Algorithmica (2005) 42: 231–248
DOI: 10.1007/s00453-005-1167-9

Algorithmica

The *k*-Splittable Flow Problem[1]

Georg Baier,[2] Ekkehard Köhler,[2] and Martin Skutella[3]

Abstract. In traditional multi-commodity flow theory, the task is to send a certain amount of each commodity from its start to its target node, subject to capacity constraints on the edges. However, no restriction is imposed on the number of paths used for delivering each commodity; it is thus feasible to spread the flow over a large number of different paths. Motivated by routing problems arising in real-life applications, e.g., telecommunication, unsplittable flows have moved into the focus of research. Here, the demand of each commodity may not be split but has to be sent along a single path.

In this paper a generalization of this problem is studied. In the considered flow model, a commodity can be split into a bounded number of chunks which can then be routed on different paths. In contrast to classical (splittable) flows and unsplittable flows, the single-commodity case of this problem is already NP-hard and even hard to approximate. We present approximation algorithms for the single- and multi-commodity case and point out strong connections to unsplittable flows. Moreover, results on the hardness of approximation are presented. In particular, we show that some of our approximation results are in fact best possible, unless P = NP.

Key Words. Network flow, Approximation algorithm, Max-flow min-cut, Unsplittable flow.

1. Introduction.

The *k*-splittable flow problem is!a multi-commodity flow problem in which each commodity may be shipped only on a restricted number of different paths. The number of possible paths can be the same for all commodities or it may depend on the particular commodity. Problems of this kind occur, for instance, in communication networks: Customers request connections of given capacities between certain pairs of terminals in the network. If these capacities are large, it might be impossible for the network administrator to realize them unsplittably, that is, on single paths without increasing network capacity. On the other hand, the customer might not want to handle many connections of small capacity. Thus, one has to find a flow fulfilling the different customer demands and restrictions on connecting lines while respecting all capacity constraints.

Similar problems appear in transportation logistics: A number of different commodities has to be delivered to various destinations by means of trains (or ships). The possible train connections define a network with capacities on the arcs. In order to convey a commodity, it has to be packed into special containers which can then be loaded on trains.

[1] This work was supported in part by the EU Thematic Network APPOL I+II, Approximation and Online Algorithms, IST-1999-14084, IST-2001-30012, and by the Bundesministerium für Bildung und Forschung (bmb+f), Grant No. 03-MOM4B1.
[2] Fakultät II—Mathematik und Naturwissenschaften, Institut für Mathematik, Sekr. MA 6–1, Technische Universität Berlin, Straße des 17. Juni 136, D-10623 Berlin, Germany. {baier,ekoehler}@math.tu-berlin.de.
[3] Fachbereich Mathematik, Universität Dortmund, D-44221 Dortmund, Germany. martin.skutella@uni-dortmund.de.

Received March 3, 2003; revised October 23, 2003. Communicated by R. Möhring.
Online publication May 2, 2005.

In this context, it seems to be natural to impose bounds on the number of containers available for each commodity. Speaking in flow terminology, we thus have to bound the number of paths used for delivering a commodity.

Problem definition and notation. An instance of the *k-splittable flow problem* is defined on a directed or undirected graph $G = (V, E)$ with edge capacities $u_e \in \mathbb{Z}_{>0}$, $e \in E$. For arbitrary source and target nodes $s, t \in V$, let $\mathcal{P}_{s,t}$ denote the set of simple s-t-paths in G. Then a *k-splittable s-t-flow F* is specified by k pairs $(P_1, f_1), \dots, (P_k, f_k) \in \mathcal{P}_{s,t} \times \mathbb{R}_{\geq 0}$ of s-t-paths P_i and flow values f_i. We do not require that the paths $P_1, \dots, P_k$ are distinct or edge-disjoint. In particular, any k-splittable flow is also k'-splittable for all $k' \geq k$. The sum $\sum_{i=1}^{k} f_i$ is called the *s-t-flow value of F*. The flow F is *feasible* if it respects edge capacities, that is, for each edge $e \in E$ the sum of flow values on paths containing this edge must be bounded by its capacity u_e. The *maximum k-splittable s-t-flow problem* asks for a feasible k-splittable s-t-flow of maximal value.

Obviously, k-splittable s-t-flows form a special class of s-t-flows in graphs. In fact, it is a well-known result from classical network flow theory that any s-t-flow can be decomposed into the sum of at most $|E|$ flows on s-t-paths and a circulation; apart from the circulation, it is thus an $|E|$-splittable s-t-flow in our terminology. In particular, for $k \geq |E|$ the maximum k-splittable s-t-flow problem can be solved efficiently by standard network flow techniques; see, e.g., [1]. On the other hand, we show that the problem for directed graphs is NP-hard for $k = 2$. All our results except for hardness hold for directed and undirected graphs but for simpler notation we concentrate on the directed case.

We also study k-splittable s-t-flows with additional restrictions. A k-splittable s-t-flow in which all paths with non-zero flow value carry identical amounts of flow is called a *uniform k-splittable s-t-flow*. Moreover, a uniform k-splittable s-t-flow which sends flow on exactly k paths is called a *uniform exactly-k-splittable s-t-flow*. For our algorithms it is essential that we do *not* require k distinct paths. If there are also edge costs $c_e \in \mathbb{Q}_{\geq 0}$, $e \in E$, we can consider the problem with an additional budget constraint: Find a maximum k-splittable s-t-flow whose cost does not exceed a given budget $B \geq 0$. A closely related problem is the minimum-cost maximum k-splittable s-t-flow problem: Find a k-splittable s-t-flow of maximal value such that the cost of the flow is minimized. In this setting, let $c_P = \sum_{e \in P} c_e$ denote the cost of path P. Then the cost of a k-splittable s-t-flow $(P_1, f_1), \dots, (P_k, f_k)$ can be written as $\sum_{i=1}^{k} f_i c_{P_i}$, which is equal to the sum over all edges of edge cost times flow value on the edge.

In the multi-commodity variant of the k-splittable flow problem, there are ℓ terminal pairs $(s_1, t_1), \dots, (s_\ell, t_\ell)$ of source and target nodes, and a bound k_i on the number of paths allowed for each terminal pair (s_i, t_i), $i = 1, \dots, \ell$. A *k-splittable multi-commodity flow* (or simply *k-splittable flow*) F is the sum of k_i-splittable s_i-t_i-flows, for $i = 1, \dots, \ell$. If all k_i-splittable s_i-t_i-flows are uniform exactly-k_i-splittable s_i-t_i-flows, then F is called a *uniform exactly-k-splittable multi-commodity flow*. Notice that this definition allows different flow values per path for different commodities. For given demand values $D_1, \dots, D_\ell > 0$, the *k-splittable multi-commodity flow problem* (or simply *k-splittable flow problem*) is to find a feasible k-splittable flow F with s_i-t_i-flow value D_i, for $i = 1, \dots, \ell$. Here, the flow sent from s_i to t_i is also referred to as *commodity i*.

