Experimental evaluation of approximation algorithms for single-source unsplittable flow

Stavros G. Kolliopoulos* and Clifford Stein**

Abstract. In the *single-source unsplittable flow* problem, we are given a network G, a source vertex s and k commodities with sinks t_i and real-valued demands ρ_i, $1 \leq i \leq k$. We seek to route the demand ρ_i of each commodity i along a single s-t_i flow path, so that the total flow routed across any edge e is bounded by the edge capacity c_e. This NP-hard problem combines the difficulty of bin-packing with routing through an arbitrary graph and has many interesting and important variations. In this paper we initiate the experimental evaluation of approximation algorithms for unsplittable flow problems. We examine the quality of approximation achieved by several algorithms for finding a solution with near-optimal congestion. In the process we analyze theoretically a new algorithm and report on the practical relevance of heuristics based on minimum-cost flow. The experimental results demonstrate practical performance that is better than the theoretical guarantees for all algorithms tested. Moreover modifications to the algorithms to achieve better theoretical results translate to improvements in practice as well.

1 Introduction

In the *single-source unsplittable flow* problem (UFP), we are given a network $G = (V, E, u)$, a source vertex s and a set of k commodities with sinks $t_1, \ldots, t_k$ and associated real-valued demands $\rho_1, \ldots, \rho_k$. We seek to route the demand ρ_i of each commodity i, along a single s-t_i flow path, so that the total flow routed across any edge e is bounded by the edge capacity u_e. See Figure 1 for an example instance. The minimum edge capacity is assumed to have value at least $\max_i \rho_i$.

The requirement of routing each commodity on a single path bears resemblance to integral multicommodity flow and in particular to the multiple-source edge-disjoint path problem. For the latter problem, a large amount of work exists either for solving exactly interesting special cases (e.g. [10], [33], [32]) or for approximating, with limited success, various objective functions (e.g. [31],[11],

* NEC Research Institute, Inc., 4 Independence Way, Princeton, NJ 08540. Email: stavros@research.nj.nec.com. Part of this work was performed at the Department of Computer Science, Dartmouth College while partially supported by NSF Award NSF Career Award CCR-9624828.

** Dartmouth College, Department of Computer Science, Hanover, NH 03755-3510. E-mail: cliff@cs.dartmouth.edu. Research partly supported by NSF Career Award CCR-9624828.

[21], [22],[19]). Shedding light on single-source unsplittable flow may lead to a better understanding of multiple-source disjoint-path problems. From a different perspective, UFP can be approached as a variant of standard, single-commodity, maximum flow and generalizes single-source edge-disjoint paths. UFP is also important as a unifying framework for a variety of scheduling and load balancing problems [20]. It can be used to model bin-packing [20] and certain scheduling problems on parallel machines [27]. Another application for UFP is in virtual-circuit routing, where the source represents a node wishing to multicast data to selected sinks. The conceptual difficulty in UFP arises from combining both packing constraints due to the existence of capacities and path selection through a graph of arbitrary topology.

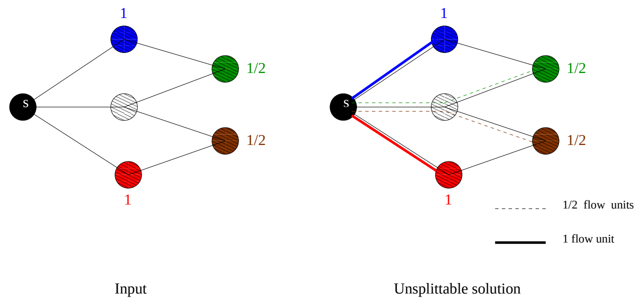

Fig. 1. Example UFP instance. Numbers on the vertices denote demands; all edge capacities are equal to 1.

The feasibility question for UFP, i.e., whether all commodities can be routed unsplittably, is strongly NP-complete [19]. Thus research has focused on efficient algorithms to obtain approximate solutions. For a minimization (resp. maximization) problem, a *ρ-approximation algorithm,* $\rho > 1$, ($\rho < 1$), outputs in polynomial time a solution with objective value at most (at least) ρ times the optimum. Different optimization versions can be defined such as minimizing congestion, maximizing the routable demand and minimizing the number of rounds [19]. In this work we focus on the congestion metric. The reader is referred to [20, 24, 7] for theoretical work on the other metrics.

Given a flow f define the *utilization* z_e of edge e as the ratio f_e/u_e. We seek an unsplittable flow f satisfying all demands, which minimizes *congestion,* i.e. the quantity $\max_e\{z_e, 1\}$. That is, the minimum congestion is the smallest number $\alpha \geq 1$, such that if we multiplied all capacities by α, f would satisfy the capacity constraints.

In this paper we initiate the experimental study of approximation algorithms for the unsplittable flow problem. Our study examines the quality of the effective approximation achieved by different algorithms, how this quality is affected by input structure, and the effect of heuristics on approximation and running time. The experimental results indicate performance which is typically better than the theoretical guarantees. Moreover modifications to the algorithms to achieve better theoretical results translate to improvements in practice as well. The main focus of our work is to examine the quality of the approximation and we have not extensively optimized our codes for speed. Thus we have largely opted for maintaining the relative simplicity and modularity that we believe is one of the advantages of the algorithms we test. However keeping the running time low allows the testing of larger instances. Therefore we report on wall clock running times, at a granularity of seconds, as an indication of the time complexity in practice. Our codes run typically much faster than multicommodity flow codes on similar size instances [28, 12, 30] but worse than maximum flow codes [5] with which they have a comparable theoretical running time [24].

We chose to focus our experimentation on the congestion metric for a variety of reasons. The approximation ratios and integrality gaps known for congestion are in general better than those known for the other metrics [19, 24, 7]. Moreover congestion is relevant for maximizing routed demand in the following sense: an unsplittable solution with congestion λ provides a guarantee that $1/\lambda$ fraction of each demand can be routed while respecting capacities. Finally, the congestion metric has been extensively studied theoretically in the setting of randomized rounding technique [31] and for its connections to multicommodity flow and cuts (e.g. [26, 17, 25, 18]).

