

A Fast Approximation Scheme for Fractional Covering Problems with Box Constraints*

Lisa Fleischer[†]

September 6, 2004

Abstract

We present the first combinatorial approximation scheme that yields a *pure* approximation guarantee for linear programs that are either covering problems with upper bounds on variables, or their duals. Existing approximation schemes for mixed covering and packing problems do not simultaneously satisfy packing and covering constraints exactly. We present the first combinatorial approximation scheme that returns solutions that simultaneously satisfy general positive covering constraints and upper bounds on variable values. For input parameter $\epsilon > 0$, the returned solution has positive linear objective function value at most $1 + \epsilon$ times the optimal value. The general algorithm requires $O(\epsilon^2 m \log(c^\top u))$ iterations, where c is the objective cost vector, u is the vector of upper bound values, and m is the number of variables. Each iteration uses an oracle that finds an (approximately) most violated constraint.

A natural set of problems that our work addresses are linear programs for various network design problems: generalized Steiner network, vertex connectivity, directed connectivity, capacitated network design, group Steiner forest. The integer versions of these problems are all NP-hard. For each of them, there is an approximation algorithm that rounds the solution to the corresponding linear program relaxation. If the LP solution is not feasible, then the corresponding integer solution will also not be feasible. Solving the linear program is often the computational bottleneck in these problems, and thus a fast approximation scheme for the LP relaxation means faster approximation algorithms.

For these applications, we introduce a new modification of the push-relabel maximum flow algorithm that allows us to perform each iteration in amortized $O(|E| + |V| \log |V|)$ time, instead of one maximum flow per iteration that is implied by the straight forward adaptation of our general algorithm. In conjunction with an observation that reduces the number of iterations to $|E| \log |V|$ for $\{0, 1\}$ constraint matrices, the modification allows us to obtain an algorithm that is faster than existing exact or approximate algorithms by a factor of at least $O(|E|)$ and by a factor of $O(|E| \log |V|)$ if the number of demand pairs is $\Omega(|V|)$.

1 Introduction

Problem Statement. A *mixed positive packing and covering program (MPCP)* is a linear program described by nonnegative, integer $n \times m$ matrix A ; nonnegative, integer $n' \times m$ matrix P ;

*An extended abstract of this paper appears in [12].

[†]Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213, USA, Email: lkf@andrew.cmu.edu. Supported in part through NSF CAREER Award CCR-0049071, NSF award EIA-0049084, and the IBM T. J. Watson Research Center.

and nonnegative integer vectors b , c , and u . It has the form

$$\min c^\top x \tag{1}$$

$$\text{s.t. } Ax \geq b \tag{2}$$

$$Px \leq u \tag{3}$$

$$x \geq 0 \tag{4}$$

The problem without (3) is a *fractional covering problem*. The problem without (2) and with (1) changed to $\max c^\top x$ is a *fractional packing problem*.

Mixed packing and covering programs can be solved exactly using standard LP solvers. However, these solvers are often slow for large problems. In response to this, approximation schemes based on Frank-Wolfe methods [16] for MPCP's were proposed in the early 1990's [22, 23, 32], generalizing ideas that appeared earlier in approximation schemes for fractional multicommodity flow problems [29, 30, 34]. Since this initial work, there has been substantial progress made on refining the algorithms. The result has been substantial reduction in complexity of both the run time and the analysis [13, 18, 24, 27, 28, 33, 36, 37]. Empirically, approximate LP algorithms have proved to be effective in obtaining "warm starts" for exact LP solvers, and have thus enabled significant reductions in run times [2, 3, 4, 20] even when exact solutions are desired. A survey of the field is contained in [4].

All of the above-mentioned approximation schemes that consider *mixed* positive packing and covering problems find solutions that are only *approximately* feasible: these schemes find a solution of a fixed value that either satisfies $Ax \geq b$ and $Px \leq (1 + \epsilon)u$ or satisfies $Ax \geq b/(1 + \epsilon)$ and $Px \leq u$. Since x does not satisfy all constraints exactly, the value of x can be arbitrarily far from the value of a true solution.

Our Contribution. In Section 2, we present an approximation scheme that finds exactly feasible solutions within value of $1 + \epsilon$ times the optimal value for the following primal and dual linear programs in variables x , and (y, z) .

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^m c(j)x(j) \\ \text{subject to} & \forall 1 \leq i \leq n : \sum_j A(i, j)x(j) \geq b(i) \\ & \forall 1 \leq j \leq m : x(j) \leq u(j) \\ & \forall 1 \leq j \leq m : x(j) \geq 0 \end{array} \tag{P}$$

The corresponding dual LP:

$$\begin{aligned}
& \mathbf{maximize} \\
& D(y, z) := \sum_{i=1}^n b(i)y(i) - \sum_{j=1}^m u(j)z(j) \\
& \mathbf{subject\ to} \\
& \forall 1 \leq j \leq m : \sum_{i=1}^n A(i, j)y(i) - z(j) \leq c(j) \\
& \forall 1 \leq i \leq n : y(i) \geq 0 \\
& \forall 1 \leq j \leq m : z(j) \geq 0.
\end{aligned} \tag{D}$$

Our algorithm uses $O(\epsilon^{-2}m \log(c^\top u))$ calls to an oracle that given a value α and a variable vector x either returns a constraint satisfying $A(i)x/b(i) < \alpha$ or confirms that there is no such constraint. Thus, our algorithm can approximately solve linear programs of the above form with an exponential number of covering constraints that are described implicitly by a polynomial time oracle.

A natural set of problems that our work addresses are linear programs for various network design problems. For these problems, a provably good integer solution can be found by rounding the solution to an LP relaxation. Examples include generalized Steiner network [26] (also called survivable network design), element connectivity [15], vertex connectivity [7, 14], directed connectivity [31], capacitated network design [5], and group Steiner forest [19].

The approximation guarantees obtained by these algorithms are achieved in comparison with the LP lower bound. If the initial fractional point is only a solution to the relaxed LP (i.e. it does not exactly satisfy both covering and box constraints), then the LP bound can be arbitrarily far off from the integer solution, and using the fractional point may lead to an infeasible solution. For some of these problems, approximation algorithms exist that do not round LP. Even in this case, the LP provides a lower bound that could be useful in practice to provide a provable approximation guarantee that is better than the generic guarantee promised by the approximation algorithm.

Our algorithm builds on the approximation schemes of Garg and Könemann [18] and Fleischer [13]. One key innovation of our algorithm is to use scaling phases to control the increase in variable values. At the end of the α -phase, the algorithm has a solution x that satisfies $Ax \geq \alpha b$ and $x \leq \alpha u$. Thus, the solution x/α is a feasible solution to MPCP. Throughout the α -phase, the solution x satisfies $Ax \geq \alpha b/(1+\epsilon)$ and $x \leq \alpha u$. Our algorithm works by using an oracle that given the current iterate x , finds a constraint i such that $\sum_{1 \leq j \leq m} A(i, j)x(j) < \alpha b(i)$. It then increases x to increase the value of the left hand side of this constraint. This is iterated. One key lemma is that if the initial problem is feasible and $A(i)x < \alpha b(i)$, then there is some index j such that $x(j) < \alpha u(j)$ and $A(i, j) > 0$. Thus our algorithm never gets stuck. In addition, with each increase, there is some index j such that the algorithm either adds a substantial amount to $x(j)$, or $x(j)$ hits its phase upper bound of $\alpha u(j)$. After at most $O(\frac{1}{\epsilon^2} \log(c^\top u))$ phases and $O(\frac{1}{\epsilon^2} m \log(c^\top u))$ iterations, the algorithm encounters a feasible, ϵ -optimal¹ solution and a proof of its approximate optimality.