To the best of our knowledge, the k-splittable flow problem has not been considered in the literature before. However, it contains the well-known unsplittable flow problem as a special case: Setting $k_i = 1$ models the requirement that commodity i must be routed unsplittably, i.e., on a single s_i-t_i-path. The unsplittable flow problem has been introduced by Kleinberg [8], [9] as a generalization of the disjoint path problem. Kleinberg shows that it comprises several NP-complete problems from areas such as packing, partitioning, scheduling, load balancing, and virtual-circuit routing.

Kleinberg introduced several optimization versions of the unsplittable flow problem. In the "minimum congestion" version, the task is to find the smallest value $\lambda > 0$ such that there exists an unsplittable flow that uses at most a λ-fraction of the capacity of any edge. An equivalent problem is the "maximum concurrent flow" problem where the aim is to maximize the routable fraction of the demand, i.e., find the maximal factor by which the given demands can be multiplied such that there still exists a feasible unsplittable flow satisfying the resulting demands. Obviously, any solution to the minimum congestion problem of value λ can be turned into a solution to the maximum concurrent flow problem of value $1/\lambda$, and vice versa. The "minimum number of rounds" version asks for a partition of the set of commodities into a minimum number of subsets (rounds) and a feasible unsplittable flow for each subset. Finally, the "maximum routable demand" problem is to find a feasible unsplittable flow for a subset of demands maximizing the sum of demands in the subset. In this paper we only consider the maximum concurrent flow problem for multi-commodity k-splittable flows.

Since, apart from some special cases, all of these problems are NP-hard, much effort has been spent in obtaining approximation results. A *ρ-approximation algorithm* is an algorithm running in polynomial time and always finding a feasible solution whose value is at most a factor of ρ away from the optimum. The value ρ is called the *performance ratio* or *performance guarantee* of the algorithm. In the following we use the convention that the performance ratio for a maximization problem is less than 1, while for a minimization problem it is greater than 1. Thus, a ρ-approximation algorithm for a maximum concurrent flow problem computes a solution in polynomial time which realizes a fraction of at least $\rho \leq 1$ times the optimal fraction of the demands.

We mention some basic results and notation used in the remainder of the paper. A flow whose flow value on any edge is a multiple of a given value $\alpha > 0$ is called *α-integral*. If we restrict to *α-integral* flows, we can round down edge capacities to the nearest multiple of α without abandoning any solution. Thus, the problem of finding α-integral flows is equivalent to finding integral flows in graphs with integral capacities; it can therefore be solved efficiently in the single-commodity case by standard flow techniques.

Related results from the literature. Kleinberg [8], [9] considers a restricted version of the unsplittable flow problem where the maximal demand is bounded from above by the minimal edge capacity. This condition is usually referred to as the *balance condition*. Assuming that the balance condition holds, the randomized rounding technique of Raghavan and Thompson [17] can be applied to the problem of minimizing congestion, yielding an $\mathcal{O}(\log|E|)$-approximation algorithm and even a constant factor approximation if the optimal congestion is at least $\Omega(\log|E|)$. For densely embedded unit-capacity graphs (generalization of two-dimensional meshes), Kleinberg [8] gives an algorithm that finds a constant factor approximation with high probability. Aspnes et al. [2] present an asymptotically tight on-line $\mathcal{O}(\log|V|)$-competitive algorithm for the related virtual circuit routing problem.

For the maximum routable demand problem in unit-capacity graphs, Kleinberg [8] gives an approximation algorithm with performance ratio $\Omega((1-D_{\max})/\sqrt{|E|})$, where $D_{\max}$ denotes the maximal demand. Kolliopoulos and Stein [14] give the first non-trivial ratio $\Omega(1/(\log|E|\sqrt{|E|}))$ independent of the edge-capacities and $D_{\max}$, Baveja and Srinivasan [5] improve the ratio to $\Omega(1/\sqrt{|E|})$. Azar and Regev [3] give a strongly polynomial algorithm also with performance ratio $\Omega(1/\sqrt{|E|})$. On the other hand, Guruswami et al. [7] show that in the directed case there is no approximation algorithm with a performance ratio better than $|E|^{-1/2+\varepsilon}$ for any $\varepsilon > 0$, unless P = NP. Still, for densely embedded unit-capacity graphs, Kleinberg and Tardos [11] provide a constant factor approximation. To get better approximation results one has to incorporate additional graph parameters into the bound. Baveja and Srinivasan [5] develop a rounding techniques to convert an arbitrary solution to an LP-relaxation into an unsplittable flow within a factor of $\Omega(1/\sqrt{|E|})$ or $\Omega(1/d)$; here, d denotes the length of a longest path in the LP solution. Using a result of Kleinberg and Rubinfeld [10], showing that $d \in \mathcal{O}(\Delta^2\alpha^{-2}\log^3|V|)$ for uniform capacity graphs with some expansion parameter α and maximal degree Δ, one can achieve a better bound. In a recent work, Kolman and Scheideler [15] use a new graph parameter, the "flow number" φ, and improve the ratio further to $\Omega(1/\varphi)$ for undirected graphs; they show that $\varphi \in \mathcal{O}(\Delta\alpha^{-1}\log|V|)$.

Without the balance condition, Guruswami et al. [7] give an algorithm with performance ratio $\Omega(1/(\sqrt{|E|}\log^{3/2}|E|))$ for the maximum routable demand problem when the demand values are polynomially bounded. Their algorithm is based on an LP-relaxation and randomized rounding. Azar and Regev [3] describe a deterministic algorithm with performance ratio $\Omega(1/(\sqrt{|E|}\log(2+D_{\max}/u_{\min})))$; here, $D_{\max}$ and $u_{\min}$ denote the maximal demand and minimal edge-capacity, respectively. Kolman and Scheideler [15] give a greedy $\Omega(1/\sqrt{|E|})$-approximation algorithm without any assumption on demands or capacities.

The unsplittable flow problem is much easier if all commodities share a common single source; nevertheless, the resulting *single source unsplittable flow problem* remains strongly NP-hard. Various constant factor approximation algorithms have been developed for the congestion version of this problem. Again assuming that the balance condition holds, Kleinberg [8], [9] gives a 16-approximation algorithm. Kolliopoulos and Stein [13] (see also [12]) introduce a partitioning and scaling technique for the single-source unsplittable flow problem. Like Kleinberg, they assume that the balance condition is fulfilled and present a 3-approximation algorithm for the congestion version without cost and a bicriteria approximation algorithm of performance ratio 2 for cost and 3 for congestion. If the balance condition is not given, they can achieve a performance guarantee of $3 + 2\sqrt{2}$ for the case without costs. Dinitz et al. [6] consider the problem without costs. They give a 2-approximation for the case with balance condition, and a 5-approximation, otherwise. Finally, Skutella [18] improved on the approximation results for the budget-constrained version by refining the scaling technique of Kolliopoulos and Stein. Without relaxing the budget constraint, he gives a 3-approximation algorithm in presence of the balance condition and a $(3 + 2\sqrt{2})$-approximation, otherwise. For an overview of results on the other optimization versions of the single-source unsplittable flow problem we refer to [18].

