*, Networks **48**, no. 2 (2006), pp. 68–76. [18, 41]

[77] M. MARTENS AND M. SKUTELLA, *Length-bounded and dynamic k-splittable flows*, in Operations Research Proceedings 2005, 2006, pp. 297–302. [41]

[78] S. T. MCCORMICK, *A polynomial algorithm for abstract maximum flows.* UBC Faculty of Commerce Working Paper 95-MSC-001, 1998. [97, 98, 103]

[79] N. MEGGIDO, *Combinatorial optimization with rational objective functions*, Mathematics of Operations Research **4** (1979), pp. 414–424. [40]

[80] E. F. MOORE, *The shortest path through a maze*, in Proceedings of the International Symposium on the Theory of Switching, Part II, Cambridge, Massachusetts, 1959, Harvard University Press, pp. 285–292. [8]

[81] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995. [95]

[82] L. Niepel and D. Safaríková, *On a generalization of Menger's theorem*, Acta Mathematica Universitatis Comenianae (1983), pp. 275–284. [40]

[83] J. B. Orlin, *A faster strongly polynomial minimum cost flow algorithm*, Operations Research **41** (1993), pp. 338–350. [11]

[84] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, 1994. [71]

[85] Y. Perl and D. Ronen, *Heuristics for finding a maximum number of disjoint bounded paths*, Networks **14** (1984), pp. 531–544. [40]

[86] S. A. Plotkin, D. B. Shmoys, and É. Tardos, *Fast approximation algorithms for fractional packing and covering problems*, Mathematics of Operations Research **20** (1995), pp. 257–301. [60, 61, 76, 77, 78, 80, 82]

[87] P. Raghavan, *Probabilistic construction of deterministic algorithms: approximating packing integer programs*, Journal of Computer and System Sciences **37** (1988), pp. 130–143. [15, 95, 100, 102]

[88] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica **7** (1987), pp. 365–374. [4, 15, 19, 86, 95, 100, 102]

[89] P. Raghavan and E. Upfal, *Efficient routing in all-optical networks*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, 1994, pp. 134–143. [17]

[90] L. Roditty, *On the k-simple shortest paths problem in weighted directed graphs*, in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 920–928. [61]

[91] J. B. Rosen, S.-Z. Sun, and G.-L. Xue, *Algorithms for the quickest path problem and the enumeration of quickest paths*, Computers and Operations Research **18** (1991), pp. 579–584. [60]

[92] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley and Sons, Chichester, 1986. [13]

[93] A. Schrijver, *Combinatorial Optimization. Polyhedra and Efficiency*, Springer, Berlin, 2003. [7, 59]

[94] F. Shahrokhi and D. W. Matula, *The maximum concurrent flow problem*, Journal of the ACM **37** (1990), pp. 318–334. [60]

[95] M. SKUTELLA, *Approximating the single source unsplittable min-cost flow problem*, Mathematical Programming **91** (2002), pp. 493–514. [17, 103, 104]

[96] R. P. STANLEY, *A matrix for counting paths in acyclic digraphs*, Journal of Combinatorial Theory Series A **74**, no. 1 (1996), pp. 169–172. [60]

[97] L. G. VALIANT, *The complexity of enumeration and reliability problems*, SIAM Journal on Computing **8**, no. 3 (1979), pp. 410–421. [59]

[98] J. Y. YEN, *Finding the k shortest loopless paths in a network*, Management Science **17**, no. 11 (1971), pp. 712–716. [60]

[99] N. E. YOUNG, *Sequential and parallel algorithms for mixed packing and covering*, in IEEE Symposium on Foundations of Computer Science, 2001, pp. 538–546. [60]