To obtain the approximation guarantee, we introduce a new dual update procedure that decreases the value of a dual solution to compensate for not increasing primal variables that are currently at their upper bound. The dual LP of our problem is not a mixed packing and covering LP, and has

¹A feasible solution to a minimization problem is ϵ -optimal if its value is at most $(1+\epsilon)$ times the optimal value. A feasible solution to a maximization problem is ϵ -optimal if its value is at least the optimal value divided by $(1+\epsilon)$.

negative coefficients in the objective function. Thus, this requires a new method to update dual variables.

In Section 3 we show how to remove dependency of number of phases and iterations on c when A is a $\{0, 1\}$ matrix, by replacing $\log(c^\top u)$ with $\log \frac{mU}{\epsilon}$, where $U = \min\{\|b\|_\infty, \|u\|_\infty\}$.

For the special case of survivable network design (also called generalized Steiner problem), we show how to implement all of the iterations that occur in one phase in $O(mn \min\{n, k\} \log n)$ time, plus linear work per iteration, where m is the number of edges, n is the number of vertices in the network, and k is the number of pairs of vertices with positive connectivity requirement. Without modification to our general algorithm, the work per iteration is $\min\{n, k\}$ maximum flow computations. This can be reduced to one maximum flow per iteration using ideas from [13]. In Section 4, we show how to reduce this further to amortized $m + n \log n$ time per iteration. Our key insight involves a new modification to the push-relabel maximum flow algorithm to show that the work per phase can be performed in the same asymptotic time as one maximum flow computation. Thus we obtain an algorithm which is faster than existing algorithms by a factor of at least $O(\min\{m, nk\})$, and faster by a factor of $O(m \log n)$ if the number of demand pairs is $\Omega(n)$.

The modified push-relabel algorithm also enables us to obtain improvements in oracle complexity for element connectivity, vertex connectivity, directed connectivity, capacitated network design, and group Steiner forest.

Related Work. The symmetric problem, the positive packing problem with lower bounds on variables, is easily transformed into a pure positive packing problem by a change of variables. With an approximate oracle that given x finds a constraint $P(j)x/u(j) \geq (1 - \epsilon) \max_i P(i)x/u(i)$, an ϵ -optimal solution can be found using earlier work. For instance, the algorithms of Grigoriadis, et al. [24] and Young [37] solve this problem by solving the dual pure covering problem, where the (perhaps exponential number of) variables are handled implicitly via the oracle.

Independently and concurrently with our own work, Garg and Khandekar [17] develop a very different approximation scheme for linear program relaxations of problems on clutters. These yield exact polynomial time approximation schemes for problems when A is restricted to be a 0-1 matrix. This captures both the survivable network design problem, and the group Steiner problem, but not capacitated network design problems. Their algorithm does not run in polynomial time for general matrices A .

In addition, their algorithm specialized to the case of SND requires $O(m \min\{n, k\} \log^2 k)$ iterations, where each iteration requires a minimum cost flow computation. In contrast, for the special case of SND our algorithm requires only $O(m \log n)$ iterations, where the amortized time per iteration is $O(n \min\{n, k\} \log n)$, and an additional $m + n$ maximum spanning tree computations. One of the improvements we obtain in the work per iteration is possible due to the fact that the oracle-problem needed to be solved each iteration is a maximum flow computation. It turns out that for some sequences of iterations, these flow computations are closely related. We show how the work performed during these closely related flow problems can be amortized. It is not known how to amortize the work for similarly closely related minimum cost flow problems, which is the oracle-problem in [17].

1.1 Preliminaries and Notation.

For vector c , $c(j)$ denotes the j -th component of c . For a matrix A , $A(i)$ denotes the i -th row of A and $A(i, j)$ denotes the j^{th} component of the i^{th} row of A . For a specified A, b, u , and c , OPT denotes the value of the optimal solution to **(P)** and **(D)**. Denote by $\mathbf{1}$ the vector of all 1's. For two vectors v and w , vw and $v^\top w$ both denote the dot product of v and w .

We assume, without loss of generality, that $\min_j c(j) > 0$. If not, all variables in the set $\{j | c(j) = 0\}$ are included in the solution at value $u(j)$, and the problem is reduced by removing these variables and modifying A and b accordingly. Thus, we also assume $\text{OPT} > 0$, since otherwise the problem is trivially solved. Without loss of generality, we assume $\min_j c(j) = 1$ by dividing c by the minimum (nonzero) component.

2 Fractional Covering with Variable Upper Bounds

In this section, we show how to obtain exactly feasible solutions x and (y, z) to both **(P)** and **(D)** with $P(x)/D(y, z) \leq 1 + \epsilon$, when A may be given implicitly by an oracle that given x and α either returns a constraint satisfying $A(i)x/b(i) < \alpha$ or confirms that there is no such constraint.

2.1 The Approximation Scheme.

Algorithm `FractionalCoverUB`(c, A, b, ϵ) (appearing in Figure 1) operates in α -phases, during which α is fixed. This algorithm differs from existing packing and covering schemes in that for all j , $x \leq \alpha u$ throughout the α -phase. This restriction is necessary so that division by α yields a feasible solution. We show that by proceeding in α -phases, this restriction does not increase the number of iterations by too much.

An α -phase starts with $x(j) \leq u(j)\alpha/(1 + \epsilon)$ for all $1 \leq j \leq m$ and $\frac{A(i)x}{b(i)} \geq \frac{\alpha}{(1 + \epsilon)}$ for all $1 \leq i \leq n$. It iteratively picks a constraint p with $\frac{A(p)x}{b(p)} < \alpha$ (lines (2) and (13)) and increases $A(p)x$ while maintaining $x \leq \alpha u$ for all j . The phase ends when $\frac{A(i)x}{b(i)} \geq \alpha$ for all $1 \leq i \leq n$. Thus, at the end of the α -phase, x/α is a feasible primal solution. The next phase starts by setting $\alpha = (1 + \epsilon)\alpha$.

To maintain $x \leq \alpha u$ while increasing $A(p)x$, `FractionalCoverUB` increases only variables in the set $Q(p) = \{1 \leq j \leq m | A(p, j) > 0 \text{ and } x(j) < u(j)\alpha\}$ (line (6)). The amount of increase (specified in line (7)) is determined so that at least one variable $x(j)$ either increases by a $(1 + \epsilon)$ factor or hits the phase upper bound of $\alpha u(j)$. We show below in Lemma 2.5 that if $Q(p)$ is empty, then the problem is infeasible. Thus, some variable is increased in each iteration.

The general method of variable increase is a Frank-Wolfe method in the sense that there is an implicit, underlying potential function $\sum_i e^{A(i)(x/\alpha c)}$ that the algorithm works to reduce [4, 16]. The primal variable update of $(1 + \epsilon\eta)$ is an approximation to $(1 + \epsilon)^\eta$, which is an approximation to $e^{\epsilon\eta}$. The analysis of our algorithm uses these relationships (see Section 2.2). In fact, the variable update in line (9) could be replaced with $x \leftarrow x e^{\epsilon\eta A(p, j)/c(j)}$ and the same asymptotic guarantees hold. In practice, $x \leftarrow x e^{\epsilon\eta A(p, j)/c(j)}$ may yield better performance.

