Maximum k–Splittable s, t-Flows

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

[1] Universität Dortmund, Fachbereich Mathematik, 44221 Dortmund, Germany
{ronald.koch, martin.skutella}@math.uni--dortmund.de
[2] Technische Universität Berlin, Institut für Mathematik, 10623 Berlin, Germany
spenke@math.tu-berlin.de

Abstract. Given a graph with a source and a sink node, the NP–hard maximum k–splittable s, t–flow (MkSF) problem is to find a flow of maximum value from s to t 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 for the 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. Usually the routing is considered before the packing. 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.

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

⋆ Supported in part by the Deutsche Forschungsgemeinschaft (DFG) as part of the DFG Focus Program 1126 "Algorithmic Aspects of Large and Complex Networks", and as part of the Collaborative Research Center "Computational Intelligence" (SFB 531).

⋆⋆ Supported by DFG Research Center Matheon "Mathematics for key technologies"

matter how many paths the flow uses. 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 and capacities $u : E \to \mathbb{Q}_{\geq 0}$. Moreover, there is a source node and a sink node $s, t \in V$. Baier, Köhler, and Skutella [7] 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 using at most k paths from s to t. We do not require the paths to be disjoint, not even different. The Maximum k–Splittable Flow problem (MkSF) consists in finding 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 [12], 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 [1].

Kleinberg [14] introduces *unsplittable flows.* These multicommodity flows route the total demand of each commodity along one single path. They generalize edge–disjoint paths. Kleinberg analyzes 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 [7] (see also [6]) 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 $O(km \log n)$-time algorithm that computes an optimum solution. Based on these insights, they present $\frac{1}{2}$–approximation algorithms for the general MkSF problem.

Koch and Spenke [15] study the complexity and approximability of the MkSF problem for different values of $k \geq 2$ on directed and undirected graphs. In particular, it is proven that the problem is already NP–hard for $k = 2$ on directed and undirected graphs. It is shown that, for arbitrary constant k, the problem cannot be approximated with a performance ratio better than $5/6$. For k being a function of graph parameters MkSF is proven to be NP–hard for all values of k within the range from 2 to $m - n + 1$ (for $n \geq 3$) whereas it is polynomially solvable for all $k \geq m - n + 2$.

Bagchi, Chaudhary, Kolman, and Scheideler [5] 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 [17] 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. Martens and Skutella [18] give approximation algorithms for length-bounded k-splittable flows and for dynamic k-splittable flows. Bley [8] analyzes the related problem of finding k vertex-disjoint s,t-paths of minimal total weight in a weighted network. Bagchi, Chaudhary, and Kolman [4] consider a generalized problem where each given pair of nodes must be connected by k edge disjoint paths of the same color. The objective is to minimize the number of colors such that two paths of the same color are edge disjoint.

Krysta, Sanders, and Vöcking [16] 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 [19] in the context of graph minors, these graphs are also relevant in several practical applications. Bodlaender [11] presents a general framework for obtaining polynomial algorithms for problems in graphs of bounded treewidth that are NP–hard in general graphs. Bodlaender [9] and Arnborg, Lagergren, and Seese [3] give general characterizations of problems that can be solved in polynomial time on graphs of bounded treewidth. The MkSF problem does not fall into any 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 [11].

The only paper we are aware of that considers flows in graphs of bounded treewidth is the one by Hagerup et al. [13]. 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 Contribution. Constructing a flow requires two decisions. One has to decide which paths should be used and what flow values should be assigned to these paths. Of course, these two decisions cannot be made independently of each other. They are coupled by the requirement to obey arc capacities. A natural approach could be to choose first a collection of paths $(P_1, \ldots, P_k)$ that should be used and then to decide about flow values $(f_1, \ldots, f_k)$ for these paths. In this paper we take the reverse approach. We first fix several candidates for 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 value patterns $(f_1, \ldots, f_k)$ 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. We believe that this result is also interesting for other flow problems (e.g., multicommodity flows). 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. 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.

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 optimally, 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. This is polynomial if k is constant.

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:

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

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 are 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 bounded by $(m+1)^k$, 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 MkSF solution 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 the paths.

Lemma 2. *Let $\epsilon > 0$ be sufficiently small. Consider a 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,

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

Interpret the item sizes $y_1, y_2, \ldots, y_k$ 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 to which item j has been assigned. 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. Notice that all subsets of size greater than ϵX are processed in this phase, such that only subsets of size at most ϵX remain for Phase II.

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:

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

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

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 path flow values. If we round down all the path 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 $(f_1, \ldots, f_k)$. $\square$

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 graph class introduced by Robertson and Seymour [19]. For constant k, MkSF turns out to be polynomially solvable if restricted to this graph class. We present a polynomial time algorithm for MkSF. For k being part of the input, the problem remains NP–hard even if restricted to graphs of bounded treewidth. This can be seen by a reduction from SubsetSum with only three nodes and two sets of parallel arcs; a proof is given by Koch and Spenke [15]. Here, we give a polynomial time approximation scheme for the MkSF problem on graphs of bounded treewidth.

