**COMBINATORICA**
Akadémiai Kiadó – Springer-Verlag

COMBINATORICA **15** (2) (1995) 187–202

# AN APPROXIMATE MAX-FLOW MIN-CUT RELATION FOR UNDIRECTED MULTICOMMODITY FLOW, WITH APPLICATIONS*

PHILIP KLEIN‡, SATISH RAO§, AJIT AGRAWAL† and R. RAVI†

*Received July 27, 1992*
*Revised July 27 1993*

In this paper, we prove the first approximate max-flow min-cut theorem for undirected multicommodity flow. We show that for a feasible flow to exist in a multicommodity problem, it is sufficient that every cut's capacity exceeds its demand by a factor of $O(\log C \log D)$, where $C$ is the sum of all finite capacities and $D$ is the sum of demands. Moreover, our theorem yields an algorithm for finding a cut that is approximately minimum *relative* to the flow that must cross it. We use this result to obtain an approximation algorithm for T. C. Hu's generalization of the multiway-cut problem. This algorithm can in turn be applied to obtain approximation algorithms for minimum deletion of clauses of a 2-CNF≡ formula, via minimization, and other problems. We also generalize the theorem to hypergraph networks; using this generalization, we can handle CNF≡ clauses with an arbitrary number of literals per clause.

## 1. Introduction

The amount of flow one can push through a network from a source to a sink clearly cannot exceed the capacity of a cut separating them, and the celebrated max-flow min-cut theorem of Ford and Fulkerson [8] and of Elias, Feinstein and Shannon [7] showed that the capacity upper bound is always tight. The value of the maximum flow is equal to the capacity of the minimum cut. This result yielded as a byproduct an algorithm for finding a minimum capacity cut.

Ever since, researchers have sought to generalize the max-flow min-cut theorem to apply to cases of *multicommodity* flow, in which each of several commodities has its own source and sink. In 1963, Hu showed [12] that such a theorem held for 2-commodity flow. In subsequent work, various special cases have been identified for which the max-flow min-cut theorem holds (see, e.g., [9,29]). As was demonstrated

Mathematics Subject Classification (1991): 68 Q 25, 68 R 10, 90 C 08, 90 C 27

* Most of the results in this paper were presented in preliminary form in "Approximation through multicommodity flow", *Proceedings, 31th Annual Symposium on Foundations of Computer Science* (1990), pp. 726–737.

† Brown University, Providence, RI. Research supported by the National Science Foundation under NSF grant CDA 8722809, by the Office of Naval and the Defense Advanced Research Projects Agency under contract N00014–83–K–0146, and ARPA Order No. 6320, Amendament 1.

‡ Research supported by NSF grant CCR–9012357 and by an NSF Presidential Young Investigator Award.

§ NEC Research Institute, Princeton, NJ. Research supported by ONR Grant N0014–88–K–0243.

fairly early, however, no such theorem can hold in general for all multicommodity flow instances.

Leighton and Rao [19] recently defined the notion of an *approximate* max-flow min-cut theorem, by showing that the capacity upper bound is within a logarithmic factor of being tight for a special class of multicommodity flow instances, called *uniform* multicommodity flow. In a uniform flow problem for a graph $G$, a flow of value 1 is required between every two nodes of $G$. Their approximate max-flow min-cut theorem also yields an algorithm for finding an approximately *sparsest cut*, which in turn is the basis for a variety of approximation algorithms for NP-complete graph problems.

In this paper, we prove an approximate max-flow min-cut theorem that holds for all undirected multicommodity flow problems. We allow both nodes and edges to have capacities.