To prove ϵ -optimality of the final solution x^* , the algorithm also maintains dual solutions (y, z) . The algorithm returns primal and dual solutions that are feasible, and have objective function value

within a $(1 + \epsilon)$ factor of each other, proving their ϵ -optimality via LP duality theory. Each increase to the primal solution x is balanced with an increase to (y, z) so that the objective function value of one can be related to the other. Since (\mathbf{P}) is not a pure covering problem, the dual problem is not a pure packing problem. The algorithm incorporates a new dual update procedure to maintain this balance. These lines (10)-(12) and (17), are not necessary to find an ϵ -optimal solution to (\mathbf{P}) , but they are helpful to prove ϵ -optimality.

Given (y, z) , let β be the minimum quantity such that $(y/\beta, z/\beta)$ is feasible for (\mathbf{D}) . Thus, $\beta := \max_j [\sum_i A(i, j)y(i) - z(j)] / c(j)$. In each iteration, one y -component is increased (line (11)). This increases the left hand side value of some dual constraints. Each time the left hand side of dual constraint j is increased by $c(j)$ (implying that if j determines β then the new β is increased by 1), this increase is balanced in (\mathbf{P}) by multiplying $x(j)$ by at least $(1 + \epsilon)$. Thus, if $y(p)$ is increased and $A(p, j) > 0$, but $x(j)$ is already at its upper bound, `FractionalCoverUB` increases $z(j)$ by $A(p, j)$ times the increase to $y(p)$ (lines (11)-(12)) so that these two increases cancel each other and the left hand side of dual constraint j remains unchanged.

Initially $x(j) = u(j)\delta$ and $(y, z) \equiv 0$. Denote by x^0 this initial value of x , and define $\alpha(0)$ as $\delta \min_i \frac{A(i)^T u}{b(i)} \geq \delta$. Then $u(j)\alpha(0) \geq u(j)\delta \geq x^0(j)$, so that $x^0/\alpha(0)$ is feasible for (\mathbf{P}) .

The algorithm terminates when $c^T x \geq 1$. (Denote this last phase by t .) The actual value of $c^T x$ is not important; what is important is its relation to the initial value of δ so that this implies that there have been a sufficient number of iterations. We choose δ later to yield that the number of iterations will be $O(m \frac{\log(c^T u)}{\epsilon^2})$.

2.2 Algorithm Analysis.

In this section, we prove the following theorem, which is one of the main results of this paper:

Theorem 2.1 *Algorithm FractionalCoverUB returns feasible, ϵ -optimal solutions to (\mathbf{P}) and (\mathbf{D}) , and requires at most $O(\epsilon^{-2}m \log(c^T u))$ oracle calls.*

Let x_l^k, y_l^k , and z_l^k denote the vectors x, y , and z at the end of the l^{th} iteration in the k^{th} phase. Let x^k, y^k , and z^k be the corresponding vectors at the end of the k^{th} phase. Let $\alpha(k)$ be the value of α in the k^{th} phase. The first lemma shows that x stays bounded by α in the α -phase.

Lemma 2.2 *Throughout FractionalCoverUB, $x(j) \leq u(j)\alpha$ for all $1 \leq j \leq m$.*

Proof: This is true at the start of the algorithm by the initial setting of x and α . By definition of η in line (7) of `FractionalCoverUB`, after incrementing $x(j)$, we have that $\frac{x(j)}{u(j)} \leq \frac{x(j)}{u(j)} (1 + \epsilon \frac{\eta}{c(j)/A(p, j)}) \leq \frac{x(j)}{u(j)} (1 + \epsilon \frac{u(j)\alpha - x(j)}{\epsilon x(j)}) = \frac{x(j)}{u(j)} + \alpha - \frac{x(j)}{u(j)} = \alpha$. ■

In the next two lemmas, we establish a bound on the number of α -phases by proving a significant increase in α between phases.

Lemma 2.3 *If there are t phases, then for all $1 \leq k < t$, $\alpha(k + 1) \geq (1 + \epsilon)\alpha(k)$.*

FractionalCoverUB(c, A, b, u, ϵ)

Input: cost vector $c \in \mathbb{R}^m$, upper bound vector u , n constraints $Ax \geq b$ given by oracle.

Output: ϵ -optimal primal and dual solutions x and (y, z) .

- (1) Initialize $x(j) \leftarrow u(j)\delta$, $(y, z) \equiv 0$, $x^* \leftarrow x$, $\alpha^* \leftarrow \operatorname{argmin}_i A(i)x/b(i)$ ².
- (2) $p \leftarrow \operatorname{argmin}_i A(i)x/b(i)$ – oracle call –
- (3) **while** $c^\top x < 1$,
- (4) $\alpha \leftarrow (1 + \epsilon)A(p)x/b(p)$ – α -phase –
- (5) **while** $A(p)x/b(p) < \alpha$ and $c^\top x < 1$, – iteration –
- (6) $Q(p) = \{1 \leq j \leq m \mid x(j) < u(j)\alpha\}$.
- (7) $\eta = \min_{j \in Q(p)} \frac{c(j)}{A(p,j)} \min\{1, \frac{u(j)\alpha - x(j)}{\epsilon x(j)}\}$ – increment amount –
- (10) $y_p \leftarrow y_p + \eta$
- (8) For $j \in Q(p)$,
- (9) $x(j) \leftarrow x(j)(1 + \epsilon \frac{\eta}{c(j)/A(p,j)})$
- (11) For $j \notin Q(p)$,
- (12) $z(j) \leftarrow z(j) + \eta A(p, j)$
- (13) $p \leftarrow \operatorname{argmin}_i A(i)x/b(i)$ – oracle call –
- (14) **end while**
- (15) If $c^\top x/\alpha < c^\top x^*/\alpha^*$, then $x^* \leftarrow x$, $\alpha^* \leftarrow \alpha$. – keep best solution –
– end α -phase –
- (16) **end while**
- (17) $\delta \leftarrow (1 + \epsilon)((1 + \epsilon)c^\top u)^{-1/\epsilon}$.
- (18) **return** x^*/α^* , $\frac{\epsilon}{\ln \frac{1+\epsilon}{\delta}}(y, z)$.

Figure 1: An FPTAS for general positive covering problem with variable upper bounds.

Proof: All phases k except possibly the last phase t terminate when when $\min A(p)x/b(p) \geq \alpha(k)$. At the start of phase $k + 1$, $\alpha(k + 1)$ is set to $(1 + \epsilon) \min A(p)x/b(p) \geq (1 + \epsilon)\alpha(k)$ in line (4). ■

Lemma 2.4 *After the completion of $t := \lceil \log_{1+\epsilon} \frac{1}{\delta} \rceil$ phases, there is at least one j with $x^t(j) \geq 1$. Thus $c^\top x^t \geq 1$.*

Proof: At start, $\alpha(0) = \delta \frac{A(p)^\top u}{b(p)}$ for some constraint p satisfying $\frac{A(p)^\top u}{b(p)} = \min_i \frac{A(i)^\top u}{b(i)}$. At phase t , we have that

$$\begin{aligned}
\frac{A(p)^\top \mathbf{1}}{b(p)} &\leq \frac{A(p)^\top u}{b(p)} &= \alpha(0) \frac{1}{\delta} \\
& &= \alpha(0)(1 + \epsilon)^{\lceil \log_{1+\epsilon} 1/\delta \rceil} \\
& &\leq \alpha(t) && \text{by Lemma 2.3} \\
& &\leq \frac{A(i)^\top x^t}{b(i)} && \text{for all } i.
\end{aligned}$$

Substituting p in for i in this last line, it follows that there is at least one index in the set $\{j \mid A(p, j) > 0\}$ that satisfies $x^t(j) \geq 1$. Since $\min_j c(j) \geq 1$, we have that $c^\top x^t \geq 1$. ■

We use the preceding lemmas to help bound the number of iterations. The following key lemma is useful to show that the value of at least one variable $x(j)$ strictly increases in each iteration.