In a recent work Bagchi et al. [4] consider fault-tolerant routings in networks. They independently define problems similar to our uniform exactly k-splittable flow problem. To ensure connection for each commodity for up to $k-1$ edge failures in the network, they require edge disjoint flow-paths per commodity. Their algorithm builds on the work of Kolman and Scheideler [15] and uses a bounded greedy algorithm which is analyzed using the "flow number."

The question of bounded splittability is also considered in the scheduling context in a recent paper by Krysta et al. [16].

Contribution of this paper. In Section 2, approximation results for the maximum k-splittable s-t-flow problem are derived. We start with the special cases $k=2$ and $k=3$ in Section 2.1 and show that approximate solutions can be obtained by executing two iterations of the classical augmenting path algorithm. In each iteration, an augmenting path with maximal residual capacity is chosen. The resulting s-t-flow is 3-splittable and its value constitutes an upper bound on the value of a maximum 2-splittable s-t-flow. These insights yield fast approximation algorithms with performance ratio $2/3$ for the maximum 2- and 3-splittable s-t-flow problem. Surprisingly, this approximation result is best possible, unless P = NP.

In order to derive approximation algorithms for higher values of k, it is first shown in Section 2.2 that a maximum uniform (budget-constrained) k-splittable s-t-flow can be computed in polynomial time. For the problem without costs, the presented algorithm is a subtle variant of the augmenting path algorithm with exactly k iterations. In contrast to the classical augmenting path algorithm, the width of a path chosen in one iteration is subject to constant decrease in later iterations. In fact, apart from its usefulness in the context of k-splittable flows, this algorithm might be of interest in its own right. We show a maximum-flow minimum-cut theorem for uniform exactly-k-splittable s-t-flows.

In Section 2.3 it is shown that the value of a maximum uniform exactly-k-splittable s-t-flow approximates the value of a maximum k-splittable s-t-flow within a factor of $\frac{1}{2}$. This result can be extended to the problem with costs yielding a $\frac{1}{2}$-approximation algorithm for the maximum budget-constrained k-splittable s-t-flow problem.

In Section 3, approximation results for the maximum concurrent flow version of the k-splittable multi-commodity flow problem are presented. We consider uniform exactly-k-splittable multi-commodity flows. As in the single-commodity case we show that the optimal values differ by a factor of at most 2. However, in contrast to the single-commodity case, the uniform exactly-k-splittable multi-commodity flow problem is already NP-hard since it contains the unsplittable flow problem as a special case. We show that the two problems are in fact equivalent, that is, an instance of the uniform exactly-k-splittable multi-commodity flow problem can be transformed into an equivalent instance of the unsplittable flow problem. In particular, an arbitrary approximation algorithm for the maximum concurrent flow version of the unsplittable flow problem with performance ratio ρ yields a $\rho/2$-approximation algorithm for the k-splittable multi-commodity flow problem. Again, this result can be generalized to the setting with costs, i.e., to the maximum concurrent budget-constrained k-splittable flow problem.

Finally, in Section 4 a result on the hardness of approximation is presented. It is obtained via a reduction from the classical SATISFIABILITY problem. The underlying construction is similar to a reduction used in [18] in a related context. Additionally, we show that a slightly modified problem in which the flow value is given and the number of flow-paths has to be minimized is strongly NP-hard, too.

2. The Single-Commodity Case.

In this section we consider the *maximum k-splittable s-t-flow problem*, that is, the case of only one commodity whose demand must be routed on at most k paths from source s to target t. This problem is already strongly NP-hard since it contains the single-source unsplittable flow problem as a special case: Assume we are given an instance of the single-source unsplittable flow problem, that is, a graph G with source s, sinks $t_1, \dots, t_k$, and corresponding demands $D_1, \dots, D_k$. We add a new node t together with edges $t_i t$ of capacity D_i, for $i = 1, \dots, k$. Computing a k-splittable flow from source s to target t of flow value $\sum_{i=1}^{k} D_i$ is equivalent to solving the given single-source unsplittable flow problem. Thus, the k-splittable s-t-flow problem is NP-hard, see, e.g., [8].

As mentioned above, the k-splittable s-t-flow problem can be solved efficiently by classical flow techniques if $k \geq |E|$. In the remainder of this section we therefore assume that $k < |E|$. We start by discussing the special cases $k = 2$ and $k = 3$.

2.1. *Approximating Maximum 2-Splittable and 3-Splittable s-t-Flows.*

We start with the maximum 2-splittable flow problem and give a simple combinatorial $\frac{2}{3}$-approximation algorithm. As we show in Section 4, this result is best possible in the sense that no approximation algorithm with a strictly better performance ratio exists for this problem, unless P = NP.

Our algorithm (described in the proof of Theorem 3) uses a special variant of the classical augmenting path algorithm for maximum flows which chooses an augmenting s-t-path of maximal residual capacity in each iteration. Notice that such a *maximum capacity path* can be found in $\mathcal{O}(|E| \log |E|)$ time by a modified Dijkstra labeling algorithm. This variant is known as the *maximum capacity augmenting path algorithm* (see, e.g., Chapter 7.3 of [1]).

We first show that the resulting flow after two iterations of the augmenting path algorithm is in fact 3-splittable.

LEMMA 1. *After two iterations of the augmenting path algorithm, the resulting flow can be decomposed into the sum of a 3-splittable flow and a circulation. Moreover, such a decomposition can be computed in linear time.*

PROOF. Let P_1 and P_2 denote the two augmenting paths found by the algorithm with corresponding flow augmentations f_1 and f_2, respectively. By construction, path P_1 consists only of forward edges. If the same holds for path P_2, then a natural flow decomposition is given by P_1 and P_2. However, since path P_2 is computed in the residual graph corresponding to the flow along P_1, it may contain backward edges.

We can give a decomposition into the sum of flows on at most three paths as follows: The first path in the decomposition is P_1 with flow value $f_1 - f_2$ (if $f_1 = f_2$, it can be ignored). If we reduce the flow along path P_1 by $f_1 - f_2$, the remaining flow is f_2-integral and has flow value $f_1 + f_2 - (f_1 - f_2) = 2f_2$. Using standard flow decomposition techniques, such a flow can be decomposed in linear time into the sum of flows of value f_2 on two s-t-paths plus a circulation (iteratively, take any flow-carrying s-t-path and reduce the flow on it by f_2). □

LEMMA 2. *After two iterations of the maximum capacity augmenting path algorithm, the value of the resulting flow is an upper bound on the value of a maximum* 2*-splittable flow.*

PROOF. Let F^* be a maximum 2-splittable flow solution which sends flow on paths P_1^* and P_2^* with flow values f_1^* and f_2^*, respectively. Moreover, let P_1 and P_2 denote the two augmenting paths found by the maximum capacity augmenting path algorithm with corresponding flow augmentations f_1 and f_2. We have to show that $f_1 + f_2 \geq f_1^* + f_2^*$.