**Theorem 1.1.** *Consider an undirected multicommodity flow instance where the sum of the demands is $D$ and the sum of the finite capacities is $C$ (and demands and capacities are integral). In order for there to exist a feasible flow, it is sufficient that every cut's capacity exceeds the demand across the cut by a factor of $O(\log C \log D)$.*

Moreover, our theorem yields an algorithm for finding a cut that is (approximately) minimum *relative* to the flow that must cross it. That is, a minimum cut is one minimizing the ratio

```
capacity of the cut/demand crossing the cut
```

**Theorem 1.2.** *There is a polynomial-time algorithm to find an approximately minimum multicommodity cut. The approximation factor is $O(\log C \log D)$, where the sum of the demands is $D$ and the sum of the finite capacities is $C$, and the demands and capacities are integral.*

By appropriate choices of source-sink pairs, one can use this algorithm to find cuts that are (approximately) minimum subject to certain criteria.

The following observation is useful in applications where either the capacities or demands (but not both) are superpolynomial. It can be proved using a rounding technique; we omit the proof.

**Observation 1.3.** *If either $C$ or $D$ is polynomial in $n$, the number of nodes of $G$, then the approximation factor in Theorem 1.2 can be improved to be $O(\log^2 n)$.*

Our first application concerns the following generalization of the multiway cut problem [6, 12]. Given a graph $G$ with edge-weights and a set of node-pairs $(u_i, v_i)$, $1 \leq i \leq k$, find a minimum-weight collection of edges of $G$ whose removal disconnects $u_i$ from $v_i$ for all $1 \leq i \leq k$. A special case of this problem was shown to be NP-hard by Dahlhous et al. We give the first approximation algorithm. Our techniques extend to handle the analogous problem in which nodes are to be removed instead of edges.

**Theorem 1.4.** *There are polynomial-time approximation algorithms for the generalized multiway edge-cut and node-cut problems. The approximation factor is $O(\log^3 n)$, where $n$ is the number of nodes in the input graph.*

The algorithm consists of a series of greedy applications of our min-cut algorithm.

Using the node-cut algorithm, we can handle another problem, deletion of clauses of a CNF $\equiv$ formula to achieve satisfiability. Such a formula has weighted clauses of the form $(p_1 \equiv p_2 \equiv \ldots \equiv p_k)$, where each $p_i$ is either a Boolean variable or the negation of a variable. The goal is to find a minimum-weight collection of clauses whose deletion results in a satisfiable formula. We give the first approximation algorithm for this problem.[1]

**Theorem 1.5.** *Minimum-weight deletion of clauses of a* CNF $\equiv$ *formula to achieve satisfiability can be approximated to within a factor of* $O(\log^3 n)$, *where* $n$ *is the number of variables appearing in the formula.*

One of our aims in this research is to develop a framework for approximate solution of graph problems of the form: delete a minimum number of nodes (or edges) in order to obtain a graph with a desired structural property. Our algorithm for CNF $\equiv$ clause deletion provides a good basis for such algorithms; one can sometimes state the structural property in the language of CNF $\equiv$ in such a way that deleting clauses corresponds to deleting edges or nodes of the graph.[2]

The *edge-deletion graph bipartization problem* — deleting a minimum number of edges to get a bipartite graph — was studied in [32]. It is easy to state using CNF $\equiv$ that a given graph is bipartite: For each node $v$, we have a Boolean variable $p_v$. For each edge $\{v, w\}$, there is a clause $(p_v \equiv \neg p_w)$ of weight 1. The resulting formula is satisfiable if and only if the graph is bipartite; any satisfying truth assignment gives a bipartition (a two-coloring). Moreover, the minimum number of clauses that must be deleted to achieve satisfiability equals the number of edges that must be deleted to achieve two-colorability. The *node-deletion graph bipartization problem* can be modeled similarly. It follows that for these problems one obtains an $O(\log^3 n)$-factor approximation algorithm.

Using similar techniques, we can solve some other problems. The *via minimization problem* (discussed in [4]) is a problem arising in the design of a printed circuit board, in which there are two layers and one wants to minimize the number of holes made in the board in order to connect wires on different layers. The geometry of the problem is fixed; the goal is to choose an assignment of pieces of wire to layers. It is straightforward to model the via minimization problem as a CNF $\equiv$ clause-deletion problem, and thereby obtain an $O(\log^3 n)$-factor approximation algorithm for it.