Lemma 2.5 *If (P) is feasible, then $Q(p) \neq \emptyset$ inside the while loop of FractionalCoverUB.*

Proof: If the problem is feasible, then $b(i) \leq \sum_j u(j)A(i, j)$ for all constraints i . Inside while loop, we have that

$$b(p)\alpha > A(p)x \geq \sum_{j \notin Q(p)} A(p, j)x(j) \geq \alpha \sum_{j \notin Q(p)} u(j)A(p, j),$$

so that $b(p) > \sum_{j \notin Q(p)} u(j)A(p, j)$. Thus, if $Q(p) = \emptyset$, this contradicts feasibility. ■

Lemma 2.6 *The algorithm terminates after at most $2m \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ iterations.*

Proof: By Lemma 2.5, at least one variable is incremented in each iteration. With each increment, the variable $x(j)$ that defines η in line (7) either increases by a factor of $1+\epsilon$, or hits $u(j)\alpha$. Variable $x(j)$ starts equal to $u(j)\delta$, and never exceeds $(1+\epsilon)/c(j)$, by the termination condition. Since x is nondecreasing, $x(j)$ can be multiplied by $(1+\epsilon)$ at most $\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ times. Since α increases at most $\log_{1+\epsilon} \frac{1}{\delta}$ times, the number of times any variable $x(j)$ can hit $u(j)\alpha$ is bounded by this same quantity. Summing this over all $1 \leq j \leq m$ yields the lemma. ■

We prove in Lemma 2.8 the feasibility of the solution to (P) returned by FractionalCoverUB. For this we require the following technical lemma.

Lemma 2.7 *Given $f_0 = 0$ and $g_0 > 0$ and sequence $\{a_1, a_2, \dots, a_t\}$ satisfying $0 \leq a_r \leq 1$ for all r , suppose $f_r \leq f_{r-1} + a_r$ and $g_r \geq g_{r-1}(1 + a_r\epsilon)$. Then $g_r/g_0 \geq (1 + \epsilon)^{f_r}$ for all r .*

Proof: Initially $g_0/g_0 = 1 = (1 + \epsilon)^{f_0}$. Since $1 + a\epsilon \geq (1 + \epsilon)^a$ for all $0 \leq a \leq 1$, the product $\prod_r (1 + a_r\epsilon) \geq (1 + \epsilon)^{\sum_r a_r}$, when $0 \leq a_r \leq 1$ for all r . Thus, $g_r/g_0 \geq \prod_{i=1}^r (1 + a_i\epsilon) \geq (1 + \epsilon)^{\sum_{i=1}^r a_i} \geq (1 + \epsilon)^{f_r}$. ■

Lemma 2.8 *The final dual solution (y^t, z^t) divided by $\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ is a feasible solution for (D).*

Proof: By our update of $x(j), y(p)$ and $z(j)$ in one iteration, when the ratio $\frac{\sum_i A(i, j)y(i) - z(j)}{c(j)}$ increases by an additive term of $\frac{\eta}{c(j)/A(p, j)} \leq 1$, the quantity $x(j)$ is multiplied by $(1 + \frac{\eta}{c(j)/A(p, j)}\epsilon)$. By the termination criteria of $c^\top x > 1$, we have that $x^t(j)/x^0(j) \leq \frac{1+\epsilon}{\delta}$. Applying Lemma 2.7 with $\{g_r\}_r = \{x_l^k(j)\}_{k, l}$, $\{f_r\}_r = \{\frac{\sum_i A(i, j)y_l^k(i) - z_l^k(j)}{c(j)}\}_{k, l}$, and $\{a_r\}_r = \{\frac{\eta_{k, l}}{c(j)/A(p, j)}\}_{k, l}$, we have that $\frac{\sum_i A(i, j)y_l^k(i) - z_l^k(j)}{c(j)}$ is bounded by $\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ for all k , which implies the lemma. ■

We are now ready to prove Theorem 2.1.

Proof of Theorem 2.1: We first show that $D(y^t/\log_{1+\epsilon} \frac{1+\epsilon}{\delta}, z^t/\log_{1+\epsilon} \frac{1+\epsilon}{\delta}) \geq c^\top x^*/\alpha^*(1+\epsilon)$. By feasibility of x^* (Lemma 2.2 and the inner while loop in FractionalCoverUB) and $(y^t, z^t)/\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ (Lemma 2.8), and weak LP duality, this implies ϵ -optimality.

Let $D(k, l) = D(y_l^k, z_l^k)$, and let $D(k) = D(y^k, z^k)$. Let $P(k) = c^\top x^k$. The change in objective function value of (D) in one iteration is $D(k, l) - D(k, l-1) = \eta[b(p) - \sum_{j \notin Q(p)} u(j)A(p, j)]$, as determined in lines (10)-(12). We now bound the change in $c^\top x$ in terms of change in D . Recall that $Q(p) = \{1 \leq j \leq m | A(p, j) > 0 \text{ and } x(j) < u(j)\alpha\}$.

$$\begin{aligned}
c^\top x_l^k &= c^\top x_{l-1}^k + \epsilon\eta \sum_{j \in Q(p)} A(p, j)x_{l-1}^k(j) \\
&= c^\top x_{l-1}^k + \epsilon\eta \left[\sum_{j=1}^l A(p, j)x_{l-1}^k(j) - \alpha(k) \sum_{j \notin Q(p)} u(j)A(p, j) \right] \\
&\leq c^\top x_{l-1}^k + \epsilon\eta \left[b(p) - \sum_{j \notin Q(p)} u(j)A(p, j) \right] \alpha(k) \\
&= c^\top x_{l-1}^k + \epsilon(D(k, l) - D(k, l-1))\alpha(k) \\
&\leq c^\top x_0^k + \epsilon\alpha(k) \sum_{j=1}^l (D(k, j) - D(k, j-1)) \\
&= c^\top x_0^k + \epsilon\alpha(k)[D(k) - D(k-1)]
\end{aligned} \tag{5}$$

Inequality (5) follows since $\alpha(k) > A(p)^\top x_l^k / b(p)$ for all iterations l in the $\alpha(k)$ -phase. Thus, at the end of the k^{th} phase, we have that

$$\begin{aligned}
P(k) &\leq P(k-1) + \epsilon\alpha(k)(D(k) - D(k-1)) \\
&\leq P(0) + \epsilon \sum_{i=1}^k (D(i) - D(i-1))\alpha(i).
\end{aligned}$$

By definition of α , $\alpha(i-1) = \alpha(i)/(1+\epsilon)$. Also, at the end of phase k , solution x^k satisfies the property that if we scale by $\alpha(k)$, then $x^k/\alpha(k)$ is feasible for (\mathbf{P}) : it satisfies both the upper bound constraints (by Lemma 2.2), and the covering constraints. Thus the optimal solution value, OPT , satisfies $\text{OPT} \leq P(x^*/\alpha^*) \leq P(k)/\alpha(k)$ for all $1 \leq k \leq t-1$. We can now rewrite an upper bound for $P(t)$, using $\frac{\alpha(i)}{1+\epsilon} \leq \frac{P(i-1)}{P(x^*/\alpha^*)}$.

