Approximation and Complexity of k–Splittable Flows

Ronald Koch[1], Martin Skutella[1], Ines Spenke[2]

[1] Universität Dortmund, Fachbereich Mathematik, D – 44221 Dortmund, Germany, {ronald.koch, martin.skutella}@math.uni–dortmund.de, http://www.mathematik.uni-dortmund.de/lsv/

[2] Technische Universität Berlin, Institut für Mathematik, D – 10623 Berlin, Germany, spenke@math.tu–berlin.de, http://www.math.tu-berlin.de/∼spenke/

Abstract. Given a graph with a source and a sink node, the NP–hard maximum k–splittable flow (MkSF) problem is to find a flow of maximum value with a flow decomposition using at most k paths [**?**]. The multicommodity variant of this problem is a natural generalization of disjoint paths and unsplittable flow problems.
Constructing a k–splittable flow requires two interdepending decisions. One has to decide on k paths (routing) and on the flow values on these paths (packing). We give efficient algorithms for computing exact and approximate solutions by decoupling the two decisions into a first packing step and a second routing step. Our main contributions are as follows:

(i) We show that for constant k a polynomial number of packing alternatives containing at least one packing used by an optimal MkSF solution can be constructed in polynomial time. If k is part of the input, we obtain a slightly weaker result. In this case we can guarantee that, for any fixed $\epsilon > 0$, the computed set of alternatives contains a packing used by a $(1-\epsilon)$–approximate solution. The latter result is based on the observation that $(1-\epsilon)$–approximate flows only require constantly many different flow values. We believe that this observation is of interest in its own right.

(ii) Based on (i), we prove that, for constant k, the MkSF problem can be solved in polynomial time on graphs of bounded treewidth. If k is part of the input, this problem is still NP–hard and we present a polynomial time approximation scheme for it.

(iii) Finally, we provide a comprehensive overview of the complexity and approximability landscape of MkSF for different values of k.

1 Introduction

Many applications in transport, telecommunication, production or traffic are modelled as flow problems. In classic flow theory, flow is sent through a network from sources to sinks respecting edge capacities. It does not matter on how many paths the flow is sent. It can split into small flow portions along a large number of paths. But many applications do not allow an arbitrarily large number of paths. For example, in logistics commodities are usually transported with a

given number of vehicles. This bounds the number of paths that can be used simultaneously. Another example is data transport in communication networks. Communication systems often split data into packages. These packages traverse the network along different paths. Every package has to carry full information about source and target of the data, about the position of this package among other packages, and so on. It is therefore not efficient to split data into too many packages. As a consequence, various applications require that the flow does not use too many paths. Classical flow algorithms do not take such restrictions into account.

Problem description. Let $G = (V, E)$ be a connected undirected or directed graph with n nodes and m edges with capacities $u : E \to \mathbb{Q}_{\geq 0}$. Moreover, there is a source and a sink node $s, t \in V$. Baier, Köhler, and Skutella [**?**] introduce the concept of *k–splittable flows*. For a given number k, a feasible s,t–flow is called *k–splittable* if it can be decomposed into flows along at most k paths leading from s to t. We do not require the paths to be disjoint, not even different. The Maximum k–Splittable Flow problem (MkSF) is to find a k–splittable s,t–flow of maximum value. Of course, k–splittability can also be considered in the more general multi–commodity setting. Then the number of s_i, t_i–paths is restricted for each commodity i. In this paper, however, we concentrate on the single–commodity case.

Results from the literature. Since the seminal work of Ford and Fulkerson [**?**], there has been a vast amount of literature on classical s,t–flows with no restriction on the number of paths used. It is well known that a maximum s,t–flow can be computed in polynomial time, for example, by augmenting path algorithms. Another classical result states that any s,t–flow can be decomposed into flow on at most m paths and cycles. For further details we refer to the book by Ahuja, Magnanti, and Orlin [**?**].

Kleinberg [**?**] introduces *unsplittable flows*. These multicommodity flows route the total demand of each commodity along one single path. They generalize edge–disjoint paths. Kleinberg analyses complexity and approximation algorithms for different unsplittable flow problems, e.g. for minimizing the congestion on edges or equivalently maximizing the throughput, for the problem of minimizing the number of rounds needed to satisfy all demands and for the problem of maximizing the total demand which can be routed simultaneously. In the multicommodity setting, k–splittable flows constitute a generalization of unsplittable flows.