Known theoretical performance guarantees. Kleinberg [19, 20] was the first to consider UFP as a distinct problem and to give (large) constant-factor approximation algorithms. The authors obtained PARTITION, a simple 4-approximation algorithm [24] and a 3.33 bound for a more elaborate version of PARTITION. Very recently, Dinitz, Garg, and Goemans [7] obtained an algorithm with a performance guarantee of $1 + \max_i\{\rho_i\} \leq 2$. This new algorithm matches the integrality gap known for the problem.

New algorithms and heuristics. We provide a new algorithm, called H_PARTITION, and prove that it is a 3-approximation algorithm. We outline the ideas behind our algorithms in Section 3. For both PARTITION and H_PARTITION we test several codes, each implementing different heuristics. All heuristics are designed so that they do not affect the theoretical performance guarantee of the underlying base algorithm (4 and 3 for PARTITION and H_PARTITION respectively). The heuristics fall into two major categories. The *cost heuristic* invokes minimum-cost flow subroutines when the original algorithm would invoke maximum flow to compute partial unsplittable solutions. The cost of an edge e is defined as an appropriate increasing function of the current congestion on e. The *sparsification heuristic* discards at an early stage of the execution the edges carrying zero amount of flow in the initial solution to the maximum flow relaxation. The motivation behind the sparsification heuristic is to decrease the running time and observe how different versions of the algorithms behave in practice since the zero edges are not exploited in the theoretical analysis. Finally, we test codes which combine both the sparsification and the cost heuristic.

Test data. We test our codes on five main families of networks. For each family we have generated an extensive set of instances by varying appropriate parameters. We test mostly on synthetic data adapted from previous experimental work for multicommodity flow [28, 12] and cuts [3] together with data from some new generators. The instances produced are designed to accommodate a variety of path and cut characteristics. Establishing a library of interesting data generators for disjoint-path and unsplittable flow problems is an important task in its own right given the practical significance of this class of problems. We believe our work makes a first contribution in this direction. We attempted to obtain access to real-world data from telecommunication and VLSI applications, but were unsuccessful due to their proprietary nature. However one of the input families we test (LA networks) is adapted from synthetic data, used in the restoration capacity study of Deng et al. [6], which was created based on actual traffic forecasts on realistic networks.

Experimental Results. In summary, the experimental results indicate, for both algorithms PARTITION and H_PARTITION, typical performance which is better than the theoretical guarantees. Theoretical improvements translate to practical improvements as well since algorithm H_PARTITION achieved significantly better approximations than algorithm PARTITION on the instances we examined. We observed that the sparsification heuristic can drastically reduce the running time without much adverse effect on congestion. Finally we show that running the cost heuristic can provide additional small improvement on congestion at the expense of an increase in running time.

2 Preliminaries

A *single-source unsplittable flow problem* (UFP) $(G, s, \mathcal{T})$ is defined on a capacitated network $G = (V, E, u)$ where a subset $\mathcal{T} = \{t_1, \ldots, t_k\}$ of V is the set of *sinks*; every edge e has a capacity u_e; each sink t_i has a demand ρ_i. The pair (t_i, ρ_i) defines the *commodity* i. To avoid cumbersome notation, we use occasionally the set $\mathcal{T}$ to denote the set of commodities. We assume in our definition that each vertex in V hosts at most one sink; the input network can always be transformed to fulfill this requirement. A *source* vertex $s \in V$ is specified. We denote by $\rho_{\max}$, $\rho_{\min}$ the maximum and minimum demand values for the problem of interest. For any $I \subseteq \mathcal{T}$, a *routing* is a set of s-t_{i_j} paths P_j, $1 \leq j \leq |I|$ used to route ρ_{i_j} amount of flow from s to t_{i_j} for each $t_{i_j} \in I$. Given a routing g, the flow g_e through edge e is equal to $\sum_{P_i \in g | P_i \ni e} \rho_i$. For clarity we will refer to a standard maximum flow as *fractional maximum flow*. We make two assumptions in our treatment of UFP. First, we assume that there is a feasible fractional flow solution. This assumption is introduced for simplicity, our approximation guarantees hold even without this assumption. The second assumption is a *balance* *assumption* that states that the maximum demand is less than or equal to the minimum edge capacity, i.e., a demand can be routed on any edge. This assumption is partly justified by the splitting of data into small size packets in actual networks so that no commodity has arbitrarily large demand. Even without the balance assumption a constant factor approximation is possible [24]. We will refer to instances that satisfy the balanced assumption as *balanced*, and ones which violate it as *unbalanced*, or more precisely, ones in which the maximum demand is ρ times the minimum capacity as ρ-unbalanced. Unless otherwise stated, we use n, m to denote $|V|, |E|$ respectively.

The following theorem is an easy consequence of the well-known augmenting path algorithm ([9], [8]) and will be of use.

Theorem 1. *Let $(G = (V, E, u), s, \mathcal{T})$ be a* UFP *on an arbitrary network G with all demands ρ_i, $1 \leq i \leq k$, equal to the same value σ, and edge capacities u_e equal to integral multiples of σ for all $e \in E$. If there is a fractional flow of value f, then there is an unsplittable flow of value at least f. This unsplittable flow can be found in time $O(km)$ where k is the number of sinks in $\mathcal{T}$.*

In the same way a routing corresponds to an edge flow, an edge flow can be represented as a path flow. Our algorithms will make use of the well-known *flow decomposition theorem* [1]. Given problem $(G, s, \mathcal{T})$ let a flow solution f be represented as an edge flow, i.e. an assignment of flow values to edges. Then one can find, in $O(nm)$ time, a representation of f as a path and cycle flow, where the paths join s to sinks in $\mathcal{T}$. In the remainder of the paper, we will assume that the cycles are deleted from the output of a flow decomposition.

3 The two basic algorithms

We present first briefly the 4-approximation algorithm PARTITION. Subsequently we present and analyze the 3-approximation algorithm H_PARTITION. Recall that in our theoretical exposition we assume that in an UFP demands lie in the interval $(0, 1]$ and capacities are at least 1. In particular let $1/D$ be the minimum demand value for real $D \geq 1$.