Consider the subgraph G' of G induced by all edges in paths P_1, P_1^*, and P_2^*. Set the capacity of each edge in G' to the smallest value such that both the optimal solution $\{(P_1^*, f_1^*), (P_2^*, f_2^*)\}$ and the flow of value f_1 on path P_1 are feasible. Consider now the residual graph of G' with respect to the flow of value f_1 on P_1. We will show that there is an s-t-path of capacity at least $f_1^* + f_2^* - f_1$ in this residual graph. This yields $f_2 \geq f_1^* + f_2^* - f_1$ since the augmenting path algorithm could have chosen this path.

If $f_1^* + f_2^* - f_1 \leq 0$, there is nothing to be shown. We can thus assume that $f_1 < f_1^* + f_2^*$. Moreover, we know by choice of P_1 that $f_1 \geq \max\{f_1^*, f_2^*\}$. Let P be a thickest s-t-path in the residual graph with bottleneck edge e. The residual capacity of edge e is strictly positive since the value of a maximum s-t-flow in G' is at least $f_1^* + f_2^* > f_1$. We distinguish the following three cases:

Case 1: *edge e is a forward edge of* P_1. Since e has positive residual capacity, both P_1^* and P_2^* must contain e such that the residual capacity of e is $f_1^* + f_2^* - f_1$.

Case 2: *edge e is a backward edge of* P_1. In this case the residual capacity of e is $f_1 \geq f_1^* + f_2^* - f_1$.

Case 3: *edge e is not contained in* P_1. In this case the residual capacity of e is at least $\min\{f_1^*, f_2^*\} \geq f_1^* + f_2^* - f_1$. □

Lemmas 1 and 2 together yield the following approximation result for the maximum 2-splittable flow problem.

THEOREM 3. *There exists a* $\frac{2}{3}$*-approximation algorithm for the maximum* 2*-splittable* s-t*-flow problem with running time* $\mathcal{O}(|E| \log |E|)$.

PROOF. Run two iterations of the maximum capacity augmenting path algorithm. The resulting flow can be decomposed into three paths and a circulation by Lemma 1. Deleting the circulation and the path with the smallest flow value yields a 2-splittable flow. The performance guarantee of $\frac{2}{3}$ is an immediate consequence of Lemma 2. Finally, the running time is dominated by the modified Dijkstra labeling algorithm which is $\mathcal{O}(|E| \log |E|)$. □

In Section 4 we show that this result is in fact tight, that is, there does not exist an approximation algorithm for the 2-splittable flow problem with performance guarantee $\frac{2}{3} + \varepsilon$ for any $\varepsilon > 0$, unless P = NP. The result for $k = 2$ from Theorem 3 can easily be carried over to the maximum 3-splittable flow problem.

COROLLARY 4. *There exists a* $2/k$*-approximation algorithm for the maximum* k*-splittable* s-t*-flow problem with running time* $\mathcal{O}(|E| \log|E|)$, *for* $k \geq 3$. *In particular there is a* $\frac{2}{3}$*-approximation algorithm for* $k = 3$.

PROOF. Again, we run two iterations of the maximum capacity augmenting path algorithm. The resulting flow is decomposed into three paths and a circulation which we remove. By Lemma 2 the flow value of the resulting 3-splittable flow is at least as big as the maximum flow on two paths. However, the flow value of a maximum 2-splittable flow is at least a fraction of $2/k$ of the flow value of a maximum k-splittable flow. □

Unfortunately, the straightforward approach which led to the approximation result in Theorem 3 cannot be extended directly to a larger number of iterations. Firstly, k iterations of the maximum capacity augmenting path algorithm do not necessarily yield an upper bound on the value of a maximum k-splittable flow; see Figure 1. Secondly, for arbitrary k, it is difficult to bound the number of flow-carrying paths necessary in a path decomposition of the flow resulting from k iterations of the maximum capacity augmenting path algorithm. In Figure 2 we show that one augmentation may even double this number.

2.2. *Computing Maximum Uniform* k*-Splittable* s-t*-Flows*. Before we turn to the general k-splittable s-t-flow problem, we first discuss the problem of determining a maximum k-splittable s-t-flow where all paths with positive flow value carry the same amount of flow. We refer to this problem as the *uniform* k-splittable s-t-flow problem. In contrast to the problem with arbitrary flow values, which is strongly NP-hard, the uniform k-splittable s-t-flow problem turns out to be solvable in polynomial time. This result is used in Section 2.3 as an important building block for the construction of approximate solutions to the general problem. Note that for uniform k-splittable flows the maximal

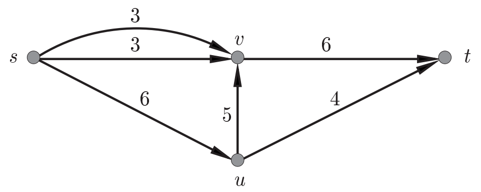

Fig. 1. A simple example showing that Lemma 2 cannot be generalized to values of k greater than 2. A 3-splittable s-t-flow can use the paths s-v-t, s-v-t, and s-u-t with flow values 3, 3, and 4, respectively. Thus, a maximum 3-splittable flow has value 10. However, in its first two iterations, the maximum capacity augmenting path algorithm chooses the paths s-u-v-t and s-v-u-t of bottleneck capacity 5 and 3. Now both edges incident to t have residual capacity 1. Thus, the maximum capacity augmenting path algorithm yields after three augmentations a flow of value 9.

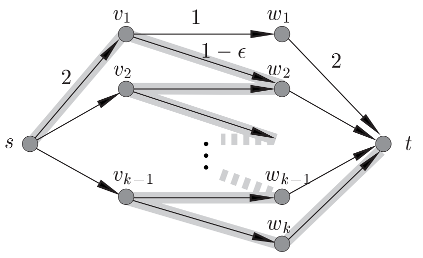

Fig. 2. A single augmentation can double the number of necessary paths in a path decomposition. Assume we have a flow of value 1 on each of the $k-1$ horizontal s-t-paths via v_i and w_i. As the augmenting path we use the gray marked s-t-path via $v_1, w_2, v_2, w_3, \ldots, w_{k-1}, v_{k-1}, w_k$ with residual capacity $1-\varepsilon$. Increasing the flow along this path requires $k-1$ additional paths.

flow value can increase even for values of k greater than $|E|$. Consider the graph consisting of only two parallel s-t-edges with capacity 1 and 2; a maximum uniform 2-splittable s-t-flow has value 2 and a maximum uniform 3-splittable s-t-flow has value 3.

Consider first a further specialization of the k-splittable s-t-flow problem where flows of identical value must be sent on *exactly* k paths. We refer to this problem as the *uniform exactly-k*-splittable s-t-flow problem. Assume that we know the maximal flow value D. Then our problem is equivalent to finding a D/k-integral s-t-flow of value D; notice that such a flow can easily be decomposed into a sum of k path-flows of value D/k plus a circulation (iteratively, take any flow-carrying s-t-path and reduce the flow on it by D/k). As mentioned above, this problem is efficiently solvable by standard flow techniques. Thus, we can find an optimal flow if we know the optimal flow value D.