Baier, Köhler, and Skutella [**?**] (see also [**?**]) investigate k–splittable flows in the single- and in the multi–commodity setting. They prove NP–hardness of MkSF in directed graphs for all constant $k \geq 2$. For the special case of the *uniform* MkSF, where all k paths must carry the same amount of flow, they give a maxflow–mincut type result as well as an $O(km \log n)$ algorithm that computes an optimum solution. Based on these insights, they present $\frac{1}{2}$–approximation algorithms for the general MkSF problem. Bagchi, Chaudhary, Scheideler, and Kolman [**?**] consider fault tolerant routings in networks and define notions similar to k–splittable flows. To ensure connection for each commodity

for up to $k-1$ edge failures in the network, they require edge disjoint flow–paths per commodity. Martens and Skutella [**?**] consider a new variant of k–splittable multi–commodity flows with upper bounds on the amount of flow sent along each path. The objective is to minimize the congestion of arcs. They prove that any ρ–approximation for the unsplittable flow problem gives a 2ρ–approximation for two different variants of the considered problem.

Krysta, Sanders, and Vöcking [**?**] consider related problems in the area of machine scheduling problems by imposing a bound on the number of preemptions of each task. In their k–splittable scheduling problem, each task can be split into at most $k \geq 2$ pieces that are assigned to different machines. They describe a polynomial time algorithm for finding an exact solution for the k–splittable scheduling problem and a slightly more general problem. This algorithm has a running time which is exponential in the number of machines but linear in the number of tasks.

Many NP–hard problems on graphs become easy when restricted to special graph classes. In this context, graphs of bounded treewidth have turned out to be a particularly successful concept. Originally introduced by Robertson and Seymour [**?**] in the context of graph minors, these graphs are also relevant in several practical applications. Bodlaender [**?**] presents a general framework for obtaining polynomial algorithms for problems in graphs of bounded treewidth that are NP–hard in general graphs. Bodlaender [**?**] and Arnborg, Lagergren, and Seese [**?**] give general characterizations of problems that can be solved in polynomial time on graphs of bounded treewidth. The MkSF problem does not fall into one of these classes of problems. For a more detailed account of concepts and results in this area we refer to the survey paper by Bodlaender [**?**].

The only paper we are aware of that considers flows in graphs of bounded treewidth is the one by Hagerup et al. [**?**]. Given a graph with a constant number of terminals and with arc capacities, they show that all realizable demand/supply patterns at the terminals can be found efficiently in graphs of bounded treewidth.

Our paper. Constructing k–splittable flows requires to decide which paths should be used and what flow values should be sent. Of course, these two decisions cannot be made independently of each other but are coupled by the requirement to obey arc capacities. A natural approach is to first choose a collection of paths $(P_1, \ldots, P_k)$. Arc capacities then bound possible tuples of flow values $(f_1, \ldots, f_k)$ on these paths. In this paper we take the reverse approach. We first fix flow values $(f_1, \ldots, f_k)$ that we wish to send (packing). Then, in the second step (routing), we try to find a collection of paths $(P_1, \ldots, P_k)$ on which these flow values can be routed without violating arc capacities.

In Section 2 we consider the packing step, first for fixed k, then for k being part of the input. The number of possibilities for flow values $(f_1, \ldots, f_k)$ in an optimal solution of MkSF is a priori not bounded. For fixed k, we describe how to determine a polynomial number of alternatives for $(f_1, \ldots, f_k)$ containing the flow value pattern of at least one optimal solution to MkSF. These alternatives are determined in polynomial time by solving certain linear equation systems. We do not know whether all of these alternatives can be routed in G without

violating capacities. But we know that at least one alternative can be routed yielding an optimal solution to MkSF.

Not surprisingly, the situation gets more difficult when k is no longer constant but part of the input. We prove that, for any fixed $\epsilon > 0$, there exists a $(1-\epsilon)$–approximate solution to MkSF that only uses constantly many flow values on paths. To be more precise, $|\{f_1, \ldots, f_k\}| \in O(\log(1/\epsilon)/\epsilon^2)$ for this solution. We believe that this result is also interesting for other flow problems (e.g., multicommodity flows etc.). As a result of this observation, we can "guess" the flow values used by a $(1-\epsilon)$–approximate solution to the MkSF problem while only increasing the running time of the subsequent routing procedure by a polynomial factor.

In Section 3 we consider the routing step on graphs of bounded treewidth. For constant k, the problem can be solved to optimality in polynomial time. Surprisingly, however, if k is part of the input, the MkSF problem is NP–hard on graphs of bounded treewidth. Based on our results from Section 2 and standard dynamic programming techniques, we obtain a polynomial–time approximation scheme (PTAS) in this case.

Finally, in Section 4 we classify the complexity and approximability of the MkSF problem for different values of $k \geq 2$ on directed and undirected graphs. In particular, we prove that the problem on undirected graphs is already NP–hard for $k = 2$. So far, NP–hardness was only known for the case of directed graphs. Moreover, we show that, for arbitrary constant k, the problem cannot be approximated with performance ratio better than $5/6$. The question whether MkSF is also NP–hard for "large" values of k, like for example $k = m/2$, has so far been open. We prove that the problem is NP–hard for all values of k within the range from 2 to $m - n + 1$ (for $n \geq 3$). For $k \geq m - n + 2$ the problem can be solved optimally in polynomial time.