PARTITION works by partitioning the original UFP into subproblems, each containing only demands in a fixed range. We use an initial maximum-flow solution to allocate capacity to different subproblems. Capacities in the subproblems are arbitrary and may violate the balance assumption. The subproblems are separately solved by subroutine INTERVAL_ROUTING which finds low-congetion solutions; the partial solutions are then combined by superimposing the flow paths obtained in the respective routings on the original network G. The combination step will have a subadditive effect on the total congestion and thus a near-optimal global solution is achieved. Subroutine INTERVAL_ROUTING deals with a subproblem Π that has demands in the range $(1/2^i, 1/2^{i-1}]$ as follows. Let f be a feasible fractional solution to Π. Round all demands up to $1/2^{i-1}$ and call Π' the resulting UFP instance. To obtain a feasible fractional solution for Π', it suffices to multiply the flow f_e and the capacity u_e on each edge e

by a factor of at most 2. Further round all capacities up to the closest multiple of $1/2^{i-1}$ by adding at most $1/2^{i-1}$. The new edge capacities are now at most $2u_e + 1/2^{i-1}$. By Theorem 1 an unsplittable routing for Π' can now be found in polynomial time. Algorithm PARTITION is described formally in Fig. 2. Observe that an edge e which is assigned capacity $u_e^i = 0$ at Step 2, may still be used by subroutine INTERVAL_ROUTING on G_i to route up to a_i amount of flow.

Step 1. Find a feasible fractional solution f.
Step 2. Define a partition of the $[1/D, 1]$ interval into $\xi = \lfloor \log D \rfloor + 1$ consecutive subintervals $(1/2^{\lfloor \log D \rfloor + 1}, 1/2^{\lfloor \log D \rfloor}], (1/2^{\lfloor \log D \rfloor}, 1/2^{\lfloor \log D \rfloor - 1}], \ldots, (1/2, 1]$. Construct ξ copies of G where the set of sinks in G_i, $1 \leq i \leq \xi$, is the subset $\mathcal{T}_i$ of $\mathcal{T}$ with demands in the interval $(1/2^i, 1/2^{i-1}]$. Using flow decomposition determine for every edge e the amount u_e^i of u_e used by f to route flow to sinks in $\mathcal{T}_i$. Set the capacity of edge e in G_i to u_e^i.
Step 3. Invoke INTERVAL_ROUTING on each G_i, $1 \leq i \leq \xi$, to obtain an unsplittable routing g^i. Output routing g, the union of the path sets g^i, $1 \leq i \leq \xi$.

Fig. 2. Algorithm PARTITION$(G = (V, E, u), s, \mathcal{T})$.

Theorem 2. *[24] Given an* UFP Π, *Algorithm* PARTITION *finds a 4-approximation for congestion in polynomial time.*

The running time is dominated by the time to perform $O(\log n)$ maximum flow computations and to perform flow decomposition. The fastest maximum flow algorithms currently have running time $O(nm \log_{\frac{m}{n \log n}} n)$[13] and $O(\min(n^{2/3}, m^{1/2}) m \log(\frac{n^2}{m}) \log U)$ [14] when the capacities are integers in the range $[1, \ldots, U]$. See [23] for exact computations of running times. In our implementations, we use a maximum flow routine repeatedly in order to perform flow decomposition.

3.1 A 3-approximation algorithm

An improved approximation algorithm can be obtained by combining the partitioning idea above with a more careful treatment of the subproblems. At the first publication of our results [24] we achieved a $(3.33 + o(1))$-approximation. Subsequently we improved our scheme to obtain a 3 ratio; prior to this improvement, a 2-approximation for congestion was independently obtained by Dinitz, Garg and Goemans [7]. See [23] for a detailed exposition.

We give first without proof an application of the INTERVAL_ROUTING subroutine.

Lemma 1. *Let* $\Pi = (G, s, \mathcal{T})$, *be a* UFP *in which all demands have value* $1/2^x$ *for* $x \in N$, *and all capacities are multiples of* $1/2^{x+1}$, *and let* f *be a fractional flow solution such that the flow* f_e *through any edge is a multiple of* $1/2^{x+1}$. *We* *can find in polynomial time an unsplittable routing g where the flow g_e through an edge e is at most $f_e + 1/2^{x+1}$.*

Algorithm PARTITION finds a routing for each subproblem by scaling up subproblem capacities to ensure they conform to the requirements of Lemma 1. The new algorithm operates in phases, during which the number of distinct paths used to route flow for a particular commodity is progressively decreased. Call *granularity* the minimum amount of flow routed along a single path to any commodity, which has not been routed yet unsplittably. Algorithm PARTITION starts with a fractional solution of arbitrary granularity and essentially "in parallel" for all subproblems, rounds this fractional solution to an unsplittable one. In the new algorithm, we delay the rounding process in the sense that we keep computing a fractional solution of successively increasing granularity. The granularity will always be a power of $1/2$ and this unit will be doubled after each iteration. Once the granularity surpasses $1/2^x$, for some x, we guarantee that all commodities with demands $1/2^x$ or less have already been routed unsplittably. Every iteration improves the granularity at the expense of increasing the congestion. The method has the advantage that by Lemma 1, a fractional solution of granularity $1/2^x$ requires only a $1/2^x$ additive offset to current capacities to find a new flow of granularity $1/2^{x-1}$. Therefore, if the algorithm starts with a fractional solution of granularity $1/2^\lambda$, for some potentially large λ, the total increase to an edge capacity from then on would be at most $\sum_{j=1}^{j=\lambda} 1/2^j < 1$. Expressing the granularity in powers of $1/2$ requires an initial rounding of the demand values; this rounding will force us to scale capacities by at most a factor of 2. We first provide an algorithm for the case in which demand values are powers of $1/2$. The algorithm for the general case will then follow easily.

Theorem 3. *Let $\Pi = (G = (V, E, u), s, \mathcal{T})$ be a* UFP, *in which all demand values are of the form $2^{-\nu}$ with $\nu \in N$. There is an algorithm,* 2H_PARTITION, *which obtains, in polynomial time, an unsplittable routing such that the flow through any edge e is at most $zu_e + \rho_{\max} - \rho_{\min}$ where z is the optimum congestion.*

Remark: We first obtained a relative $(1 + \rho_{\max} - \rho_{\min})$-approximation for the case with demand values in $\{1/2, 1\}$ in [24]. Dinitz, Garg and Goemans [7] first extended that theorem to arbitrary powers of $1/2$. Their derivation uses a different approach.