Unfortunately, neither the flow value per path D/k nor the total flow value D have to be integral in general. Consider a graph consisting of two parallel edges from s to t, each with capacity 1. For odd $k = 2q+1 > 2$, there is an optimal solution that uses q times the first edge and $q+1$ times the second edge as paths. Thus, the maximal flow value per path is $1/(q+1)$ which yields a non-integral uniform exactly-k-splittable flow of value $2 - 1/(q+1)$. Even if we do not require exactly k paths, the optimal value may be non-integral; an example is given in Figure 3.

How "fractional" is an optimal solution in the "worst" case? Consider an arbitrary optimal solution. Since the flow value is maximal, there must exist at least one edge with a tight capacity constraint. Thus, the flow value on any path is equal to the capacity of this edge divided by the number of paths using this edge in the optimal solution under consideration. Hence, possible flow values are of the form u_e/i, for some edge $e \in E$

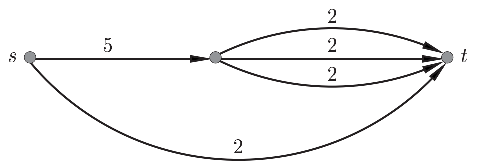

Fig. 3. An instance showing that the value of a maximum uniform k-splittable s-t-flow is in general not integral. The values attached to the edges denote the edge capacities. A maximum uniform 4-splittable s-t-flow has value $\frac{20}{3}$ and uses four paths with flow value $\frac{5}{3}$.

and some number $i \in \{1, \ldots, k\}$. That is, there are at most $|E|k$ possible values which can be enumerated in polynomial time.

OBSERVATION 5. *The maximum uniform exactly-k-splittable s-t-flow problem can be solved in polynomial time.*

If we add costs and consider the budget-constrained version of the maximum uniform exactly-k-splittable s-t-flow problem, the situation gets more complicated. In this case the flow value per path may be forced by the budget constraint instead of a capacity constraint. Nevertheless, we can prove the following result.

THEOREM 6. *A maximum budget-constrained uniform exactly-k-splittable s-t-flow can be computed in time $\mathcal{O}(k^2|E|^2 \log|E|)$.*

PROOF. Consider an arbitrary optimal solution to the problem under consideration and let $\alpha \geq 1$ be the maximal factor by which the solution can be scaled without violating the capacity of an edge. Clearly, the scaled flow is a uniform exactly-k-splittable s-t-flow of flow value ku_e/i for some tight edge $e \in E$ and some number $i \in \{1, \ldots, k\}$. Moreover, assuming that the budget constraint is tight, its cost is αB where B is the given budget. Therefore, a minimum-cost u_e/i-integral flow with flow value ku_e/i has cost βB with $\beta \leq \alpha$. Thus, scaling it by a factor $\min\{1/\beta, 1\}$ yields an optimal budget-constrained uniform exactly-k-splittable s-t-flow.

Using this insight, we can now state an algorithm which solves the maximum budget-constrained uniform exactly-k-splittable s-t-flow in polynomial time. For all $e \in E$ and $i \in \{1, \ldots, k\}$, compute a minimum-cost u_e/i-integral flow with flow value ku_e/i and scale it down until it obeys the cost-constraint. From the resulting $k|E|$ budget-constrained uniform exactly-k-splittable s-t-flows take one with maximal flow value. As shown above, this yields a maximum budget-constrained uniform exactly-k-splittable s-t-flow.

The running time of this algorithm is dominated by the $k|E|$ minimum-cost flow computations. Each minimum-cost flow can be computed in time $\mathcal{O}(k|E| \log|E|)$ by the successive shortest path algorithm. $\square$

Before we present an algorithm with a much better running time for the problem without costs, we show a maximum-flow minimum-cut theorem for uniform exactly-k-splittable s-t-flows. As in standard flow theory an s-t-cut C in graph G is a subset of the vertices such that $C \cup (V \backslash C)$ is a partition of the vertex-set that separates s and t. In contrast to standard flow theory we have to define the capacity of a cut C in relation to k to deal with the restriction to exactly k paths. We formulate the cut capacity as a packing problem. Let A be a multi-set of bins with capacities corresponding to the capacities of all edges from C to $V \backslash C$. The *k-uniform cut capacity* $c_k(C)$ of C is the maximal volume of a packing of k identically sized packages into the bins of multi-set A. An s-t-cut of minimal capacity $c_k(s, t) := \min\{c_k(C) | s \in C \subseteq V \backslash \{t\}\}$ is called a *minimum k-uniform s-t-cut*. Now we can formulate an analog to the maximum-flow minimum-cut theorem of Ford and Fulkerson for uniform exactly-k-splittable s-t-flows.

THEOREM 7. *The minimal k-uniform s-t-cut value $c_k(s, t)$ equals the maximal flow value of a uniform exactly-k-splittable s-t-flow.*

PROOF. Obviously $c_k(s, t)$ is an upper bound on the maximum uniform exactly-k-splittable s-t-flow value. It remains to show the reverse relation. Let G' be a copy of graph G with edge capacities $u'_e := \lfloor ku_e/c_k(s, t)\rfloor$, for $e \in E$; by definition of u'_e, exactly u'_e objects of size $c_k(s, t)/k$ fit into a bin of capacity u_e. Thus, by definition of $c_k(C)$, the standard minimum s-t-cut of G' has capacity at least k. Integral flow theory grants an integral s-t-flow of value k in G'. That flow can be decomposed into k paths each of flow value 1. Using these k paths with flow value $c_k(s, t)/k$ per path in the original graph G yields a feasible uniform exactly k-splittable s-t-flow of value $c_k(s, t)$. □

Now we present a more efficient algorithm for the problem without costs. The algorithm is a variant of the augmenting path algorithm with exactly k iterations. As augmenting paths we use maximum capacity paths in a special residual graph which is defined as follows.

Let $P_1, \dots, P_i$ be the paths found in iterations 1 to i, and let f_i denote their common flow value after iteration i. The resulting s-t-flow of value if_i is denoted by F_i. We construct the following residual graph $\overline{G}_i$: For each edge e of the original graph, let q^i_e be the number of paths among $P_1, \dots, P_i$ containing e. The residual capacity of edge e in $\overline{G}_i$ is set to $u_e/(1 + q^i_e)$. Moreover, if $q^i_e > 0$, we additionally insert a backward edge of capacity f_i into $\overline{G}_i$.

In $\overline{G}_i$ we compute a maximum capacity s-t-path P_{i+1} with bottleneck capacity $f_{i+1} := u_{P_{i+1}}$. Then a new flow F_{i+1} of value $(i + 1)f_{i+1}$ is defined as follows. Send f_{i+1} units of flow on all paths $P_1, \dots, P_i, P_{i+1}$. By definition of the residual graph $\overline{G}_i$, the resulting flow F_{i+1} is feasible in G. However, since path P_{i+1} may contain backward edges, we still have to argue that F_{i+1} is a uniform exactly-$(i + 1)$-splittable flow. This follows immediately since F_{i+1} is f_{i+1}-integral and has flow value $(i + 1)f_{i+1}$. Hence, we obtain a new set of $i + 1$ paths which have flow value f_{i+1} each and provide a feasible $(i + 1)$-splittable flow.