Due to space limitations, we omit some proofs in this extended abstract. More details are given in the full version of the paper which can be found on the authors' homepages.

2 The packing stage

As mentioned in the introduction, we want to solve MkSF as a two–stage problem with a packing and a routing stage. Here, we consider the packing stage. Lemma 1 shows that, in order to solve the MkSF problem to optimality, it is not necessary to take all rational valued k–tuples $(f_1, ..., f_k)$ into account. It suffices to consider only $O(m^k)$ candidates of such tuples.

Lemma 1. *If k is constant, it is sufficient to consider $O(m^k)$ candidates to obtain a packing $(f_1, ..., f_k)$ of an optimal solution to* MkSF*. An appropriate set of candidates can be determined in $O(m^k)$ time.*

Proof. If we knew paths $P_1, \ldots, P_k$ used in an optimum solution to MkSF, then corresponding optimal path flow values $(f_1, \ldots, f_k)$ could be obtained by solving

the following linear program:

$$\begin{aligned} \max\ & f_1 + f_2 + ... + f_k & & \\ \text{s.t.}\ & \sum_{i\in\{1,...,k\}:\ e\in P_i} f_i \le u_e & & \text{for all } e \in E(G), \\ & f_i \ge 0 & & \text{for all } i \in \{1, ..., k\}. \end{aligned}$$

There exists an optimum solution to this linear program which corresponds to a vertex of the underlying polytope defined by the $m+k$ inequalities. Every vertex of this polytope is defined by a subsystem consisting of k linearly independent inequalities which must be tight for this vertex. The resulting system of k linear equations is given by a regular $\{0,1\}$–matrix of size $k \times k$ and a right hand side vector consisting of edge capacity values and zeros. Since the number of matrices in $\{0,1\}^{k\times k}$ is 2^{k^2} and the number of possible right hand side vectors is at most $(m+1)^k$, there are only $O(m^k)$ possible solutions to such equation systems. This yields $O(m^k)$ candidates for flow values $(f_1, ..., f_k)$ in an optimum solution to MkSF. Notice that each candidate can be computed in constant time by solving a system of linear equations of size $k \times k$. □

For constant k only a polynomial number of candidates has to be considered. If k is not constant but part of the input, however, the latter insight is not useful in obtaining efficient algorithms for MkSF since the number of candidate solutions is exponential in k and thus in the input size.

We can overcome this problem if, instead of looking for flow values $(f_1, ..., f_k)$ in an optimum solution, we settle for a near–optimum solution.

Assume that an optimum solution to MkSF assigns flow values $x_1, \ldots, x_k$ to paths $P_1, \ldots, P_k$. The following packing lemma shows that, for arbitrary $\epsilon > 0$, there exists a k–splittable flow of value at least $1 - \epsilon$ times the value of an optimum flow which uses only a constant number (depending on ϵ) of different flow values on paths.

Lemma 2. *Let $\epsilon > 0$ be sufficiently small. Consider an arbitrary collection of k bins with capacities $x_1, \ldots, x_k$. Then, there exist k items with sizes $y_1, \ldots, y_k$ such that*

(i) the items can be packed into the given bins without violating capacities,
(ii) there are at most $3\log(1/\epsilon)/\epsilon^2$ different item sizes, and
(iii) the total item size is close to the total bin capacity, that is,

$$\sum_{i=1}^{k} y_i \ \ge\ (1-4\epsilon)\sum_{i=1}^{k} x_i \ .$$

Interpret the item sizes $y_1, \ldots, y_k$ as flow values. Then, for each $j = 1, \ldots, k$, we can route flow of value y_j along path P_i where i is the bin which item j has been assigned to. The resulting k–splittable flow does not violate capacities due to (i) and its flow value is almost optimal due to (iii).

Proof. Let $X := \sum_{i=1}^{k} x_i$ denote the total bin capacity. We recursively define a partition of the set of bins into subsets $B_1, B_2, \ldots, B_\ell$ as follows. Consider the bins in order of non–increasing capacities. Add the first bin to B_1. Keep adding bins to B_1 as long as the total capacity of bins in B_1 is at most ϵX. The first bin which cannot be added to B_1 due to this restriction goes to B_2. The following bins are added to B_2 as long as the total capacity of bins in B_2 is at most ϵX and so on. Since, except for the last subset, the total capacity of bins in each subset is at least $\epsilon X/2$, the number of subsets obtained in this way is $\ell \leq 2/\epsilon$. Notice that the first few subsets may contain a single bin of size greater than ϵX. All further subsets contain bins whose total capacity is at most ϵX.

For all but at most three subsets of size at most ϵX, we will fill all bins i contained in these subsets with items of total volume at least $(1-\epsilon)x_i$. We shortly argue that such a packing fulfills property (iii): The total capacity of bins contained in the three neglected subsets is at most $3\epsilon X$. The remaining capacity of at least $(1-3\epsilon)X$ is filled up to at least a $(1-\epsilon)$–fraction. Thus, the total size of all items packed is at least $(1-\epsilon)(1-3\epsilon)X \geq (1-4\epsilon)X$, for $\epsilon > 0$.