Proof. We describe the algorithm 2H_PARTITION. The crux of the algorithm is a relaxed decision procedure that addresses the following question. Is there an unsplittable flow after scaling all the input capacities by x? The procedure will either return no or will output a solution where the capacity of edge e is at most $xu_e + \rho_{\max} - \rho_{\min}$. Embedding the relaxed decision procedure in a binary search completes the algorithm. See [16] for background on relaxed decision procedures. The following claim will be used to show the correctness of the procedure.

Claim. Given an UFP Π with capacity function r let Π' be the modified version of Π in which every edge capacity r_e has been rounded down to the closest multiple r'_e of $1/2^\lambda$. If the optimum congestion is 1 for Π, the optimum congestion is 1 for Π' as well.

Proof of Claim. Let G be the network with capacity function r and G' be the network with rounded capacities r'. The amount $r_e - r'_e$ of capacity will be unused by any optimal unsplittable flow in G. The reason is that the sum $\sum_{i \in S} 1/2^i$, for any finite multiset $S \subset N$ is equal to a multiple of $1/2^{i_0}$, $i_0 = \min_{i \in S}\{i\}$. ∎

We describe now the relaxed decision procedure. Let u' denote the scaled capacity function xu. Let $\rho_{\min} = 1/2^\lambda$. The relaxed decision procedure has at most $\lambda - \log(\rho_{\max}^{-1}) + 1$ iterations. During the first iteration, split every commodity in $\mathcal{T}$, called an *original commodity*, with demand $1/2^x$, $0 \le x \le \lambda$, into $2^{\lambda - x}$ *virtual commodities* each with demand $1/2^\lambda$. Call $\mathcal{T}_1$ the resulting set of commodities. Round down all capacities to the closest multiple of $1/2^\lambda$. By Theorem 1 we can find a fractional flow solution $f^0 \equiv g^1$, for $\mathcal{T}_1$, in which the flow through any edge is a multiple of $1/2^\lambda$. If this solution does not satisfy all demands the decision procedure returns no; by Claim 3.1 no unsplittable solution is possible without increasing the capacities u'. Otherwise the procedure continues as follows.

Observe that f^0 is a fractional solution for the set of original commodities as well, a solution with granularity $1/2^\lambda$. Let u_e^j denote the total flow through an edge after iteration j. The flow u_e^1 is trivially $f_e^0 \le u'_e \le u_e$. At iteration $i > 1$, we extract from g_{i-1}, using flow decomposition, the set of flow paths f^{i-1} used to route the set S_{i-1} of virtual commodities in $\mathcal{T}_{i-1}$ which correspond to original commodities in $\mathcal{T}$ with demand $1/2^{\lambda-i+1}$ or more. Pair up the virtual commodities in S_{i-1} corresponding to the same original commodity; this is possible since the demand of an original commodity is inductively an even multiple of the demands of the virtual commodities. Call the resulting set of virtual commodities $\mathcal{T}_i$. $\mathcal{T}_i$ contains virtual commodities with demand $1/2^{\lambda-i+1}$. The set of flow paths f^{i-1} is a fractional solution for the commodity set $\mathcal{T}_i$. By Lemma 1, we can find an unsplittable routing g^i for $\mathcal{T}_i$ such that the flow g_e^i through an edge e is at most $f_e^{i-1} + 1/2^{\lambda-i+2}$. Now the granularity has been doubled to $1/2^{\lambda-i+1}$. A crucial observation is that from this quantity, g_e^i, only an amount of at most $1/2^{\lambda-i+2}$ corresponds to new flow, added to e during iteration i. It is easy to see that the algorithm maintains the following two invariants.

INV1: After iteration i, all original commodities with demands $1/2^{\lambda-i+1}$ or less have been routed unsplittably.

INV2: After iteration $i > 1$, the total flow u_e^i though edge e, which is due to all i first iterations, satisfies the relation $u_e^i \le u_e^{i-1} + 1/2^{\lambda-i+2}$.

Given that $u_e^1 \le u'_e$ the total flow though e used to route unsplittably all commodities in $\mathcal{T}$ is at most

$$u'_e + \sum_{i=2}^{\lambda - \log(\rho_{\max}^{-1})+1} 1/2^{\lambda-i+2} = xu_e + \rho_{\max} - 1/2^\lambda = xu_e + \rho_{\max} - \rho_{\min}.$$

Therefore the relaxed decision procedure returns the claimed guarantee. For the running time, we observe that in each of the $l = \lambda - \log(\rho_{\max}^{-1}) + 1$ iterations, it appears that the number of virtual commodities could be as large as $k2^l$, where k is the number of the original commodities. There are two approaches to ensure polynomiality of our method. One is to consider only demands of value $1/2^\lambda > 1/n^d$ for some $d > 1$. It is known [24] that we can route the very small demands while affecting the approximation ratio by an $o(1)$ factor. The other approach relies on the observation that at iteration i, the set S_{i-1} of flow paths yielded by the flow decomposition has size $O(km)$ since it corresponds to a fractional solution for a subset of the original commodities. In other words, virtual commodities in S_{i-1} corresponding to the same original commodity have potentially been routed along the same path in g^i. Hence, by appropriately merging virtual commodities corresponding to the same original commodity, and increasing the granularity at different rates for distinct original commodities, we can always run an iteration on a polynomial-size set of virtual commodities. We opted to present a "pseudopolynomial" version of the decision procedure for clarity. □

Theorem 4. *Let* $\Pi = (G = (V, E, u), s, \mathcal{T})$ *be a* UFP. *There is an algorithm,* H_PARTITION, *which obtains in polynomial time a* $\min\{3-\rho_{\min}, 2+2\rho_{\max}-\rho_{\min}\}$ *approximation for congestion.*