$$P(t) \leq P(0) + \frac{\epsilon(1+\epsilon)}{P(x^*/\alpha^*)} \sum_{i=1}^t (D(i) - D(i-1))P(i-1).$$

The remaining analysis follows [13, 18]. Note that, for any k , $P(t)$ is maximum when for all $i < t$, $P(i)$ takes on the maximum value. Hence, assuming that $P(i)$ represents this maximum value, and thus substituting it for the maximum value, we have that (using $1+x \leq e^x$)

$$\begin{aligned}
P(t) &\leq P(t-1)(1 + \epsilon(1+\epsilon)[D(t) - D(t-1)]/P(x^*/\alpha^*)) \\
&\leq P(t-1)e^{\epsilon(1+\epsilon)[D(t)-D(t-1)]/P(x^*/\alpha^*)} \\
&\leq P(0)e^{\epsilon(1+\epsilon)D(t)/P(x^*/\alpha^*)}.
\end{aligned}$$

Together with Lemma 2.4 and the fact that $P(0) \leq \delta c^\top u$, this implies $1 \leq \delta c^\top u e^{\epsilon(1+\epsilon)D(t)/P(x^*/\alpha^*)}$, or equivalently, the following bound on the ratio between the best feasible solution to (\mathbf{GSN}) found and final (unscaled) dual solution.

$$\frac{P(x^*/\alpha^*)}{D(t)} \leq \frac{\epsilon(1+\epsilon)}{\ln(\delta c^\top u)^{-1}}.$$

Choosing $\delta = (1+\epsilon)((1+\epsilon)c^\top u)^{-1/\epsilon}$ implies 2ϵ -optimality.

For this choice of δ , the number of phases is $O(\epsilon^{-2} \log(c^\top u))$ by Lemma 2.4, and by Lemma 2.6 the number of oracle calls is $O(\epsilon^{-2} m \log(c^\top u))$. \blacksquare

3 Zero-One Constraint Matrices

When A is a 0-1 matrix, and we can bound the value of OPT from above and below within a polynomial factor of accuracy, then we can replace the dependence in number of phases and iterations on c with $\log(\frac{mU}{\epsilon})$, where $U := \max_j u(j)$. In particular, if $\max_i b(i)$ or $\max_j u(j)$ is bounded by a polynomial in m , then this implies that the number of phases and iterations are $O(\log m\epsilon)$ and $O(m \log m\epsilon)$, respectively.

This is done via two observations. Let x^* be a solution of value OPT .

- 1) If $c(j) > \text{OPT}$, then $x^*(j) = 0$ (see Lemma 3.2 below);
- 2) if $c(j)u(j) < \epsilon \text{OPT}/m$, then setting $x(j) = u(j)$ for all such variables j increases the value of the solution by at most ϵOPT , and hence choosing ϵ appropriately maintains our approximation guarantee.

Of course, we can only use these observations if we can bound OPT .

Theorem 3.1 *If we have an estimate EST of OPT such that*

$$\text{OPT} \leq \text{EST} \leq \gamma \text{OPT} \tag{6}$$

for γ bounded by a polynomial in m , and all entries of A are 0 or 1, then the number of phases and iterations required by FractionalCoverUB are $O(\epsilon^{-2} \log \frac{mU}{\epsilon})$ and $O(\epsilon^{-2} m \log \frac{mU}{\epsilon})$, respectively.

Proof: Given EST , remove all variables j with $c(j) > \gamma \text{EST}$ using Lemma 3.2. For all j with $c(j)u(j) < \epsilon \text{EST}/m$, set $x(j) = u(j)$. Then, after scaling the costs of the remaining variables so that $\min_j c(j)u(j) = 1$, we have that $c^\top u \leq \gamma mU/\epsilon$. Thus, $\log(c^\top u) = \log \frac{mU}{\epsilon}$. ■

Lemma 3.2 *If $c(j) > \text{OPT}$, then $x^*(j) = 0$.*

Proof: Suppose variable j_0 has $c(j_0) > \text{OPT}$ and $x^*(j_0) > 0$. Then $\mu := x^*(j_0) < 1$. Let F be the set of variables with strictly fractional $x^*(j)/u(j)$ value, i.e. $F := \{j \mid 0 < x^*(j) < u(j)\}$. We modify x^* to obtain x' as follows:

$$x'(j) = \begin{cases} 0 & \text{if } j = j_0 \\ u(j) & \text{if } x^*(j)/u(j) > 1 - \mu \\ \frac{x^*(j)}{1 - \mu} & \text{if } x^*(j)/u(j) \leq 1 - \mu. \end{cases}$$

The cost of x' is less than OPT :

$$cx' \leq \frac{cx^* - c(j_0)\mu}{1 - \mu} < \frac{\text{OPT} - \text{OPT}\mu}{1 - \mu} \leq \text{OPT}.$$

We now show that x' is a feasible solution to (\mathbf{P}) , contradicting the optimality of x^* . Consider constraint p with $A(p, j_0) = 1$ and $\mu_p := b(p) - [A(p)^\top x^* - \mu] > 0$. Define $F_{>} := \{j \in F - \{j_0\} : x^*(j)/u(j) > 1 - \mu\}$ and $F_{\leq} := \{j \in F - \{j_0\} : x^*(j)/u(j) \leq 1 - \mu\}$. We show that the increase of values of variables in either $F_{>}$ or F_{\leq} compensates for the loss of $x^*(j)$. If $\sum_{j \in F_{>}} A(p, j)[u(j) - x^*(j)] \geq \mu_p$, then $\sum_j A(p, j)x'(j) \geq b(p)$. Otherwise, $\sum_{j \in F_{>}} A(p, j)[u(j) - x^*(j)] = \mu' < \mu_p \leq \mu$. Since $b(p)$ is integer and x^* is feasible, $\sum_{j \in F_{\leq}} A(p, j)x^*(j) = N - \mu_p + \mu' \geq 1 - \mu$ for some positive

integer N . This implies that the variables in F_{\leq} are increased sufficiently to compensate for the loss of value $x^*(j)$:

$$\begin{aligned} \sum_{j \in F_{\leq}} A(p, j) [x'(j) - x^*(j)] &= \\ \sum_{j \in F_{\leq}} A(p, j) x^*(j) \left[\frac{1}{1 - \mu} - 1 \right] &\geq \mu. \end{aligned}$$

■

Garg and Khandekar [17] give a version of Lemma 3.2 when there are no upper bounds on the value of $x(e)$.

4 Fractional Network Design

The generalized Steiner network (GSN) problem is defined by a graph $G = (V, E)$ with $m = |E|$ and $n = |V|$, an edge-cost vector c and a connectivity oracle f . Given a subset of vertices $S \subseteq V$, $f(S)$ specifies the minimum number of edges with exactly one end point in S that must be included in any feasible solution. The objective is to find a minimum cost set of edges satisfying f on all subsets $S \subseteq V$.