Packing Phase I: For all subsets B_p whose largest bin capacity is within a factor $1/\epsilon$ of its smallest bin capacity, we pack one item into each bin in B_p using at most $1 + \log_{1+\epsilon}(1/\epsilon)$ different item sizes: Take the smallest bin in B_p and denote its capacity by z; pack an item of size z into all bins of capacity at most $(1+\epsilon)z$ in B_p. Remove all packed bins and continue recursively.

Packing Phase II: In order to simplify notation, the subsets that were not treated in phase I are re–indexed and denoted by $B'_1, \ldots, B'_{\ell'}$; the smallest bin in B'_j is at least as large as the largest bin in B'_{j+1}, for $j = 1, \ldots, \ell'-1$. The largest bin capacity in B'_j is denoted by z_j. We ignore all bins in $B'_{\ell'-2} \cup B'_{\ell'-1} \cup B'_{\ell'}$. For $j = 1, \ldots, \ell' - 3$, greedily pack all bins in B'_j using at most $|B'_{j+2}|$ items of size z_{j+2}.

It remains to prove that each packed bin is filled up to at least a fraction $1-\epsilon$ of its capacity. First notice that the capacity x_i of each bin $i \in B'_j$ is greater than z_{j+2}/ϵ. This is due to the fact that the ratio of the largest and smallest capacity of bins in B'_{j+1} is greater than $1/\epsilon$ (otherwise, subset B'_{j+1} would have been treated in phase I). Thus, if enough items of size z_{j+2} are available, each bin $i \in B'_j$ can be filled leaving a slack smaller than $z_{j+2} < \epsilon x_i$. In order to prove that enough items are available, it suffices to show that the total volume of $|B'_{j+2}| + 1$ items of size z_{j+2} exceeds the total capacity of all bins in B'_j:

$$\sum_{i \in B'_j} x_i \;\leq\; \epsilon X \;<\; \sum_{i \in B'_{j+2}} x_i + z_{j+2} \;\leq\; (|B'_{j+2}| + 1) \cdot z_{j+2} \;\; .$$

The number of different item sizes used in phase I and II is bounded by $\ell(1 + \log_{1+\epsilon}(1/\epsilon)) \leq 3\log(1/\epsilon)/\epsilon^2$ for ϵ small enough. Moreover, at most k items are used and the sizes of the remaining items can be set to zero. □

Corollary 1. *If the value of an optimum solution to* MkSF *is known, flow values together with multiplicities denoting the number of paths which carry these flow values used by a* $(1-\epsilon)$*–approximate solution can be obtained by testing* $(\frac{k}{\epsilon})^{O(\log(1/\epsilon)/\epsilon^2)}$ *candidates.*

Proof. We denote the value of an optimum solution by OPT. As discussed above, it follows from Lemma 2 that there exists a $(1-\epsilon)$–approximate solution which uses $O(\log(1/\epsilon)/\epsilon^2)$ different flow values. If we round down all flow values to multiples of $\epsilon OPT/k$, we lose another factor of at most $1-\epsilon$ in the flow value. The resulting flow is therefore still $(1-2\epsilon)$–approximate and uses $O(\log(1/\epsilon)/\epsilon^2)$ out of k/ϵ possible flow values. These flow values can therefore be guessed by trying all $(\frac{k}{\epsilon})^{O(\log(1/\epsilon)/\epsilon^2)}$ possible alternatives. For each fixed alternative, we have to assign a number to each flow value of this alternative which determines the number of paths carrying this flow value. For each alternative, the number of different assignments is bounded by $k^{O(\log(1/\epsilon)/\epsilon^2)}$. Thus, we have to test $(\frac{k}{\epsilon})^{O(\log(1/\epsilon)/\epsilon^2)}$ candidates. □

Notice that one can get rid of the assumption that the value of an optimum solution is known. Using standard binary search, OPT can be determined within a factor of $1-\epsilon$ while increasing the running time of the embedded algorithm only by a polynomial factor.

3 The routing stage in graphs of bounded treewidth

In this section we consider the MkSF problem on graphs of bounded treewidth, a graphclass introduced by Robertson and Seymour [**?**]. For constant k, we present a polynomial time algorithm for MkSF. For arbitrary k, the problem remains NP–hard even if restricted to graphs of bounded treewidth (with only three nodes and two sets of parallel arcs). We give a polynomial time approximation scheme for the general MkSF problem on graphs of bounded treewidth.