THEOREM 8. *The above algorithm finds a maximum uniform exactly-k-splittable s-t-flow in time $\mathcal{O}(k|E| \log|E|)$.*

We prove Theorem 8 by showing that there is a k-uniform s-t-cut of capacity kf_k. First we mention two observations on the behavior of the flow value with increasing k. The maximal s-t-flow value of a uniform exactly-k-splittable flow is not monotone in k. Consider the graph with two parallel unit-capacity s-t edges discussed above. For k even, the maximal flow value is 2. However, for k odd, the maximal flow value is $2-2/(k+1)$. On the other hand, although the total flow value is not monotone, the flow value per path is monotonically non-increasing for increasing k. Otherwise we could use the solution with the greater flow value per path and remove surplus paths yielding a solution contradicting maximality.

If a modified Dijkstra labeling algorithm is used to find a maximum capacity s-t-path in the ith iteration, we can employ the labeling to specify a minimum i-uniform s-t-cut.

Note that the label at a node v represents the capacity of a maximum bottleneck capacity s-v-path. Let C_i denote the set of all vertices which have a label strictly greater than the label f_i of t; assume s gets the label infinity. Obviously $C_i \cup (V \backslash C_i)$ is a partition separating s and t.

LEMMA 9. *If the flow value per path drops in iteration i (i.e., $f_{i-1} > f_i$), then cut C_i is a minimum i-uniform s-t-cut and $c_i(s, t) = i f_i$.*

PROOF. Assume cut C_i has an i-uniform cut capacity greater than $i f_i$. Thus, there are edges from C_i to $V \backslash C_i$ which concurrently can have a load greater than $i f_i$. We show that this assumption implies a label greater than f_i for at least one node in $V \backslash C_i$, contradicting our choice of C_i.

Consider a packing for cut C_i of volume $c_i(C_i)$; thus the package size $\sigma := c_i(C_i)/i$ is greater than f_i. Until iteration $i-1$ the algorithm constructed $i-1$ paths crossing the cut C_i of flow value f_{i-1} $(> f_i)$ each. Since all backward edges have capacity f_{i-1} $(> f_i)$, no backward edge is crossing the cut in the forward direction. Hence, none of the constructed $i-1$ paths is crossing the cut more than once. The considered packing has to put on at least one edge $e = vw$ from C_i to $V \backslash C_i$ more than q_e^{i-1} packages; in particular, $u_e/(1 + q_e^{i-1}) \geq \sigma$. However, $u_e/(1 + q_e^{i-1})$ is the capacity of edge e in $\overline{G}_i$. Since v is in C_i, it got a label greater than f_i. Thus, the labeling algorithm assigns w a label greater than f_i as well, contradicting the choice of C_i. As a consequence, each packing for cut C_i has package size at most f_i.

Since the above given augmenting path algorithm generates a feasible uniform exactly-i-splittable s-t-flow of value $i f_i$, there exists no i-uniform s-t-cut of value less than $i f_i$. Hence C_i is a minimum i-uniform s-t-cut. □

In general, Lemma 9 is wrong if the flow value per path does not change in iteration i. In that case we can use a cut from previous iterations.

LEMMA 10. *Assume $i > j$ are iteration indices such that $f_{j-1} > f_j = \cdots = f_i$. Then cut C_j is a minimum i-uniform s-t-cut and $c_i(s, t) = i f_i$.*

PROOF. From Lemma 9 we know that C_j is a minimum j-uniform s-t-cut. Thus, the maximal package size for C_j as a j-uniform s-t-cut is f_j. If we have to place i instead of j packages into the same set of bins we cannot expect a larger package size. On the other hand, we know that i packages of size f_j fit into the bins. □

PROOF OF THEOREM 8. The correctness follows immediately from Lemmas 9 and 10. The running time of the algorithm is bounded by k times the time required to find a maximum capacity s-t-path. The latter problem can be solved by a modified Dijkstra algorithm in time $\mathcal{O}(|E| \log |E|)$. □

As mentioned above, the flow value of a maximum uniform exactly-k-splittable flow is not necessarily increasing with k. However, since our algorithm computes a maximum uniform exactly-i-splittable flow for all $i = 1, \ldots, k$, we can easily determine the number of paths maximizing the flow value.

THEOREM 11. *The maximum uniform k-splittable s-t-flow problem can be solved efficiently in time $\mathcal{O}(k|E|\log|E|)$.*

2.3. *Approximating Maximum k-Splittable s-t-Flows.* We show that computing a maximum (budget-constrained) uniform k-splittable s-t-flow yields a constant factor approximation for the maximum (budget-constrained) k-splittable s-t-flow problem.

THEOREM 12. *Computing a maximum* (*budget-constrained*) *uniform exactly-k-splittable s-t-flow is an approximation algorithm with performance ratio $\frac{1}{2}$ for the maximum* (*budget-constrained*) *k-splittable s-t-flow problem.*

PROOF. Consider an optimal solution F of value OPT to the general (budget-constrained) k-splittable s-t-flow problem. We show that there exists a (budget-constrained) uniform exactly-k-splittable s-t-flow which sends D:=OPT/$(2k)$ units of flow on each of the k paths. In particular, the value of this flow is exactly OPT/2.

In the given optimal solution F, any flow-carrying path P with flow value $f_P > 0$ can be replaced by $\lfloor f_P/D \rfloor$ copies of P, each carrying D units of flow. It suffices to show that the resulting solution consists of at least k paths, each with flow value D. Using the fact that the number of paths used by F is bounded by k, we get

$$\sum_{P\,:\,f_P>0} \lfloor f_P/D \rfloor \;\geq\; \frac{\text{OPT}}{D} - k = k.$$

Finally, observe that the given conversion never increases flow on an edge and therefore never increases cost. This concludes the proof. □

On the other hand, we can show that the result in Theorem 12 is tight, that is, there exist instances with an asymptotic gap of 2 between the value of a maximum k-splittable s-t-flow and the value of a maximum uniform k-splittable s-t-flow. Consider a graph with two nodes s and t, and k parallel s-t edges. One of them has capacity $k-1$, the others have unit capacity. A maximum uniform k-splittable s-t-flow in this graph has value k; the maximum k-splittable s-t-flow has value $2(k-1)$.

In Section 2.2 we have seen that in contrast to standard s-t-flows maximum uniform (exactly) k-splittable s-t-flows may have only non-integral solutions in graphs with integral edge capacities. Now we give an example that the same holds for the general maximum k-splittable s-t-flow problem. Consider a maximum 4-splittable s-t-flow in the graph of Figure 4.