Proof. We describe the algorithm H_PARTITION. Round up every demand to its closest power of $1/2$. Call the resulting UFP Π' with sink set $\mathcal{T}'$. Multiply all capacities by at most 2. Then there is a fractional feasible solution f' to Π'. Let u' be the capacity function in Π'. The purpose of the rounding is to be able to express later on the granularity as a power of $1/2$. The remainder of the proof consists of finding an unsplittable solution to Π'; this solution can be efficiently converted to an unsplittable solution to Π without further capacity increase. Such a solution to Π' can be found by the algorithm 2H_PARTITION from Theorem 3. Observe that the optimum congestion for Π' is at most z, the optimum congestion for Π. 2H_PARTITION will route through edge e flow that is at most

$$zu'_e + \rho'_{\max} - \rho'_{\min} \leq 2zu_e + \rho'_{\max} - \rho'_{\min},$$

where $\rho'_{\max}$ and $\rho'_{\min}$ denote the maximum and minimum demand values in Π'. But $\rho'_{\max} < 2\rho_{\max}$, $\rho'_{\max} \leq 1$, and $\rho'_{\min} \geq \rho_{\min}$ from the construction of Π'. Therefore $\rho'_{\max} - \rho'_{\min} \leq 1 - \rho_{\min}$ and $\rho'_{\max} - \rho'_{\min} \leq 2\rho_{\max} - \rho_{\min}$. Dividing the upper bound on the flow by $zu_e \geq 1$ yields the claimed guarantees on congestion. □

4 Heuristics used

The heuristics we designed fall into two major categories. We tested them both individually and in combination with each other.

The theoretical analysis of PARTITION and H_PARTITION is not affected if after finding the initial solution to the maximum flow relaxation, edges carrying zero flow, called *zero edges* are discarded. The *sparsification heuristic* implements this observation. The motivation behind the sparsification heuristic is to decrease the running time and also observe how the theoretical versions of the algorithms behave in practice since the zero edges are not exploited in the theoretical analysis. On random graph instances where maximum flow solutions use typically a small fraction of the edges, the sparsification heuristic resulted in drastic reductions of the running time. We observed that the approximation guarantee typically degrades by a small amount when a large number of edges is discarded. However the reduction on running time and space requirements makes the sparsification heuristic attractive for large instances. We also tested an intermediate approach where a randomly chosen fraction of the zero edges (on the average between 30 and 50 percent) is retained.

The *cost heuristic* invokes a minimum-cost flow subroutine when the original algorithm would invoke maximum flow to compute partial unsplittable solutions. In algorithm PARTITION we use minimum cost flow at Step 3 to find the partial unsplittable solution to each subproblem. In algorithm H_PARTITION minimum-cost flow is used to compute at iteration i the unsplittable routing g_i for the set T_i of virtual commodities (cf. proof of Theorem 3). That is we use a minimum-cost flow routine to successively improve the granularity of the fractional solution. The cost of an edge e is defined as an appropriate increasing function of the current utilization of e. Let λ denote the current congestion. For robustness we use the ratio z_e/λ in the calculation of the cost $c(e)$. We tested using functions that depend either linearly or exponentially on z_e/λ. The objective was to discourage the repeated use of edges which were already heavily utilized. Exponential cost functions tend to penalize heavily the edge with the maximum utilization in a given set of edges. Their theoretical success in multicommodity flow work [34, 18] motivated their use in our setting. In particular we let $c^L(e) = \beta_1 z_e/\lambda$ and $c^E(e) = 2^{\beta_2 z_e/\lambda}$ and tested both functions for different values of β_1, β_2. Observe that an edge with utilization 0 so far will incur a cost of 1 under c^E, while 0 seems a more natural cost assignment for this case. However experiments suggested that the c^E function behaved marginally better than $c^E - 1$ by imposing limited moderation on the introduction of new edges in the solution.

5 Experimental framework

Software and hardware resources. We conducted our experiments on a 400 MHz dual Pentium Pro PC with 500 MB of RAM and 130 MB swap space. None of the executions we report on here exceeded the system RAM or used the second system processor. The operating system was Linux, version 9. Our programs were written in C and compiled using `gcc`, version 2.7.2, with the `-O3` optimization option.

Codes tested. For the implementation of the maximum flow and minimum cost flow subroutines of our algorithms we used the codes described in [5] and [4] respectively. We also used the Cherkassky-Goldberg code to implement the flow decomposition step as described in Section 3. We assume integral capacities and demands in the unsplittable flow input. We test the following codes:

`h_prf`: this is algorithm PARTITION without sparsification.

`h_prf2`: this is algorithm PARTITION with intermediate sparsification. Zero cost edges are sampled with probability .3 and if the coin toss fails they are discarded.

`h_prf3`: this is algorithm PARTITION with all the zero edges discarded.

`3al`: this is algorithm H_PARTITION without sparsification. We implement a simplified version of algorithm H_PARTITION with theoretical guarantee 3 (in contrast to $3 - \rho_{\min}$ in Theorem 4).

`3al2`: this is algorithm H_PARTITION with all the zero edges discarded.

`c/h_prf10y`: version of `h_prf` with the cost heuristic. Linear cost function with slope $\beta_1 = 10^y$.

`c/h_prfexpy`: version of `h_prf` with the cost heuristic. Exponential cost function with coefficient $\beta_2 = y$.

`c3/3al10y`: version of `3al` with the cost heuristic. Linear cost function with slope $\beta_1 = 10^y$.

`c3/3alexpy`: version of `3al` with the cost heuristic. Exponential cost function with coefficient $\beta_2 = y$.

`c3_2/3al210y`: version of `3al2` with the cost heuristic. Linear cost function with slope $\beta_1 = 10^y$.

`c3_2/3al2expy`: version of `3al2` with the cost heuristic. Exponential cost function with coefficient $\beta_2 = y$.

Input classes. We generated data from four input classes. For each class we generated a variety of instances varying different parameters. The generators use randomness to produce different instances for the same parameter values. To make our experiments repeatable the seed of the pseudorandom generator is an input parameter for all generators. If no seed is given, a fixed default value is chosen. We used the default seed in generating all inputs. The four classes used follow.