Theorem 1. *On graphs of bounded treewidth, the* MkSF *problem can be solved in polynomial time if k is constant. For arbitrary k, the problem is NP–hard and there exists a polynomial time approximation scheme.*

3.1 Preliminaries on graphs of bounded treewidth

Given a graph $G=(V,E)$ (directed or undirected), a *tree decomposition* is a pair (T,χ) where T is a tree and $\chi=\{X_i|X_i\subseteq V, i\in V(T)\}$ is a family of subsets of V associated with the nodes of T such that the following conditions hold: (i) Each node of G is contained in a subset X_i for some $i\in V(T)$. (ii) For each edge in G there exists a node $i\in V(T)$ such that X_i contains both endpoints of that edge. (iii) For each node $u\in V(G)$, the vertices $i\in V(T)$ with $u\in X_i$ span a subtree of T.

The width of a tree decomposition (T,χ) is $\max_{i\in V(T)}|X_i|-1$. The treewidth of a graph G is the minimum width over all tree decompositions of G. Given as input a graph G and an integer ω, it is NP–complete to decide if G has treewidth at most ω; see [**?**]. On the other hand, if the treewidth of G is bounded by a fixed constant, a decomposition tree can be constructed in linear time [**?**].

We can restrict to tree decompositions featuring a special structure: A tree decomposition (T,χ) of G is called *nice* if T is a rooted binary tree and if the

nodes partition into four types: A *join node* $i \in V(T)$ has two children $j, h \in V(T)$ fulfilling $X_i = X_j = X_h$. An *introduce node* $i \in V(T)$ has only one child j and that child fulfills $X_j \subset X_i$. A *forget node* $i \in V(T)$ has only one child $j \in V(T)$ and that child fulfills $X_j = X_i \cup \{u\}$ for some $u \in V(G) \setminus X_i$. Finally, for a *leaf node* $i \in V(T)$, the set X_i consists of some node $u \in V(G)$ together with a subset of its neighborhood. Furthermore, in a nice tree decomposition, there is a leaf containing u and v, for each edge $(u, v) \in E(G)$. For a given graph G, a tree decomposition can be transformed into a nice tree decomposition of the same width in linear time with tree size $O(|V(G)|)$; see, e.g., [**?**].

3.2 The algorithm

In the following description of the algorithm we restrict to the case of simple (directed) graphs without parallel edges. Without going into further details, we remark that Theorem 1 also holds for non–simple graphs. As a result of Section 2, a polynomial time algorithm for the following problem on graphs of bounded treewidth will prove Theorem 1.

Given: Directed or undirected graph $G = (V, E)$ with edge capacities $u : E \to \mathbb{Q}_{>0}$, a source $s \in V$, and a sink $t \in V$; a constant number of flow values $f_1, \ldots, f_\ell \in \mathbb{Q}_{>0}$ together with multiplicities $q_1, \ldots, q_\ell \in \mathbb{N}$ which are polynomially bounded in the size of G.

Task: For $j = 1, \ldots, \ell$, find q_j paths (not necessarily distinct) from s to t such that sending f_j flow units along each path (simultaneously for all j) does not violate edge capacities; alternatively, decide that no such flow exists.

Theorem 2. *On graphs of constantly bounded treewidth, the problem stated above can be solved in polynomial time. Moreover, if the multiplicities q_j, $j = 1, \ldots, \ell$, are all constant, it can be solved in linear time.*

For the sake of simplicity, we reduce the flow problem to a circulation problem by introducing a new edge from t to s with sufficiently large capacity (notice that adding an edge to a graph increases its treewidth by at most one). Now the problem can be reformulated as follows: Find a feasible circulation in the extended graph which fulfills the additional requirement that, for $j = 1, \ldots, \ell$, exactly q_j cycles (not necessarily distinct) each carrying flow value f_j have to traverse the special edge from t to s.

Algorithms exploiting bounded treewidth of the input graph are usually based on a dynamic programming approach that proceeds bottom–up in the decomposition tree. Our algorithm follows along the same line. For a general description of this approach we refer to [**?**].

The rough idea of the algorithm is as follows. Each edge of graph G is associated with exactly one leaf of T containing its two endpoints. For a tree node $i \in V(T)$ we denote by $G_i = (V_i, E_i)$ the subgraph of G given by

$$V_i := \{v \in V(G) \mid v \in X_h \text{ with } h = i \text{ or } h \text{ is descendant of } i \text{ in } T\} \quad \text{and}$$
$$E_i := \{e \in E(G) \mid e \text{ is associated with } i \text{ or a descendant of } i \text{ in } T\} \ .$$

For every tree node $i \in V(T)$, we determine all possible ways of sending flow in graph G_i on X_i–paths. (A path is called an X_i–*path* if its ends are distinct vertices in X_i and no internal vertex belongs to X_i.) To be more precise, each possible *state* of node i is specified by the following information: For every ordered pair of distinct vertices $(u, v) \in X_i \times X_i$ and for every $j \in \{1, \ldots, \ell\}$, we give the number $\pi(u, v, j) \leq q_j$ of (not necessarily different) X_i–paths between u and v carrying f_j units of flow.