In the next section, we give the proof of our main results, an approximate max-flow min-cut theorem for undirected multicommodity flow and an approximation algorithm for multicommodity min-cut. In Section 3 we show how using the min-cut algorithm in a greedy fashion yields approximation algorithms for separating a constant fraction of the demands. We show how a greedy application of

[1] A complementary result was obtained by Johnson [13] in 1974; he showed that for CNF formulae the maximum size of a satisfiable subset of clauses could be approximated to within a small constant factor; indeed, for every formula, all but at most a constant fraction of the clauses can be satisfied simultaneously. In view of this latter result, it makes sense to focus on how many clauses must be deleted to achieve satisfiability.

[2] Cf. Papadimitriou and Yannakakis's idea [24] of expressing NP-complete problems with quantified sentences in order to study their approximability.

this algorithm in turn yields an approximation algorithm for separating all the demands. This algorithm approximately solves the generalized multiway-cut problem of Theorem 1.4. In Section 4, we show how to use this algorithm to approximately solve the CNF$\equiv$ clause-deletion problem. In Section 5, we briefly consider the application of the algorithm to the via minimization problem. We make some final remarks in Section 6.

## 2. An approximate max-flow min-cut theorem for undirected multicommodity flow

### 2.1. Preliminaries

In this section we prove our approximate max-flow min-cut theorem for undirected multicommodity flow. An instance of undirected multicommodity flow consists of a network $G$ with node-capacities $\mathrm{CAP}(v)$ and edge-capacities $\mathrm{CAP}(uv)$, and commodities $\{(s_i,t_i,d(i)) : i=1,\ldots,k\}$, where $s_i$ and $t_i$ are, respectively, the source and sink of commodity $i$, and $d(i)$ is the demand of commodity $i$. We assume all demands and capacities are integral. A multicommodity flow $f$ is an assignment of flows to all the commodities; a flow for the $i^{th}$ commodity is a way of routing $d(i)$ units of the commodity from $s_i$ to $t_i$ through the edges of $G$. The multicommodity flow $f$ is said to satisfy the demands. We say $f$ is feasible with respect to the capacities $\mathrm{CAP}(\cdot)$ if for every edge $uv$ (every node $v$), the total number of units of all commodities routed through edge $uv$ (through node $v$) is at most $\mathrm{CAP}(uv)$ (at most $\mathrm{CAP}(v)$). For a node $v$, flow originating at $v$ is not counted as being routed through $v$.

A *concurrent flow* [30] of *capacity utilization* $u$ is a multicommodity flow satisfying the demands and feasible with respect to the multiplied capacities $u\cdot\mathrm{CAP}(\cdot)$. The concurrent flow problem is to find a flow $f$ with minimum capacity utilization. The concurrent flow problem relates to feasibility of ordinary multicommodity flow: a multicommodity flow is feasible if and only if the minimum capacity utilization is at most 1.

An equivalent way to formulate the concurrent flow problem is as follows: a concurrent flow of throughput $z$ is a multicommodity flow satisfying the multiplied demands $z\cdot d(\cdot)$ and feasible with respect to the capacities $\mathrm{CAP}(\cdot)$. The concurrent flow problem is then to find a flow with maximum throughput.

For a subset $W$ of the nodes and edges of $G$, let $\mathrm{CAP}(W)$ denote the sum of capacities of the elements of $W$. We say $W$ *separates* a commodity if the endpoints of the commodity are in different components of $G-W$. For purposes of this definition, we treat each node in $W$ as a singleton connected component of $G-W$. Let $\mathrm{DEM}(W)$ denote the sum of demands of commodities separated by $W$.