3. The Multi-Commodity Case. In this section we extend some techniques and results from the previous section to the general situation of multiple commodities. We consider the (budget-constrained) maximum concurrent flow problem where the aim is to maximize the routable fraction, i.e., maximize the factor by which we can multiply all

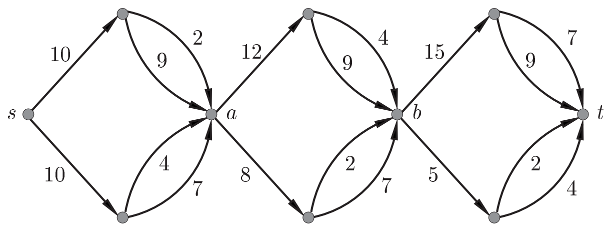

Fig. 4. There is a 4-splittable s-t-flow of flow value 20 if one uses $\frac{3}{2}$, $\frac{7}{2}$, $\frac{13}{2}$, and $\frac{17}{2}$ as path flow values. To see that there is no integral solution we show that at least one path must have a flow value between 1 and 2. It is easy to see that there is no 3-splittable s-t-flow of value 20 and that no solution may have a flow path of value larger than 9. Furthermore, at each of the three nodes s, a, and b two paths must branch upward and the other two paths must branch downward. Consider an arbitrary maximum solution. First assume there is a path of flow value $x < 1$. The two edges adjacent to s would force a path of flow value $10 - x > 9$. This leads to a contradiction. Assume all paths have flow value $x > 2$. Thus we can remove all edges of capacity 2 which can be seen to be infeasible. Now assume there is at least one path of value 1 (or of value 2). Again the edges adjacent to s force another path of value $9 = 10 - 1$ (or $8 = 10 - 2$). That path forces a path of value $3 = 12 - 9$ (or $4 = 12 - 8$) on the edge of capacity 12. That fixes the last path to a value of 7 (or 6). These values are not feasible.

demands and still get a realizable instance respecting the given capacities (and the budget constraint). Notice that the maximum concurrent flow problem reduces to the maximum flow problem if only a single commodity is considered. In some sense the maximum concurrent flow problem is more closely related to the single-commodity maximum flow problem than the versions that minimize the number of rounds or that maximize the routable demand. For the maximum concurrent flow problem only an optimal routing has to be found whereas, both in the minimum number of rounds and the maximum routable demand version, one has to solve two problems, a selection and a routing problem.

As in Section 2.3, we use uniform exactly-k-splittable flows in order to find approximate solutions.

THEOREM 13. *For the multi-commodity case*, *the value of a maximum concurrent* (*budget-constrained*) *uniform exactly-k-splittable flow approximates the optimal value of the corresponding general maximum concurrent* (*budget-constrained*) *k-splittable flow problem within a factor of* $\frac{1}{2}$.

PROOF. As in the proof of Theorem 12, we turn an arbitrary k-splittable flow into a uniform exactly-k-splittable flow which satisfies at least one-half of the original demands. Notice that the technique described in the proof of Theorem 12 can be applied to the commodities one after another. Since flow on an edge is never increased during this conversion, the resulting uniform exactly-k-splittable flow is feasible (and obeys the budget-constraint). □

However, in contrast to the single-commodity case, the multi-commodity case of the maximum concurrent uniform exactly-k-splittable flow problem is NP-hard, since it contains the unsplittable flow problem as a special case (set all k_i to 1, for $i = 1, \ldots, \ell$). Thus, we can only expect approximate solutions.

Assume that we want to route concurrently a fraction μ of the demand of each commodity. Then each path of commodity i has to carry exactly $\mu D_i/k_i$ units of flow. In particular, there is no choice for the number of paths and for the amount of flow if μ is known. Thus, the problem is equivalent to the following unsplittable flow problem on the same graph: For each terminal pair (s_i, t_i) of our k-splittable flow problem, $i = 1, \ldots, \ell$, we introduce k_i terminal pairs (s_i, t_i) and assign a demand value of D_i/k_i to all of them. A concurrent unsplittable flow for the new problem instance with value μ is also a concurrent flow solution for the uniform exactly-k-splittable flow problem of the same value, and vice versa. Combining this observation with Theorem 13 yields the following result.

THEOREM 14. *Any ρ-approximation algorithm for the maximum concurrent* (*budget-constrained*) *unsplittable flow problem yields an approximation algorithm with performance guarantee $\rho/2$ for the maximum concurrent* (*budget-constrained*) *k-splittable flow problem.*

There are various approximation algorithms known for the maximum concurrent unsplittable flow problem, see [8]. The randomized rounding technique of Raghavan and Thompson [17] yields a logarithmic approximation. For the case of a single-source node, there even exist constant factor approximation algorithms, see [9], [13], [6], and [18]. More details can be found in the Introduction.

As pointed out by an anonymous referee, the linear program relaxation of the maximum concurrent unsplittable flow problem is also a relaxation of the k-splittable flow problem. Hence, any ρ-approximation algorithm for the unsplittable flow problem which compares the approximate solution with a solution to the fractional relaxation already leads to a ρ-approximation algorithm for the k-splittable flow problem. In particular, this applies to the currently best known approximation algorithms for the maximum concurrent unsplittable flow problem [8], [6], [18].

4. Complexity and Non-Approximability Results.

In this section we show that the k-splittable s-t-flow problem is NP-hard in the strong sense for directed graphs, even if k is fixed to 2. More precisely, we give a reduction from SAT showing that for $k = 2$ no polynomial-time algorithm with performance ratio strictly better than $\frac{2}{3}$ exists, unless P = NP.

THEOREM 15. *It is strongly NP-hard to approximate instances of the maximum* 2-*splittable s-t-flow problem with an approximation ratio strictly better than* $\frac{2}{3}$.

The following proof of Theorem 15 is similar to and inspired by the proof of Theorem 9 and Corollary 8 of [18].

PROOF. We give a reduction from the NP-complete SAT problem. Given a SAT instance with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$, we construct a graph of size linear in n and m with source s and sink t. We will show that for a satisfiable instance of the SAT

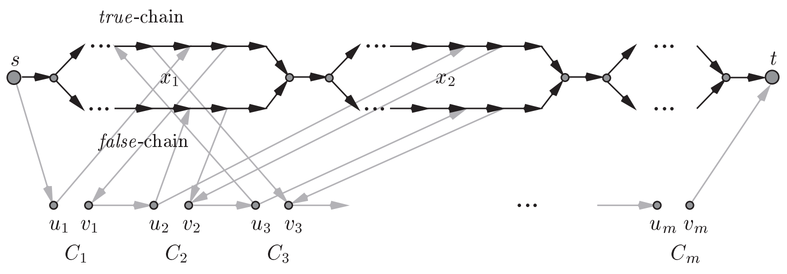

Fig. 5. A graph for encoding a SAT instance as a 2-splittable flow problem. An s-t-path in the subgraph of black edges describes a truth assignment for the variables $x_1, \ldots, x_n$. An s-t-path through the nodes $u_1, v_1, \ldots, u_m, v_m$ proves the feasibility of the assignment. All black edges have capacity 2, all gray edges have capacity 1. In this example: $C_1 = \neg x_1$, $C_2 = x_1 \vee \neg x_2$, and $C_3 = \neg x_1 \vee x_2$.

problem, there is a 2-splittable s-t-flow of value 3. Otherwise, if the SAT instance is not satisfiable, a maximum 2-splittable s-t-flow has value 2. Thus, an approximation algorithm with performance ratio strictly better than $\frac{2}{3}$ would imply a polynomial algorithm for deciding the satisfiability of the SAT instance.