Notice that the number of possible states at node i is at most $\prod_{j=1}^{\ell}(1+q_j)^{|X_i|^2}$ and thus polynomially bounded. Of course, we are only interested in states/flows that can be realized without violating edge capacities. Moreover, if the special edge from t to s is contained in G_i, we only consider flows where the number of X_i–paths of flow value f_j using that edge is exactly q_j, for $j = 1, \ldots, \ell$. These two requirements are taken care of when computing the set of feasible states at the leaf nodes of T. In the following we give an overview of how the required information can be computed at the nodes i of T. Since we basically follow a standard approach, further details are omitted.

If i is a *leaf node*, G_i contains only a constant number of edges. For each such edge $(u, v) \in E_i$, we generate all possible configurations $\pi(u, v, j)$, $j = 1, \ldots, \ell$, that do not violate the capacity of that edge. Of course, if the edge happens to be the special one from t to s, only the unique feasible configuration is generated. By taking all possible combinations of configurations at the edges, we get the set of all states of node i.

If i is an *introduce node*, the set of all states of i is identical to the set of all states of its only child i'. Notice that no flow can be sent from or received by terminals in $X_i \setminus X_{i'}$ since no edge in G_i is incident with one of these terminals.

If i is a *forget node*, the set of all states of i can be obtained from the set of all states of its only child i' as follows: Delete all states of i' that do not fulfill flow conservation at the unique node $u \in X_{i'} \setminus X_i$ separately for every $j = 1, \ldots, \ell$. For the remaining states, generate all possible matchings of incoming and outgoing flow paths of the same flow value at node u. This yields all possible flow patterns between terminals $X_i = X_{i'} \setminus \{u\}$.

Finally, if i is a *join node*, every feasible state of i can be generated by adding two states, one from each child node. Of course, we only consider sums for which $\pi(u, v, j) \leq q_j$, for all $u, v \in X_i$ and $j = 1, \ldots, \ell$.

As it is always the case with this approach, the answer to the problem which we like to solve can be found at the root node r of tree T: There exists a feasible solution if and only if r has a state where flow conservation is fulfilled at all nodes in X_r, separately for every $j = 1, \ldots, \ell$. A solution (circulation) can be obtained by traversing the tree forwards to the leafs beginning with a feasible state at r. We omit all further details.

We conclude this section with a generalization of the obtained result. Notice that the described approach can also be applied if, instead of only one source s and one sink t, there is a constant number of source–sink pairs (commodities).

Corollary 2. *For a constant number of commodities and constant k, the k–splittable multicommodity problem can be solved in polynomial time on graphs*

of bounded treewidth. If we drop the requirement on k, we still obtain a polynomial time approximation scheme for the maximum k–splittable multicommodity problem with a fixed number of commodities.

4 Complexity and approximability for general graphs

In this Section we return to general graphs. We analyze the hardness of MkSF problems and their approximability. In the first part, we consider constant values of $k \geq 2$. MkSF is shown to be strongly NP–hard. We show that there is no approximation algorithm with performance ratio better than $\frac{5}{6}$. This is the first constant bound given for this problem. In the second part, k is a function of the number of vertices n and edges m. We classify NP–hard and polynomially solvable cases. For the sake of simplicity we restrict ourselves to undirected graphs, but any result in this section can be applied to the directed case by minor modifications in the proofs.

4.1 Constant k

In [**?**] the NP–hardness of MkSF is proven for constant $k \geq 2$ in directed graphs. The construction given there does not apply to undirected graphs. Theorem 3 shows that the NP–hardness also holds for the undirected case. Furthermore, a new construction in the proof enables us to derive two bounds on the approximability. To simplify notation, we denote the problem MkSF with $k = 2$ by M2SF, as well for other values of k.

Theorem 3. *For all constant $k \geq 2$,* MkSF *is strongly NP–hard and cannot be approximated with performance guarantee better than $\frac{k}{k+1}$, unless $P = NP$.*

Proof. First we give a reduction from 3SAT to M2SF and show that a satisfiable instance of 3SAT yields an optimum solution of value 3 whereas a nonsatisfiable instance yields an optimum solution of value 2 for the corresponding M2SF–instance. Later we extend the reduction to any constant $k \geq 2$.

Consider a 3SAT–instance with variables $x_1, ..., x_r$ and clauses $C_1, ..., C_q$. In the following we construct the corresponding M2SF–instance in two steps illustrated in Figures 1 and 2.

Step 1, Figure 1 (top): The graph constructed in this step represents the clauses of the 3SAT–instance. Introduce two nodes s and t and two nodes a_j, b_j for every clause C_j. For every litaral of C_j we construct an a_j, b_j–path, which we initialize with one edge $\{a_j, b_j\}$. These paths will be expanded in Step 2. Connect the clause representations by the $q+1$ edges $\{s, a_1\}, \{b_1, a_2\}, \{b_2, a_3\}, ..., \{b_q, t\}$. All edges created in this step get capacity 1. The construction so far allows s, t–paths traversing each clause along one path representing one literal of the clause. To control that such s, t–paths do not use paths that belong to contrary literals we introduce a blocking construction in Step 2.