A multicommodity minimum cut is a subset $W$ that minimizes the ratio of capacity to demand. That is, define the *minimum cut ratio* $S$ as follows:

$$S = \min_W \frac{\mathrm{CAP}(W)}{\mathrm{DEM}(W)} \tag{1}$$

where $W$ ranges over all subsets of nodes and edges.[3] It is easy to see that the maximum throughput is at most the minimum cut ratio. For suppose the minimum cut is $W$, and $f$ is a multicommodity flow of throughput $z$. Then $f$ must route at least $z \cdot \mathrm{DEM}(W)$ units of flow through $W$. Since $f$ is feasible with respect to the capacities, the capacity of $W$ must be at least $z \cdot \mathrm{DEM}(W)$. This proves that $z$ is at most $\mathrm{CAP}(W)/\mathrm{DEM}(W)$.

An equivalent statement is that the value of the capacity utilization is at least $1/S$. Consequently, for a multicommodity flow problem to be feasible, the minimum cut ratio must be at least one. Theorem 1.1 states an approximate converse, namely that if the minimum cut ratio is at least $O(\log C \log D)$ then the multicommodity flow problem is feasible. We proceed with the proof of Theorem 1.1.

We can assume without loss of generality that only nodes are assigned capacities, i.e. that the capacity of every edge is unbounded. Simply replace an edge $vw$ that has capacity $c$ with a pair of edges $vx$ and $xw$, where $x$ is a new node that has capacity $c$. As we will see, infinite capacities are not counted in $C$, the sum of capacities. We can model the case in which only (original) edges have capacities by assigning finite capacities to the new nodes $x$ and unbounded capacities to the original nodes.

Next, we formulate the linear program dual to the problem of minimizing the capacity utilization $u$. Let $\ell$ be any nonnegative length function assigning lengths to the finite-capacity nodes of the graph $G$; the length of a node with unbounded capacity is considered to be zero. Let $dist_\ell(v,w)$ be the resulting shortest path distance between $v$ and $w$ (including the lengths of $v$ and $w$ ). The dual linear program is maximizing

$$\sum_{\text{commodity } i} dist_\ell(s_i, t_i) d(i) \tag{2}$$

subject to the constraint

$$\sum_{\text{nodes } v} \mathrm{CAP}(v)\ell(v) = 1, \tag{3}$$

where the sum is over the finite-capacity nodes. In particular, the value of (2) is always at most the value of the capacity utilization $u$ by weak linear programming

[3] In the case where only edges have finite capacity, we can restrict our attention to edge-cuts $W$ of the form $\Gamma(X)$, where $X$ is a node-subset and $\Gamma(X)$ denotes the set of edges having exactly one endpoint in $X$. From any edge-cut $W$ one can easily extract an equally good cut of this form that separates the graph into exactly two components. This observation permits a particularly nice formulation of the min-cut problem, in which commodities are represented as edges in the graph $G$. Each pair $s_i, t_i$ of commodity endpoints is considered a *demand* edge, and is assigned a weight $\mathrm{DEM}(s_i t_i)$ equal to the demand value $d(i)$. The original edges of $G$ are called *capacity edges.* For a node-subset $X$, the capacity-to-demand ratio of the cut $\Gamma(X)$ is the ratio $\mathrm{CAP}(\Gamma(X))/\mathrm{DEM}(\Gamma(X))$ of the weight of its capacity edges to the weight of its demand edges. We can think of the weight of demand edges as signifying benefit and the weight of capacity edges as signifying cost; a minimum cut then becomes a cut whose cost-to-benefit ratio is minimized.

duality. Moreover, when $\ell$ maximizes (2) and $f$ minimizes $u$, the sum (2) equals $u$. That is, at optimality, we have

$$\sum_i dist_\ell(s_i, t_i)d(i) = u \geq 1/S. \tag{4}$$

Our main result in this section is to show that the leftmost expression is not much more than the rightmost expression.