`genrmf`. This is adapted from the GENRMF generator of Goldfarb and Grigoriadis [15, 2]. The input parameters are `a b c1 c2 k d`. The generated network has b frames (grids) of size $a \times a$, for a total of $a * a * b$ vertices. In each frame each vertex is connected with its neighbors in all four directions. In addition, the vertices of a frame are connected one-to-one with the vertices of the next frame via a random permutation of those vertices. The source is the lower left vertex of the first frame. Vertices become sinks with probability $1/k$ and their demand is chosen uniformly at random from the interval $[1, d]$. The capacities are randomly chosen integers from $(c1, c2)$ in the case of interframe edges, and $(1, c2 * a * a)$ for the in-frame edges.

`noigen`. This is adapted from the noigen generator used in [3, 29] for minimum-cut experimentation. The input parameters are `n d t p k`. The network has n nodes and an edge between a pair of vertices is included with probability $1/d$. Vertices are randomly distributed among t components. Capacities are chosen uniformly from a prespecified range $[l, 2l]$ in the case of intercomponent edges and from $[pl, 2pl]$ for intracomponent edges, p being a positive integer. Only vertices belonging to one of the $t-1$ components not containing the source can become sinks, each with probability $1/k$. The desired effect of the construction is for commodities to contend for the light intercomponent cuts. Demand for commodities is chosen uniformly from the range $[1, 2l]$.

`rangen`. This generates a random graph $G(n,p)$ with input parameters `n p c1 c2 k d`, where n is the number of nodes, p is the edge probability, capacities are in the range $(c1, c2)$, k is the number of commodities and demands are in the range $(0, d)$.

`satgen`. `satgen` first generates a random graph $G(n,p)$ as in `rangen` and then uses the following procedure to designate commodities. Two vertices s and t are picked from G and maximum flow is computed from s to t. Let v be the value of the flow. New nodes corresponding to sinks are incrementally added each connected only to t and with a randomly chosen demand value. The process stops when the total demand reaches v, the value of the minimum s-t cut or when the number of added commodities reaches the input parameter k, typically given as a crude upper bound.

`LA networks`. This is a synthetic family of networks generated by Deng et al. [6] as part of the AT&T capacity restoration project. Although this project does not deal with unsplittable flow, the data readily gives rise to multiple-source unsplittable flow instances. The networks are derived from applying planarity-preserving edge contractions to a street map of Los Angeles. Pairs of terminals are placed on vertices following an exponential distribution arising from actual traffic forecasts. We transformed the instances to single-source by picking the largest degree vertex as the source.

	2		6		12		25		50		100	
program name	cong.	time	cong.	time	cong.	time	cong.	time	cong.	time	cong.	time
3al	1.64	1	2.37	1	2.14	1	1.59	1	1.18	1	1.23	3
3al2	2.50	0	2.51	0	2.41	0	1.47	1	1.25	1	1.19	2
c/h_prf103	1.49	0	2.91	0	2.51	0	1.85	1	1.67	1	1.55	2
c/h_prfexp10	1.49	0	2.17	0	2.15	1	1.85	1	1.44	1	1.59	2
c3/3al103	2.56	0	2.29	1	1.96	2	1.46	2	1.25	4	1.16	7
c3/3alexp10	1.49	0	1.86	1	1.74	1	1.81	2	1.27	2	1.18	3
c3_2/3al2103	1.49	0	2.51	0	1.76	1	1.92	1	1.57	2	1.44	2
c3_2/3al2exp10	1.49	0	2.13	0	2.04	1	1.59	1	1.32	1	1.31	2
h_prf	2.00	0	2.47	0	2.51	0	2.15	0	1.50	0	1.45	1
h_prf2	2.00	0	2.47	0	2.51	0	2.14	0	1.50	1	1.41	2
h_prf3	2.00	0	2.75	0	2.20	0	1.74	0	1.36	0	1.46	1

Table 1. family noi_commodities: noigen 1000 1 2 10 i; 7981 edges; 2-unbalanced family; i is the expected percentage of sinks in the non-source component.

	2		4		8		16		32	
program name	cong.	time	cong.	time	cong.	time	cong.	time	cong.	time
3al	1.18	1	1.12	2	1.09	3	1.04	3	1.06	3
3al2	1.25	1	1.15	1	1.09	2	1.08	1	1.05	1
c/h_prf103	1.67	1	1.47	2	1.33	2	1.24	2	1.24	2
c/h_prfexp10	1.44	1	1.40	1	1.26	1	1.29	1	1.25	2
c3/3al103	1.25	4	1.22	6	1.13	7	1.12	5	1.07	6
c3/3alexp10	1.27	2	1.12	3	1.11	3	1.12	3	1.11	4
c3_2/3al2103	1.57	2	1.24	1	1.09	2	1.14	2	1.10	1
c3_2/3al2exp10	1.32	1	1.16	2	1.15	2	1.13	2	1.12	2
h_prf	1.50	1	1.42	1	1.30	1	1.22	1	1.24	2
h_prf2	1.55	1	1.40	1	1.22	2	1.23	2	1.18	2
h_prf3	1.36	0	1.22	1	1.38	0	1.28	1	1.20	1

Table 2. family noi_components: noigen 1000 1 i 10 50; 7981 edges; 2-unbalanced family; i is the number of components.

	32		64		128		256		512		1024	
program name	cong.	time	cong.	time	cong.	time	cong.	time	cong.	time	cong.	time
3al	5.33	0	7.00	1	8.00	2	8.50	10	7.50	42	8.00	185
3al2	7.50	0	5.50	0	6.00	1	7.00	4	7.50	22	9.00	93
c/h_prf103	7.00	0	7.50	0	6.50	0	7.50	4	8.00	16	8.00	70
c/h_prfexp10	7.50	0	4.50	0	7.00	1	7.00	4	7.50	16	8.00	70
c3/3al103	8.00	0	4.50	1	8.00	2	7.50	12	7.50	63	7.50	278
c3/3alexp10	7.50	0	7.00	1	7.50	3	7.50	12	7.50	55	7.50	243
c3_2/3al2103	7.50	0	4.50	0	7.00	1	7.50	5	7.50	22	7.50	95
c3_2/3al2exp10	7.50	0	4.50	0	7.00	1	7.50	5	7.50	22	8.00	95
h_prf	7.50	0	7.00	0	6.00	1	9.50	3	7.50	16	8.50	70
h_prf2	7.50	0	7.00	0	7.50	1	10.00	4	10.50	16	8.50	70
h_prf3	5.33	0	7.00	0	9.50	0	7.50	0	7.50	3	8.00	11

Table 3. family ran_dense: rangen -a i*2 -b 30 -c1 1 -c2 16 -k i -d 16; 1228, 4890, 19733, 78454, 313922, 1257086 edges; 16-unbalanced family; i*2 is the number of vertices.