A natural special case of this problem is the undirected edge connectivity problem. We will refer to this case as *survivable network design (SND)*. In this problem, each pair (i, j) of vertices has a connectivity requirement $r(i, j)$ that is the number of (i, j) edge disjoint paths required in any feasible solution. Then $f_{\text{SND}}(S) = \max_{i \in S, j \notin S} r(i, j)$. The integer version of SND is NP-hard. Jain [26] describes an approximation algorithm that finds an integer solution with value at most twice the optimal value. His algorithm iteratively rounds the solution to the linear program formulation, and thus its efficiency depends on the efficiency of the LP solver.

The linear program for GSN is given below. For each edge e there is a variable $x(e)$. In an integer solution x , $x(e) = 1$ means that e is in the final solution and $x(e) = 0$ means that e is not in the solution. Define $\Delta(S)$ for $S \subset V$ to be the set of edges with exactly one end point in S . Note that for the problem to be feasible $f(\emptyset) = f(V) = 0$, and $f(S) \leq |\Delta(S)|$.

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c(e)x(e) \\ \forall S \subset V : \quad & \sum_{e \in \Delta(S)} x(e) \geq f(S) \\ \forall e : \quad & x(e) \leq 1 \\ \forall e : \quad & x(e) \geq 0 \end{aligned} \tag{GSN}$$

We denote the optimal solution value to (GSN) by OPT. To implement FractionalCoverUB to solve (GSN), we require an oracle that finds a constraint $\sum_{e \in \Delta(S)} x(e) < \alpha f(S)$ if one exists.

Williamson [35] describes efficient oracles for general f . For SND, there is a well-known oracle for $f_{\text{SND}}(S)$ that uses a Gomory-Hu tree. We introduce a new, efficient procedure for finding inequalities of the form $\sum_{e \in \Delta(S)} x(e) < \alpha f_{\text{SND}}(S)$ if they exist. Define k as the cardinality of the set $\{(i, j) \in V \times V \mid r_{ij} > 0\}$, so that $k = O(n^2)$. Naively using a tree that represents connectivity requirements, each search requires $\min\{k, n\}$ maximum flow computations in a network with capacities determined

by x . A simplification based on an idea introduced in [13] for multicommodity flows, cycles through each edge of the tree, and reduces the computation per search to one maximum flow. We show how to perform all searches in an α -phase in the same asymptotic time as $\min\{n, k\}$ maximum flow computations. Since the average number of iterations per phase is $O(m)$, this implies a further reduction in complexity.

4.1 Connectivity Trees and an oracle for SND.

A *connectivity tree* of an undirected, weighted graph G is an edge-weighted tree T_G that concisely represents connectivity information about G : it has the property that for all $i, j \in V$, the weight of the least weight edge on the path from i to j in T_G is at least the weight of (i, j) in G . This concept is a variant of the idea of the Gomory-Hu tree S_G , where the weight of the least weight edge from i to j in S_G equals the value of the minimum cut separating i and j in G . However, T_G is easier to compute: T_G is a maximum weight spanning tree in G , and thus can be computed in $O(m \log n)$ time.

Let $H = (V, E_H)$ be a graph on V that contains an edge for every pair (i, j) with $r_{ij} > 0$. Note that $k = |E_H|$. Interpret r_{ij} as the weight of edge (i, j) . In the connectivity tree T_H of H , the (implied) connectivity requirement for (i, j) is the weight of the minimum weight edge in the path from i to j in T_H . Let $F_e \subset E$ be the cut set of G defined by the removal of edge e in T_H : The removal of e creates subtrees T_H^1 and T_H^2 . Define $F_e \subset E$ to be the set of edges in G with one end point in T_H^1 and one end point in T_H^2 . Let x be a vector of edges capacities for G . Let $\nu(i, j)$ be the value of the maximum flow from i to j (= value of minimum cut separating i and j) in G with edge capacities x .

Lemma 4.1 *x satisfies all connectivity requirements r if and only if $\nu(i, j) \geq r_{ij}$ for all $(i, j) \in T_H$.*

Proof: The “only if” direction is clear. Let u and v be nonadjacent vertices in T_H with $r_{u,v} > 0$. Let $(u = w_0, w_1, \dots, w_q = v)$ be the adjacent vertices on the unique path P from u to v in T_H . Each cut in G separating u from v must separate at least one pair of consecutive vertices in P . Since $\nu(i, j) \geq r_{ij}$ for each $(i, j) \in T_H$, each cut in G separating u from v has value at least the minimum weight edge in P , which is at least r_{uv} . Hence, x satisfies the connectivity requirement for (u, v) . ■

Lemma 4.1 is useful both to obtain a good estimate of OPT for SND, and also for finding violated inequalities of SND. We first explain its applicability to obtaining good estimates of OPT.

Estimating OPT.

Lemma 4.2 *An estimate EST of OPT for SND that satisfies (6) with $\gamma = n$ can be found with $\min\{n, k\}$ minimum cost flow computations in a unit capacity graph.*

Proof: $k \geq n$: For edge $(i, j) \in T_H$, the minimum cost flow of value r_{ij} from i to j in the graph G with unit capacities and costs c may be assumed to be integral. The set of edges that carry flow thus define a minimum cost network that allows for connectivity r_{ij} between i and j . Let l_{ij} be the cost of this flow. Then clearly $l_{ij} \leq \text{OPT}$. Repeating this for each edge in T_H yields a

collection of edges. This collection of edges is sufficient to meet all connectivity requirements by Lemma 4.1. Thus we have that $\max_{e \in T_H} l_e \leq \text{OPT} \leq \sum_{e \in T_H} l_e \leq (n-1) \max_{e \in T_H} l_e$. Hence, the estimate $\max_{e \in T_H} l_e$ satisfies (6) for $\gamma = n$.

If $k < n$, then we don't need to use T_H and can simply compute l_e for each $e \in E_H$ to obtain $\max_{e \in E_H} l_e \leq \text{OPT} \leq k \max_{e \in E_H} l_e$. ■

Corollary 4.3 *An ϵ -optimal solution to SND can be found using FractionalCoverUB with at most $O(\epsilon^{-2} \log \frac{n}{\epsilon})$ phases and at most $O(\epsilon^{-2} m \log \frac{n}{\epsilon})$ iterations.*

Proof: This follows from Theorem 3.1 and Lemma 4.2 ■

A good estimate of OPT may also be obtained by computing a sequence of at most $m+n$ minimum Steiner trees in subgraphs of G . Since a minimum spanning tree is a good approximation to a minimum Steiner tree, a good estimate of OPT may be obtained with at most $m+n$ minimum spanning tree computations.

SND Oracle Implementation. We now explain the applicability of Lemma 4.1 to finding violated inequalities of SND.

Lemma 4.4 *Given T_H , a set S that satisfies $x(\Delta_S) < \alpha f_{\text{SND}}(S)$, if one exists, can be found with at most $\min\{k, n\}$ s - t minimum cut computations.*

Proof: If $k \geq n$, then for each edge $(s, t) \in T_H$, compute a minimum s - t x -capacity cut S_{st} in G . If $x(S_{st}) \geq \alpha f_{\text{SND}}(S)$ for all $(s, t) \in T_H$, then by Lemma 4.1, $x(\Delta_S) \geq \alpha f_{\text{SND}}(S)$ for all $S \subset V$.