**Theorem 2.1.** *There is a polynomial-time algorithm to find a particular node-separator $W'$ such that*

$$\sum_i dist_\ell(r_i, s_i)d(i) = O(\log C \log D)\frac{\mathrm{DEM}(W')}{\mathrm{CAP}(W')} \tag{5}$$

*where $C$ is the sum of all finite capacities and $D$ is the sum of all demands, and the demands and capacities are integral.*

It follows by (4) and the definition of $S$ that

$$\max_W \frac{\mathrm{DEM}(W)}{\mathrm{CAP}(W)} \leq u \leq O(\log C \log D)\frac{\mathrm{DEM}(W')}{\mathrm{CAP}(W')} \tag{6}$$

where $W'$ is the node subset found by the algorithm. This inequality shows that $W'$ defines a near-optimal cut, proving Theorem 1.2. Moreover, since a multicommodity flow instance is feasible if $u$ is no more than 1, the inequality also proves Theorem 1.1.

### 2.2. The proof of Theorem 2.1

Our proof of the theorem follows the lines of Leighton and Rao's proof for their result concerning the uniform demand case (the case where there is a demand of 1 between every pair of nodes).

First solve the dual of the concurrent flow problem, obtaining an optimal length function $\ell$ satisfying the constraint (3). Next, assume (for now) that we know the value of $S$. We now give an algorithm to assign a path $P(s_i, t_i)$ to each commodity $i$ such that

$$\sum_i \mathrm{length}(P(s_i, t_i)) \cdot d(i) = O((1/S)\log C \log D). \tag{7}$$

The algorithm works in stages, assigning paths to disjoint sets of commodities in different stages. We use $D_t$ to denote the sum of the demands of the commodities that have not yet been assigned paths by stage $t$. To initialize, let $D_0$ equal the sum $D$ of all demands, and let $t=0$.

Each stage $t$ consists of the following steps. Decompose the graph into node-disjoint trees so that each tree has height $O((1/SD_t)\log C)$. (This step is described in more detail later.) For each commodity $i$ whose endpoints lie in a common

tree, assign to the commodity the path in that tree, and discard the commodity. Then $D_{t+1}$ is the sum of the demands of remaining commodities, i.e. those whose endpoints are in different trees. Finally, increment $t$, and loop.

In each stage $t$, the length of every path assigned is $O((1/SD_t)\log C)$, and the sum of demands of commodities that were assigned paths is $D_t - D_{t+1}$, so the total contribution of stage $t$ to the left-hand side of (7) is $O((1/S)\log C))$. It remains to describe a procedure for decomposing $G$ into trees of small height such that $D_{t+1} \leq D_t/2$, for then the number of stages is at most $\log D$, and the total contribution of all stages to the left-hand side of (7) is $O((1/S)\log C \log D)$.

**Decomposition into trees.** It is convenient to discretize the distance into tokens. Each token of distance will correspond to distance $1/C$ in the original units. That is, define

$$\ell_t(v) := \lceil C\ell(v) \rceil \tag{8}$$

for each node $v$. Each token associated with $v$ is assigned a weight equal to the capacity of $v$. Notice that since $\sum \ell(v)\mathrm{CAP}(v) \leq 1$ and rounding up adds only one extra token for each node, we have that the total token weight is

$$\sum \ell_t(v)\mathrm{CAP}(v) < 2C. \tag{9}$$

Measured in tokens, we must decompose the graph into trees of token height $O((1/SD_t)C\log C)$.

Next, repeat the following step, forming a tree $T$ (as described below), and then removing the nodes of the tree from the graph, until the graph no longer contains a source or sink.

**Forming a tree $T$:** Start at any node $r$, and compute the shortest-path tree consisting of all nodes reachable from $r$. If every node in this tree has distance at most $\lceil (5C\ln 2C)/SD_t \rceil + 1$, then let $T$ be this tree. Otherwise, we must select a distance $\hat{d} \leq \lceil (5C\ln 2C)/SD_t \rceil + 1$ at which to truncate the tree.