Theorem 1. *On graphs of bounded treewidth, the* MkSF *problem is solvable in polynomial time if k is constant. For arbitrary k, 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 a graph G and an integer ω, it is NP–complete to decide if G has treewidth at most ω [2]. On the other hand, if the treewidth of G is bounded by a fixed constant, a decomposition tree can be constructed in linear time [10]. We can restrict to tree decompositions featuring a special structure. A tree decomposition (T, χ) of G is called *nice* if:

(i) T is a rooted binary tree.
(ii) There are four types of nodes: join nodes, introduce nodes, forget nodes, and leaves. 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 $X_j = X_i \cup \{u\}$ for some $u \in V(G) \setminus X_i$. The set X_i of a *leaf* $i \in V(T)$ consists of some node $u \in V(G)$ together with a subset of its neighborhood.
(iii) 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 $O(|V(G)|)$ [20].

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 Sect. 2, a polynomial time algorithm for the following problem on graphs of bounded treewidth will prove Theorem 1.

Input: 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 $q_1, \ldots, q_\ell \in \mathbb{N}$ that 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 or decide that no such flow exists.

Theorem 2. *On graphs with treewidth bounded by a constant, the above stated problem is solvable 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: In the extended graph, find a feasible circulation which fulfills the following additional requirement: for $j = 1, \ldots, \ell$, exactly q_j cycles (not necessarily distinct) each carrying flow value f_j 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 [11].

The rough idea of the algorithm is as follows. Each edge of the 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 a 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 specify 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 does not exceed $\prod_{j=1}^{\ell}(1 + q_j)^{|X_i|^2}$ and thus is polynomially bounded. Of course, we are only interested in states (i. e., 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.

If i is a *leaf*, 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 any 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 that we want to solve is 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 leaves beginning with a feasible state at r.

Finally, we mention that the approach and the result remain true for a constant number of commodities not only for one source–sink pair. Notice that the dynamic program discussed above can be generalized to this case while only increasing the state space and the time of a bottom-up step by a constant factor.

Theorem 3. *For a constant number of commodities and constant k, the k–splittable multicommodity problem is solvable in polynomial time on graphs of bounded treewidth. For arbitrary k, there exists a polynomial time approximation scheme for the maximum k–splittable multicommodity problem with a fixed number of commodities.*

Acknowledgement

We thank two anonymous referees for many useful comments that helped to improve the presentation of the paper.

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows. Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.
2. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.
3. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.
4. A. Bagchi, A. Chaudhary, and P. Kolman. Short length menger's theorem and reliable optical routing. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures (SPAA03)*, pages 246–247. ACM Press, 2003.
5. A. Bagchi, A. Chaudhary, P. Kolman, and C. Scheideler. Algorithms for fault-tolerant routing in circuit-switched networks. In *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 265–274. ACM Press, 2002.
6. G. Baier. *Flows with Path Restrictions*. PhD thesis, TU Berlin, 2003.
7. G. Baier, E. Köhler, and M. Skutella. On the k-splittable flow problem. *Algorithmica*, 42:231–248, 2005.
8. A. Bley. On the complexity of vertex-disjoint length-restricted path problems. *Computational Complexity*, 12(3–4):131–149, 2004.
9. H. L. Bodlaender. Dynamic programming on graphs of bounded treewidth. In *Proceedings 15th International Colloquium on Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 105–118. Springer, Berlin, 1988.
10. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
11. H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In I. Privara and P. Ruzicka, editors, *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36. Springer, Berlin, 1997.

12. L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.
13. T. Hagerup, J. Katajainen, N. Nishimura, and P. Ragde. Characterizations of k-terminal flow networks and computing network flows in partial k-trees. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 641–649, 1995.
14. J. M. Kleinberg. *Approximation algorithms for disjoint paths problems.* PhD thesis, M.I.T., 1996.
15. R. Koch and I. Spenke. Complexity and approximability of k-splittable flows. *Theoretical Computer Science*, 369:338–347, 2006.
16. P. Krysta, P. Sanders, and B. Vöcking. Scheduling and traffic allocation for tasks with bounded splittability. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science*, volume 2747, pages 500–510. Springer, Berlin, 2003.
17. M. Martens and M. Skutella. Flows on few paths: Algorithms and lower bounds. In S. Albers and T. Radzik, editors, *Algorithms — ESA '04*, volume 3221 of *Lecture Notes in Computer Science*, pages 520–531. Springer, Berlin, 2004.
18. M. Martens and M. Skutella. Length-bounded and dynamic k-splittable flows. In *Proceedings of the International Scientific Annual Conference on Operations Research 2005*, pages 297–302. Springer, 2006.
19. N. Robertson and P. D. Seymour. Graph minors. II. algorithmic aspects of treewidth. *Journal of Algorithms*, 7:309–322, 1986.
20. P. Scheffler. A practical linear time algorithm for disjoint paths in graphs with bounded tree–width. Technical Report 396, TU Berlin, Fachbereich 3 Mathematik, 1994.