	2		5		10		20		50		70	
program name	cong.	time	cong.	time	cong.	time	cong.	time	cong.	time	cong.	time
3al	1.88	7	1.91	22	1.65	44	1.35	94	1.16	259	1.14	372
3al2	1.74	4	1.97	13	1.55	24	1.31	49	1.18	143	1.14	201
c/h_prf103	2.41	6	2.49	13	2.10	27	1.81	63	1.66	178	1.65	234
c/h_prfexp10	2.67	6	2.43	11	2.09	22	1.79	53	1.74	149	1.71	201
c3/3al103	1.85	29	1.88	56	1.63	88	1.42	145	1.20	272	1.18	337
c3/3alexp10	1.80	21	1.96	30	1.64	42	1.46	70	1.24	175	1.18	235
c3_2/3al2103	1.80	5	1.97	18	1.68	31	1.47	58	1.26	150	1.20	207
c3_2/3al2exp10	1.92	8	2.00	20	1.67	31	1.37	54	1.21	144	1.22	196
h_prf	2.52	5	2.46	14	2.22	29	1.86	69	1.66	199	1.63	250
h_prf2	2.65	6	2.77	16	2.15	33	1.82	76	1.70	218	1.68	257
h_prf3	2.43	3	2.49	7	2.20	16	1.80	36	1.75	108	1.67	147

Table 4. family rmf_commDem: genrmf -a 10 -b 64 -c1 64 -c2 128 -k i -d 128; 29340 edges; 2-unbalanced family; i is the expected percentage of sinks in the vertices.

6 Experimental Results

In this section we give an overview of experimental results. We experimented on 13 different families and present data on several representative ones. Tables 1-6 provide detailed experimental results. The running time is given in seconds. The congestion achieved by algorithm PARTITION, on a balanced input, was typically less than 2.3 without any heuristics. The largest value observed was 2.89. Similarly for H_PARTITION the congestion, on a balanced input, was typically less than 1.65 and always at most 1.75. The theoretically better algorithm H_PARTITION does better in practice as well. To test the robustness of the approximation guarantee we relaxed, on several instances, the balance assumption allowing the maximum demand to be twice the minimum capacity or more, see Tables 1-5. The approximation guarantee was not significantly affected. Even in the extreme case when the balance assumption was violated by a factor of 16, as in Table 3, the best code achieved a congestion of 7.5. Performace on the smallest instances of the LA family (not shown here) was well within the typical values reported above. On larger LA instances, the exponential distribution generated an excessive number of commodities (on the order of 30,000 and up) for our codes to handle in the time frames we considered (a few minutes).

The sparsification heuristic proved beneficial in terms of efficiency as it typically cut the running time by half. One would expect that it would incur a degradation in congestion as it discards potentially useful capacity. This turned out not to be true in our tests. There is no clear winner in our experiments between `h_prf` and `h_prf3` or between `3al` and `3al2`. One possible explanation is that in all cases the same maximum flow code is running in subroutines, which would tend to route flow in a similar fashion.

	2		4		16		64	
program name	cong.	time	cong.	time	cong.	time	cong.	time
3al	2.26	0	2.18	0	1.48	9	1.21	214
3al2	1.82	0	2.72	0	1.46	4	1.20	102
c/h_prf103	2.45	0	3.10	0	1.88	5	1.68	140
c/h_prfexp10	2.45	0	3.01	0	1.98	4	1.73	117
c3/3al103	1.42	0	2.03	1	1.53	13	1.24	225
c3/3alexp10	1.72	0	2.11	1	1.57	7	1.22	145
c3_2/3al2103	1.50	0	2.40	1	1.70	5	1.32	113
c3_2/3al2exp10	1.61	0	2.17	0	1.59	5	1.27	112
h_prf	1.85	0	3.56	0	2.05	6	1.72	154
h_prf2	1.85	0	3.56	0	1.94	6	1.75	171
h_prf3	2.38	0	3.12	0	1.92	3	1.72	87

Table 5. family rmf_depthDem: genrmf -a 10 -b i -c1 64 -c2 128 -k 40 -d 128; 820, 1740, 7260, 29340 edges; 2-unbalanced family; i is the number of frames.

In contrast, the cost heuristic achieved modest improvements with respect to the theoretical versions of the algorithms. The gains were more tangible when the cost heuristic was applied to H_PARTITION which solves subproblems in an incremental fashion using information from the subproblems solved previously. On instances where `3al` or `3al2` achieved a congestion of around 1.7 the gains were of the order of 5-10% depending on the actual cost function used. On instances where H_PARTITION did really well, achieving a congestion close to 1 the cost heuristic offered no improvement or even deteriorated the approximation.

Typical values of the cost function coefficients that achieved the best results were $\beta_1 = 10^3$, 10^4 for the linear case and $\beta_2 = 10$ or 25 for the exponential case. In this abstract we report results for $\beta_1 = 10^3$ and $\beta_2 = 10$. The exponential function performed slightly better than the linear function, with the gap closing as the network size or the number of commodities increased, in which case the actual numerical values for costs became less important.

	1		2		6		12		25		50	
program name	cong.	time	cong.	time	cong.	time	cong.	time	cong.	time	cong.	time
3al	1.55	1	1.56	1	1.67	5	1.64	11	1.64	27	1.70	64
3al2	1.45	0	1.67	1	1.78	3	1.57	6	1.67	15	1.75	35
c/h_prf103	1.80	1	2.11	0	2.22	2	2.15	4	2.33	11	2.33	26
c/h_prfexp10	1.92	0	2.11	1	2.11	2	2.15	5	2.33	11	2.33	24
c3/3al103	1.44	1	1.64	2	1.44	7	1.67	14	1.56	30	1.56	130
c3/3alexp10	1.30	1	1.33	2	1.56	7	1.57	14	1.56	34	1.56	78
c3_2/3al2103	1.15	1	1.40	1	1.44	3	1.56	6	1.56	15	1.56	34
c3_2/3al2exp10	1.27	0	1.44	1	1.44	3	1.56	6	1.56	14	1.57	34
h_prf	1.92	0	2.11	1	2.11	2	2.33	4	2.22	11	2.33	27
h_prf2	1.92	0	2.22	0	2.33	2	2.56	4	2.60	12	2.89	28
h_prf3	2.00	0	2.20	1	2.18	1	2.33	1	2.22	3	2.42	9

Table 6. family sat_density: satgen -a 1000 -b i -c1 8 -c2 16 -k 10000 -d 8; 9967, 20076, 59977, 120081, 250379, 500828 edges; 22, 61, 138, 281, 682, 1350 commodities; i is the expected percentage of pairs of vertices joined by an edge.