If $k < n$, then compute a minimum s - t x -capacity cut for each $(s, t) \in E_H$. ■

Lemma 4.4 implies that $\min\{k, n\}$ flow computations per iteration are sufficient to find a most violated inequality for SND. Using a technique introduced in [13], this can be reduced to a single flow computation per iteration, plus an additional $\min\{k, n\}$ flow computations per phase: The results of Section 2 still hold if we replace a most violated inequality with any inequality that satisfies $A(p)^\top x < ab(p)$. Finding *some* violated inequality instead of the most violated one may be done more efficiently. To do this, we fix an ordering of the edges in T_H . In the α -phase, we go through this ordering exactly once. For edge $(i, j) \in T_H$, we compute the value of an i - j maximum flow. If $\nu(i, j) \geq \alpha r_{ij}$, then we are done with the edge for the phase. Otherwise, we find an inequality $\nu(i, j) < \alpha r_{ij}$ that we can use in this iteration. We continue with the iteration in FractionalCoverUB, increasing x . In the next iteration of FractionalCoverUB, we start the search for a violated inequality with edge (i, j) .

We have established the following lemma:

Lemma 4.5 *The number of maximum flow computations required to obtain an ϵ -optimal solution to SND via algorithm FractionalCoverUB is $O(\epsilon^{-2} m \log(n/\epsilon))$.*

4.2 Faster implementation of FractionalCoverUB for SND.

In this section, we prove Theorem 4.7, which implies the second main result of this paper:

Theorem 4.6 *An ϵ -optimal solution to SND can be found with $\min\{n, k\}$ calls to a minimum cost flow algorithm for uniform capacity graphs, plus at most $O(\epsilon^{-2}m(m + n \min\{n, k\} \log n) \log n)$ additional work.*

Theorem 4.6 represents a run time improvement over existing algorithms by a factor of at least $O(\min\{m, nk\})$, and by a factor of $O(m \log n)$ if $k = \Omega(n)$.

Theorem 4.7 *The total work done by the oracle for the SND problem in order to find an ϵ -approximate solution via FractionalCoverUB is at most $\min\{n, k\}O(mn \log(n^2/m))$ per α -phase.*

To prove Theorem 4.7, we modify the variant of push-relabel algorithm of Goldberg and Tarjan [21] that uses the gap heuristic [10], and the dynamic tree implementation.

Push-Relabel algorithm. The push relabel algorithm for maximum s - t flow starts with a *pre-flow*: a function g that satisfies capacity constraints ($g(v, w) \leq u(v, w)$ for given capacity vector u) and nonnegativity constraints ($g(v, w) \geq 0$), but relaxes flow conservation so that vertices other than the source s may have *excess*: more flow may enter the vertex than leave it. It also uses vertex labels d that satisfy $d(s) = n$, and $d(v) \leq d(w) + 1$ if $g(v, w) < u(v, w)$. Initially, $d(v) = 0$ for all $v \neq s$. The push-relabel algorithm iteratively selects a vertex v with $\text{excess}(v) > 0$ and performs either a *push* operation or a *relabel* operation. **Push**(v, w) applies if $\text{excess}(v) > 0$, $g(v, w) < u(v, w)$, and $d(v) = d(w) + 1$. It increases $g(v, w)$ and decreases $\text{excess}(v)$ by $\min\{\text{excess}(v), u(v, w) - g(v, w)\}$. **Relabel**(v) applies if $\text{excess}(v) > 0$ and $g(v, w) = u(v, w)$ for all w with $d(v) = d(w) + 1$. It increases $d(v)$ by 1.

The run time of this algorithm depends on the number of relabels and push operations. Goldberg and Tarjan show that 1) the number of relabel operations is $O(n^2)$; 2) with appropriate choice of v and w in each iteration, for instance, using a FIFO selection rule for a vertex with excess, the number of saturating push operations is $O(mn)$ and the number of relabel operations is $O(n^3)$. (A *saturating push* is one that increases $g(v, w)$ by $u(v, w) - g(v, w)$. A push that is not saturating is *nonsaturating*.) Using dynamic tree implementation, the total work over all push operations is $O(nm \log(n^2/m))$.

Gap heuristic. The gap heuristic is due independently to Cherkassky [8, 9]; and Derigs and Meier [11]. Its practical use is demonstrated in Cherkassky and Goldberg [10]. It is based on the observation that if a vertex v with label $d(v)$ is eligible for a relabel operation, and there are no other vertices with this label, then the set of vertices with label $d(v)$ or higher will be on the source side of any minimum s - t cut. The heuristic says that all vertices with label at least $d(v)$ can be relabeled to n without affecting the correctness or asymptotic complexity of the algorithm. We call this set of vertices the *cut implied by the gap*.

Modified push-relabel with gap heuristic for finding violated cuts. At the start of the α -phase, the minimum x -capacity i - j cut in G has value at least $\alpha r_{ij}/(1 + \epsilon)$. We use the push-relabel algorithm to increase this to αr_{ij} .

Proof of Theorem 4.7: In one phase, we fix an ordering of edges in T_H and cycle through these edges once. For $(i, j) \in T_H$, we start using the push-relabel algorithm to compute a minimum i - j cut in G with the current iterate x as the capacity vector.

If the set of arcs leaving i has capacity less than αr_{ij} we use $\{i\}$ as the first cut considered by algorithm `FractionalCoverUB` in line (2) or line (13). Otherwise, we run the push-relabel algorithm until the first gap is encountered. If the gap implies a cut S that has capacity at least αr_{ij} , we simply contract all $v \in S$ into the source, and continue with the modified push-relabel. If the cut S implied by the gap has $x(\Delta(S)) < \alpha r_{ij}$, we use S as the determining inequality in line (13), freeze the current state of the modified push-relabel algorithm, and continue with the steps in `FractionalCoverUB` until it returns to line (13). Inside this iteration, `FractionalCoverUB` increases the x -capacity of edges only in $\Delta(S)$ (lines (6),(8),(9)). On return to line (13), `FractionalCoverUB` checks if S still has $x(\Delta(S)) < \alpha r_{ij}$, and if so, continues on with it. Otherwise, the modified push-relabel is unfrozen and resumed until a new gap (and thus a new cut) is found. This repeats until this modified push-relabel algorithm finds a flow of value at least αr_{ij} , indicating that the connectivity between i and j is at least αr_{ij} .

Note that when the approximation scheme returns to the push-relabel algorithm, the current set of labels is valid for the new capacities and old flow. Thus, the number of relabels performed by this modified algorithm over the course of the approximation scheme is no more than the number performed by the unmodified push-relabel algorithm. It remains to bound the number of pushes.

Each time the approximation scheme returns to the push-relabel algorithm, it finds a *new* cut. Since the source-side of these cuts is increasing in size, there are at most n of these for any (i, j) pair. Each time the approximation scheme returns to push-relabel, it increases the capacities of edges across the old cut. This can affect the number of saturating pushes by the number of edges that cross the cut. Since this is at most m , we have that the total number of additional saturating pushes throughout the (i, j) invocation is at most $O(mn)$. Using the dynamic tree data structure, the total additional work to update capacities is $O(nm \log(n^2/m))$.

The proof that the number of saturating pushes in the FIFO implementation of push-relabel is $O(n^3)$ uses a potential ϕ defined as sum the labels of all vertices with excess. ϕ decreases in any pass through the FIFO queue in which no vertex is relabeled, and if ϕ increases, then the total increase of labels increases by at least as much. Thus the total number of passes through the FIFO queue is at most $O(n^2)$. In each pass of the queue, each node can have at most one nonsaturating push.

In modified push relabel, the algorithm is frozen part way through a pass. When it is resumed, some capacities have increased, but this does not affect the value of ϕ . Hence the total number of nonsaturating pushes remains unchanged.