Step 2, Figure 1 (bottom): Assume that there are h pairs of contrary literals x_i and $\bar{x}_i$, which belong to different clauses. Consider the l-th pair and assume that x_i appears in a clause C and $\bar{x}_i$ in a clause C'. Insert one edge $\{y_l, z_l\}$ into an edge $\{u, v\}$ of unit capacity of the path representing x_i. The new edges $\{u, y_l\}$ and $\{z_l, v\}$ get a capacity of 1 and the edge $\{y_l, z_l\}$ gets a capacity of 2. Analogously, insert an edge $\{y'_l, z'_l\}$ into an edge $\{u', v'\}$ of unit capacity of the path representing $\bar{x}_i$. Introduce two nodes c_l and d_l and edges $\{c_l, y_l\}, \{d_l, z_l\}, \{c_l, y'_l), \{d_l, z'_l\}$ with capacities 2 to get a blocking construction for the l-th pair of contrary literals. To complete the construction we add edges $\{s, c_1\}, \{d_1, c_2\}, \{d_2, c_3\}, ..., \{d_{h-1}, c_h\}, \{d_h, t\}$, also with capacities 2.

Figure 2 shows the entire construction for an example instance. Notice, that this reduction is of polynomial size because the number of nodes is at most quadratic in the number q of clauses and the maximum degree of a node is 4. Furthermore, any s, t–flow has a value less than or equal to 3, because the edges incident to s have together a capacity of 3. Next we show that any 2–splittable flow with a value greater than 2 implies the satisfiability of the 3SAT–instance.

Let us consider two s, t–paths which together carry a flow of value greater than 2. So there is a path P_1 with flow value greater than 1, which therefore can only use edges of capacity 2. Such edges only occure in the blocking construction of contrary literals and because of the structure of the graph P_1 must traverse all these constructions. The second path P_2 must be disjoint from P_1 because all edge capacities are bounded by 2. So it has to traverse all clause representations constructed in Step 1. While traversing the clauses, P_1 never sends flow along paths representing contrary literals simultaneously because P_2 blocks at least one of them. Refering to the 3SAT instance, set $x_i := 1$ if P_2 traverses an a_j, b_j–path representing x_i in one arbritrary clause C_j and otherwise set $x_i := 0$. Then, every variable is set to 0 or 1 and every clause contains one true literal. So we have described a satisfying assignment for the 3SAT–instance.

On the other hand, every satisfiable 3SAT instance implies a 2–splittable flow of value 3. Choose one satisfied literal for each clause in a satisfying assignment. Route one unit of flow along a path P_2 traversing the representations of the clauses always along the path of the chosen literal. Afterwards, we send two units of flow along the blocking constructions using one s, t–path P_1. This is possible because P_2 never traverses paths of contrary literals simultaneously. We get a 2–splittable s, t–flow of value 3.

Thus, 3SAT can be reduced to M2SF and a 3SAT–instance is satisfiable if and only if a maximum 2–splittable flow has value 3 and is not satisfiable if and only if the maximum value is 2.

To extend the reduction to all constant $k \geq 2$ we add $k - 2$ s, t–edges with capacity 1. Then a 3SAT instance is satisfiable if and only if a k–splittable flow has a maximum value of $k + 1$ and is not satisfiable if and only if the maximum value is k. So MkSF is strongly NP–hard (because of the NP–hardness of 3SAT an the constantly bounded capacities in the reduction) and cannot be approximated with guarantee better than $\frac{k}{k+1}$, unless $P = NP$. □

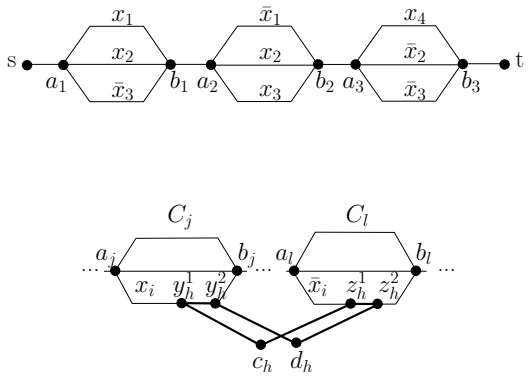

Fig. 1. Step 1 (top) and step 2 (bottom) for the 3SAT instance $x_1 \vee x_2 \vee \bar{x}_3$, $\bar{x}_1 \vee x_2 \vee x_3$, $\bar{x}_2 \vee \bar{x}_3 \vee x_4$