In summary, algorithm H_PARTITION achieved significantly better approximations than algorithm PARTITION on the instances we examined. The sparsification heuristic can reduce drastically the running time. Running the cost heuristic can provide additional improvement on congestion at the expense of an increase in running time.

7 Future work

There are several directions to pursue in future work. The main task would be to implement the new 2-approximation algorithm of Dinitz et al. [7] and see how it behaves in practice. Testing on data with real network characteristics is important if such instances can be made available. Designing new generators that push the algorithms to meet their theoretical guarantees will enhance our understanding of the problem. Finally, a problem of minimizing makespan on parallel machines [27] reduces to single-source unsplittable flow on a two-level graph with simple topology. It would be interesting to see how the unsplittable flow algorithms behave on actual instances of this problem.

Acknowledgement. Thanks to Aravind Srinivasan for useful discussions and for motivating this project. An early version of a $G(n,p)$ generator was written by Albert Lim. We also wish to thank Michel Goemans for sending us a copy of [7]. We are grateful to Jeff Westbrook for giving us access to the LA networks. The first author thanks Peter Blicher and Satish Rao for useful discussions.

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.
2. Tamas Badics. GENRMF. `ftp://dimacs.rutgers.edu/pub/netflow/generators/-network/ genrmf/`, 1991.
3. C. Chekuri, A. V. Goldberg, D. Karger, M. Levine, and C. Stein. Experimental study of minimum cut algorithms. In *Proc. of 8th SODA*, 324–333, 1997.
4. B. V. Cherkassky and A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22:1–29, 1997.
5. B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.
6. M. Deng, D. F. Lynch, S. J. Phillips, and J. R. Westbrook. Algorithms for restoration planning in a telecommunications network. In *ALENEX99*, 1999.
7. Y. Dinitz, N. Garg, and M. Goemans. On the single-source unsplittable flow problem. In *Proc. of 39th FOCS*, 290–299, November 1998.
8. P. Elias, A. Feinstein, and C. E. Shannon. Note on maximum flow through a network. *IRE Transactions on Information Theory IT-2*, 117–199, 1956.
9. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canad. J. Math.*, 8:399–404, 1956.
10. A. Frank. Packing paths, cuts and circuits - a survey. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows and VLSI-Layout*, 49–100. Springer-Verlag, Berlin, 1990.
11. N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–20, 1997.
12. A. V. Goldberg, J. D. Oldham, S. A. Plotkin, and C. Stein. An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow. In *Proc. of the 6th Conference on Integer Programming and Combinatorial Optimization*, 338–352, 1998. Published as Lecture Notes in Computer Science 1412, Springer-Verlag",.
13. A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.
14. A.V. Goldberg and S. Rao. Beyond the flow decomposition barrier. In *Proc. of the 38th Annual Symposium on Foundations of Computer Science*, 2–11, 1997.
15. D. Goldfarb and M. Grigoriadis. A computational comparison of the Dinic and Network Simplex methods for maximum flow. *Annals of Operations Research*, 13:83–123, 1988.
16. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

17. P. Klein, A. Agrawal, R. Ravi, and S. Rao. Approximation through multicommodity flow. In *Proc. of the 31st Annual Symposium on Foundations of Computer Science*, 726–737, 1990.
18. P. Klein, S. A. Plotkin, C. Stein, and É. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23(3):466–487, June 1994.
19. J. M. Kleinberg. *Approximation algorithms for disjoint paths problems.* PhD thesis, MIT, Cambridge, MA, May 1996.
20. J. M. Kleinberg. Single-source unsplittable flow. In *Proc. of the 37th Annual Symposium on Foundations of Computer Science*, 68–77, October 1996.
21. J. M. Kleinberg and É. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. In *Proc. of the 27th Annual ACM Symposium on Theory of Computing*, 26–35, 1995.
22. J. M. Kleinberg and É. Tardos. Disjoint paths in densely-embedded graphs. In *Proc. of the 36th Annual Symposium on Foundations of Computer Science*, 52–61, 1995.
23. S. G. Kolliopoulos. *Exact and Approximation Algorithms for Network Flow and Disjoint-Path Problems.* PhD thesis, Dartmouth College, Hanover, NH, August 1998.
24. S. G. Kolliopoulos and C. Stein. Improved approximation algorithms for unsplittable flow problems. In *Proc. of the 38th Annual Symposium on Foundations of Computer Science*, 426–435, 1997.
25. T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50:228–243, 1995.
26. T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proc. of the 29th Annual Symposium on Foundations of Computer Science*, 422–431, 1988.
27. J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.
28. T. Leong, P. Shor, and C. Stein. Implementation of a combinatorial multicommodity flow algorithm. In D. S. Johnson and C. C. McGeoch, editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Volume 12, Network Flows and Matching*. October 1991.
29. Hiroshi Nagamochi, Tadashi Ono, and Toshihde Ibaraki. Implementing an efficient minimum capacity cut algorithm. *Mathematical Programming*, 67:325–241, 1994.
30. CPLEX Optimization, Inc. Using the CPLEX callable library. Software Manual, 1995.
31. P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.
32. N. Robertson and P. D. Seymour. Outline of a disjoint paths algorithm. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows and VLSI-Layout*. Springer-Verlag, Berlin, 1990.
33. A. Schrijver. Homotopic routing methods. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows and VLSI-Layout*. Springer-Verlag, Berlin, 1990.
34. F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37:318 – 334, 1990.