Thus the total work devoted to finding cuts for one edge in T_H is $O(nm \log(n^2/m))$.

A phase ends after this modified push relabel is performed once for each edge in T_H . Hence, the work devoted to finding cuts in one phase is asymptotically equivalent to the work to perform $\min\{n, k\}$ push-relabel maximum flow computations. ■

Without using dynamic trees, it is possible to obtain a run time of $O(n^2\sqrt{m})$ per edge of T_H by replacing the FIFO vertex selection rule with the max-label vertex selection [6].

Proof of Theorem 4.6: The algorithmic steps can be partitioned into three groups: time spent in maximum flow computations, time spent per violated cut, and time spent to bound OPT. By Theorem 4.7, the number of push-relabel maximum flow computations over the course of the algorithm is $\min\{n, k\}$ times the number of phases. Thus, by Corollary 4.3 and the run time complexity of the push-relabel algorithm, the total work to execute the maximum flow computations

is $O(\epsilon^{-2}mn \min\{n, k\} \log^2 n)$. There is at most linear work per cut, and the number of cuts is bounded by the number of iterations, so the number of steps is bounded by $O(\epsilon^{-2}m^2 \log n)$. Finally, to bound OPT we use $\min\{n, k\}$ minimum cost flow computations in uniform capacity graphs (Lemma 4.2). ■

4.3 Efficient oracles for other network design problems.

The modified push-relabel algorithm above is also useful in designing fast oracles for other network design problems such as element connectivity, vertex connectivity, directed connectivity, capacitated network design, and group Steiner forest. We briefly discuss the use for vertex connectivity and directed connectivity below. The modifications necessary to adopt these ideas to the remaining problems are similarly straightforward, and hence omitted.

Vertex connectivity and directed connectivity. The *vertex connectivity* problem is a generalization of (GSN) in that given, for all $\{i, j\} \in V \times V$, the connectivity requirement r_{ij} the goal is to select a minimum weight subset of edges $F \subseteq E$ in a weighted, undirected graph $G = (V, E)$ such that there are at least r_{ij} *vertex disjoint* paths between i and j .

Vertex connectivity problems can be modeled as edge connectivity problems in a slightly larger directed graph [1]. Thus it suffices to describe an oracle for the directed connectivity problem. We extend the connectivity tree concept used for the undirected case to a *strong connectivity graph*. This graph has a directed path from i to j of capacity at least r_{ij} . In the worst case, this contains all edges with $r_{i,j} > 0$, although it may be possible to reduce this by excluding (i, j) if there already exists a path of edges all with value r_{ij} or higher.

Once this graph is built, the required oracle need only check that for each edge (i, j) in H , the maximum flow value is at least αr_{ij} in the graph G with capacities determined by current variable vector x . On an iteration basis, this is precisely the modified push-relabel oracle described for (GSN).

Theorem 4.8 *An ϵ -optimal solution to directed SND can be found with k calls to a minimum cost flow algorithm for uniform capacity graphs, plus at most $O(\epsilon^{-2}m(m + nk \log n) \log n)$ additional work.*

Acknowledgment

Thanks to Neal Young, Cliff Stein, Dan Bienstock, Garud Iyengar, and the anonymous referees for their helpful comments.

References

- [1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [2] D. Bienstock. Experiments with a network design algorithm using ϵ -approximate linear programs. Submitted for publication, 1996.

- [3] D. Bienstock. Approximately solving large-scale linear programs. i. strengthening lower bounds and accelerating convergence. Technical report, CORC Report 1999-1, Columbia University, 1999. Extended abstract: Proc. 11th Ann. ACM-SIAM Symposium on Discrete Algorithms, January 2000.
- [4] Daniel Bienstock. *Potential function algorithms for approximately solving linear programs: theory and practice*. International Series in Operations Research and Management Science. Kluwer Academic Publisher, 2002.
- [5] R. D. Carr, L. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 106–115, 2000.
- [6] J. Cheriyan and K. Mehlhorn. An analysis of the highest-level selection rule in the preflow-push max-flow algorithm. *Information Processing Letters*, 69:239–242, 1999.
- [7] J. Cheriyan, S. Venpala, and A. Vetta. Approximation algorithms for minimum-cost k-vertex connected subgraphs. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, 2002.
- [8] B. V. Cherkassky. A fast algorithm for computing maximum flow in a network. In A. V. Karzanov, editor, *Collected Papers: Combinatorial Methods for Flow Problems*, volume 3, pages 90–96. The Institute for Systems Studies, Moscow, 1979. In Russian.
- [9] B. V. Cherkassky. A fast algorithm for constructing a maximum flow through a network. In *Selected Topics in Discrete Mathematics (Moscow, 1972-1990)*, number 158 in Amer. Math. Soc. Transl. Ser. 2, pages 23–30. Amer. Math. Soc., Providence, RI, 1994.
- [10] B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.
- [11] U. Derigs and W. Meier. Implementing Goldberg’s max-flow algorithm — a computational investigation. *Z. Oper. Res.*, 33(6):383–403, 1989.
- [12] L. Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004. To appear.
- [13] L. K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.
- [14] L. K. Fleischer. A 2-approximation for minimum cost $\{0, 1, 2\}$ vertex connectivity. In *8th International Integer Programming and Combinatorial Optimization Conference*, pages 115–129, 2001.
- [15] L. K. Fleischer, K. Jain, and D. P. Williamson. An iterative rounding 2-approximation algorithm for the element connectivity problem. In IEEE [25], pages 339–347.
- [16] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Res. Logistics Quarterly*, 3:149–154, 1956.
- [17] N. Garg and R. Khandekar. Fast approximation algorithms for fractional Steiner forest and related problems. In *43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 500–, 2002.
- [18] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *39th Annual IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.
- [19] N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000.
- [20] A. V. Goldberg, J. D. Oldham, S. Plotkin, and C. Stein. An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 338–352, Berlin, 1998. Springer.
- [21] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.

- [22] M. D. Grigoriadis and L. G. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4:86–107, 1994.
- [23] M. D. Grigoriadis and L. G. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Mathematics of Operations Research*, 21:321–340, 1996.
- [24] M. D. Grigoriadis, L. G. Khachiyan, L. Porkolab, and J. Villavicencio. Approximate max-min resource sharing for structured concave optimization. *SJO*, 11(4):1081–1091, 2001.
- [25] *42nd Annual Symposium on Foundations of Computer Science*, 2001.
- [26] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.
- [27] G. Karakostas. Faster approximation schemes for fractional multicommodity flow. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.
- [28] D. Karger and S. Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 18–25, 1995.
- [29] P. Klein, S. 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*, 23:466–487, 1994.
- [30] 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.
- [31] V. Melkonian and E. Tardos. Approximation algorithms for a directed network design problem. In *7th International Integer Programming and Combinatorial Optimization Conference*, pages 345–360, 1999.
- [32] S. A. Plotkin, D. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.
- [33] T. Radzik. Fast deterministic approximation for the multicommodity flow problem. *Mathematical Programming*, 78:43–58, 1997.
- [34] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37:318–334, 1990.
- [35] D. P. Williamson. *On the Design of Approximation Algorithms for a Class of Graph Problems*. PhD thesis, MIT Dept. of Computer Science, September 1993.
- [36] N. Young. Randomized rounding without solving the linear program. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, 1995.
- [37] N. Young. Sequential and parallel algorithms for mixed packing and covering. In IEEE [25]. 538-546.