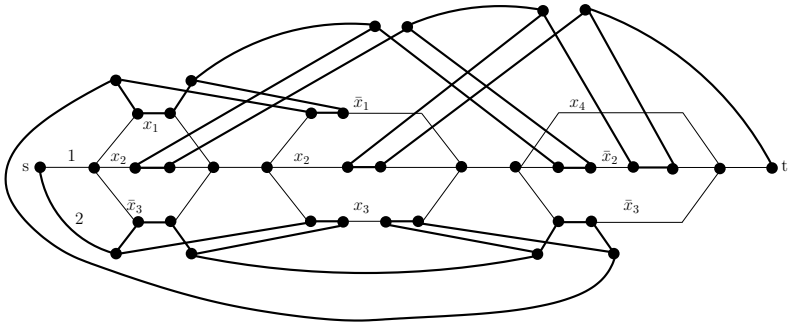

Fig. 2. Entire construction for the 3SAT instance $x_1 \vee x_2 \vee \bar{x}_3$, $\bar{x}_1 \vee x_2 \vee x_3$, $\bar{x}_2 \vee \bar{x}_3 \vee x_4$

Corollary 3. MkSF, $k \geq 2$, *cannot be approximated with performance guarantee better than* $\frac{5}{6}$, *unless* $P = NP$. *(Proof omitted.)*

4.2 k as a function of m and n

Here, we consider k as a function of the number of edges m and of the number of nodes n of a graph G. Note, that k is not seen as a part of the input, but a property of the problem MkSF. Thus, for different functions k we consider different problems. Some functions result in polynomially solvable problems.

Lemma 3. *For a graph* G, MkSF *with* $m - n + 2 \leq k$ *is polynomial solvable.*

Proof. We show, that any maximum s,t–flow f in G can be decomposed into at most $m - n + 2$ path and cycles in polynomial time. Consider an orientation of the edges of G such that f is still a feasible flow and add an edge (t, s) of infinite capacity to obtain a directed graph G'. Setting the flow on the edge (t, s) to

value(f) results in a circulation f' in G'. Each decomposition of f' in cycles easily yields a decomposition of f in paths and cycles with the same number of elements.

We compute a decomposition of f' with the standard decomposition algorithm of Fulkerson. That means, start with any flow carrying edge and go through G' only using edges with a positive amount of flow until a cycle is closed. Assign the maximal possible flow value to this cycle with respect to f' and reduce f' by this cycle flow. Repeat this procedure until $f' = 0$. Since in any iteration the flow value of at least one edge is decreased to 0 the incidence vectors of these cycles are linearly independent. Furthermore, the cycle space of G' has a dimension of $m+1-n+1 = m-n+2$ such that the computed decomposition of f' contains no more than $m-n+2$ cycles. □

In the following, we show that MkSF is NP–hard for all k with $2 \leq k \leq m-n+1$. This is done in two steps. We prove the NP–hardness for $2 \leq k \leq m-m^\epsilon$ by a reduction from 3SAT and then for $m^\epsilon \leq k \leq m-n+1$ by a reduction from SubsetSum. In both cases $\epsilon \in (0,1)$.

Theorem 4. *For all constant $\epsilon \in (0,1)$ MkSF with $2 \leq k \leq m-m^\epsilon$ is strongly NP–hard and cannot be approximated with a guarantee better than $\frac{k}{k+1}$, unless $P = NP$.*

Proof. Given $\epsilon \in (0,1)$ we reduce 3SAT to MkSF with a k arbitrary in the range $2 \leq k \leq m-m^\epsilon$.

According to Theorem 3 it suffice to show that the graph G consisting of the graph G' shown in Figure 2 together with $k-2$ additional s,t–edges from is of polynomial size in relation to the size of the considered 3SAT instance. Let m' be the number of edges of G'. Then we have $m = k-2+m'$ and it follows:

$$m \leq m - m^\epsilon - 2 + m' \quad \Rightarrow \quad m^\epsilon \leq m' - 2 \quad \Rightarrow \quad m \leq (m'-2)^{\frac{1}{\epsilon}}.$$

Thus, m is polynomial in m' and because m' is polynomially bounded it holds also for G. □

Theorem 5. MkSF *with $k = m-n+1$ is NP–hard for every given $n > 2$. (Proof omitted.)*

Corollary 4. *Given $\epsilon \in (0,1)$ MkSF with $m^\epsilon \leq k \leq m-n+1$ is NP–hard for every given $n > 2$. (Proof omitted.)*

Corollary 5. MkSF *with $2 \leq k \leq m-n+1$ is NP–hard for all graphs with $n > 2$.*

Proof. Choose $\epsilon := \frac{1}{3}$. Theorem 4 proves the NP–hardness of MkSF for $2 \leq k \leq m-m^{1/3}$. Corollary 4 shows the NP–hardness for $m^{1/3} \leq k \leq m-n+1$. Since $m^{1/3} \leq m-m^{1/3}$ for $m \geq 3$ the bounds overlap what proves the corollary. ($m \leq 2$ does not allow any k here) □