We construct the graph in two stages. First, the subgraph for encoding a truth assignment of $x_1, \ldots, x_n$ is described. This subgraph is acyclic with $n + 1$ bridges and a unique source s and sink t; see the part drawn in black in Figure 5. For each variable x_i, there is a segment built by two parallel paths between consecutive bridges. One of these paths is called the *true*-chain, the other the *false*-chain of x_i. The *true*-chain (*false*-chain) of variable x_i contains two consecutive edges for each clause in which x_i occurs negated (unnegated). We call the first edge of each pair the *representing* edge of the corresponding clause. The order of clauses in each chain is monotonically decreasing by clause index. All edges in this "variable subgraph" have capacity 2. If an s-t-path with flow value 2 uses the *true*-chain (*false*-chain) of x_i then we interpret it as assigning the value true (false) to x_i.

Secondly, there is a subgraph for encoding the clauses. The idea is to construct a second s-t-path using the representing edges. This path gets blocked if the assignment given by the first path is not satisfying. For each clause C_i there are two nodes u_i, v_i and edges from v_{i-1} to u_i; the node v_0 coincides with the source s and u_{m+1} coincides with the sink t. Node u_i is connected to all tail nodes of representing edges of clause C_i and v_i is connected to all head nodes of representing edges of C_i (see the gray edges in Figure 5). All of these edges get capacity 1.

Assume that the SAT instance is satisfiable. We fix a satisfying assignment and send 2 units of flow on a single s-t-path through the "variable subgraph". This path uses the *true*-chain of x_i if the variable x_i is set to true and the *false*-chain otherwise. By construction of the graph, there is a path for clause C_j from u_j to v_j via the representing edge of C_j in the *true*/*false*-chains of variable x_i. This u_j-v_j-path is disjoint from the above chosen s-t-path if the assignment of x_i satisfies clause C_i. Since, by assumption, there is a satisfying assignment, we can construct an s-t-path that is disjoint from the first s-t-path. Consequently, there is a 2-splittable s-t-flow of value 3 if the SAT instance is satisfiable. Furthermore, by the capacity of the edges leaving s, this is maximal. It remains to show that a 2-splittable s-t-flow of value greater than 2 yields a satisfying truth assignment.

First notice that the two paths must be edge disjoint, because otherwise the flow value is at most 2. Second, one of the paths must be entirely in the black "variable graph" since otherwise both paths have flow value at most 1.

Path P_1 with flow value greater than 1 is used for assigning the values to the variables. As mentioned, this path lies entirely in the "variable graph". Therefore, for each variable x_i it has to use all edges of either the *true*- or the *false*-chain of x_i. Path P_2, on the other hand, has to leave s via edge (s, u_1) and then traverses representing edges for the clauses C_i in order of increasing index, intercepted by (v_i, u_{i+1}) edges. The consecutive traversal of representing edges according to increasing clause index is assured by the decreasing ordering of the representing edges within each variable segment. Consequently, P_2 cannot skip any of the u_i, v_i nodes and thus verifies that our assignment is satisfying. □

In Section 2.1 we have seen that the maximum uniform k-splittable s-t-flow problem can be solved in polynomial time. However, as soon as we allow small differences in the flow values on the paths, the problem becomes NP-hard again.

COROLLARY 16. *For arbitrary $\varepsilon > 0$, it is strongly NP-hard to compute a maximum 2-splittable s-t-flow, already if the flow values on the two paths may differ by a factor of no more than $1 + \varepsilon$.*

On the other hand, the problem described in Corollary 16 is easier to approximate since a maximum uniform k-splittable s-t-flow is obviously a $1/(1 + \varepsilon)$-approximate solution.

Finally, we want to mention a closely related problem. Instead of looking for a maximum s-t-flow using at most k paths one can require an amount of flow and ask for the minimal number of paths necessary to satisfy this demand on flow. The construction in the proof of Theorem 15 shows that this problem is NP-hard as well. The SAT instance is a *yes*-instance if and only if the minimal number of paths necessary to ship 3 units of flow is 2.

COROLLARY 17. *Computing an s-t-flow of given value using a minimal number of paths is strongly NP-hard.*

Acknowledgements. We thank three anonymous referees for valuable comments that improved the readability of the paper.

References

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

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, On-line routing of virtual circuits with applications to load balancing and machine scheduling, *Journal of the Association for Computing Machinery*, **44** (1997), 486–504.

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

[4] A. Bagchi, A. Chaudhary, C. Scheideler, and P. Kolman, Algorithms for fault-tolerant routing in circuit switched networks, in *Proceedings of the* 14*th Annual ACM Symposium on Parallel Algorithms and Architectures*, 2002, pp. 265–274.

[5] A. Baveja and A. Srinivasan, Approximation algorithms for disjoint paths and related routing and packing problems, *Mathematics of Operations Research*, **25** (2000), 255–280.

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

[7] 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* 31*st Annual ACM Symposium on Theory of Computing*, 1999, pp. 19–28.

[8] J. Kleinberg, Approximation Algorithms for Disjoint Paths Problems, Ph.D. thesis, MIT, 1996.

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

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

[11] J. Kleinberg and E. Tardos, Disjoint paths in densely embedded graphs, in *Proceedings of the* 36*th Annual Symposium on Foundations of Computer Science*, 1995, pp. 52–61.

[12] S. G. Kolliopoulos, Exact and Approximation Algorithms for Network Flow and Disjoint-Path Problems, Ph.D. thesis, Dartmouth College, 1998.

[13] S. G. Kolliopoulos and C. Stein, Improved approximation algorithms for unsplittable flow problems, in *Proceedings of the* 38*th Annual Symposium on Foundations of Computer Science*, 1997, pp. 426–435.

[14] S. G. Kolliopoulos and C. Stein, Approximating disjoint-path problems using greedy algorithms and packing integer programs, in *Proceedings of the* 6*th Conference on Integer Programming and Combinatorial Optimization*, 1998, pp. 153–168.

[15] P. Kolman and C. Scheideler, Improved bounds for the unsplittable flow problem, in *Proceedings of the* 13*th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 184–193.

[16] P. Krysta, P. Sanders, and B. Vöcking, Scheduling and traffic allocation for tasks with bounded splittability, in *Proceedings of the* 28*th International Symposium on Mathematical Foundations of Computer Science*, 2003, pp. 500–510.

[17] P. Raghavan and C. Thompson, Randomized rounding: a technique for provably good algorithms and algorithmic proofs, *Combinatorica*, **7** (1987), 365–374.

[18] M. Skutella, Approximating the single-source unsplittable min-cost flow problem, *Mathematical Programming*, **91** (2002), 493–514.