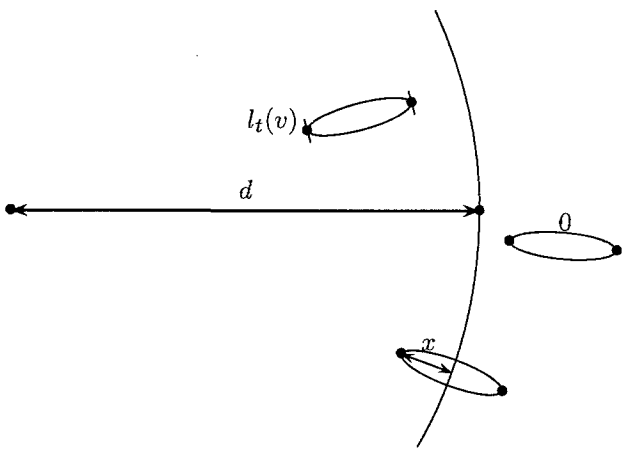


*Fig. 1.* If the distance to $v$ not including $v$'s length is $d-x$, then $\min(x, \ell_t(v))$ of $v$'s $\ell_t(v)$ tokens are within distance $d$

There is a natural notion of how many of a node's tokens lies within a given distance $d$ from the root $r$. A node $v$ has $\ell_t(v)$ tokens in total. Let $d-x$ be the distance to $v$ not including $v$'s length. The number of $v$'s tokens within distance $d$ is defined to be $\min(x, \ell_t(v))$. Denote this value by $\text{amount}(v,d)$. We determine the total *weight* of the subgraph within distance $d$ of $r$, defined as follows.

$$\text{weight}(d) = \sum_{\text{node } v} \text{amount}(v,d) \cdot \text{CAP}(v).$$

We use binary search on the range of distances $d = 1, 2, 3 \ldots \lceil (5C \ln 2C)/SD_t \rceil + 1$, to find a $\hat{d}$ such that

$$\text{weight}(\hat{d}+1) \leq \left(1 + \frac{SD_t}{4C}\right) \text{weight}(\hat{d}). \tag{10}$$

That is, the weight at level $\hat{d}$ does not expand by much when we go out one more token.

The binary search to find such a $\hat{d}$ proceeds as follows. First, we set $d_l$ to 1 and $d_r$ to $\lceil (5C \ln 2C)/SD_t \rceil + 1$. Note that $\text{weight}(d_l)$ is at least 1 since every node has capacity at least one. Hence $\text{weight}(d_l) \geq (1 + SD_t/4C)^{d_l - 1}$. Also note that

$$(1 + SD_t/4C)^{d_r - 1} \geq (1 + SD_t/4C)^{(5C \ln 2C)/SD_t} > 2C$$

using $\log_{1+\epsilon} x \leq \ln x/\epsilon$ for small $\epsilon$ and $(1+\epsilon)^{\log_{1+\epsilon'} x} > x$ whenever $\epsilon' < \epsilon$. Since $2C$ is more than the *total* weight in the whole graph by equation (9), we must have that $\text{weight}(d_r)$ is strictly less than $(1 + SD_t/4C)^{d_r - 1}$. Thus there must be a level between $d_l$ and $d_r$ where the weight does not expand by $(1 + SD_t/4C)$. Now, we consider the level $d' = \lfloor (d_l + d_r)/2 \rfloor$. If $\text{weight}(d')$ is at least $(1 + SD_t/4C)^{d' - 1}$ then set $d_l = d'$, otherwise set $d_r = d'$. Note that a level that does not expand still exists between $d_l$ and $d_r$ and the number of levels between $d_l$ and $d_r$ has been halved. Thus, after $O(\log C)$ iterations, $d_l - d_r$ becomes a constant so $\hat{d}$ can easily be found. (We use this binary search in the algorithm so that the running time is polynomial in $\log C$ and $n$, i.e., the size of the input graph.)

Once $\hat{d}$ has been chosen, let our truncated tree $T$ be the portion of the shortest-path tree spanning nodes all of whose tokens are within distance $\hat{d}$. The *boundary of* $T$ is defined to be the set of nodes not in $T$ but adjacent to nodes of $T$. Let $\text{weight}(T)$ denote the sum of weights of all the tokens in $T$ and in the boundary of $T$. Let $B_T$ denote the boundary of $T$. Each node $v$ in $B_T$ contributes $\text{CAP}(v)$ more weight to $\text{weight}(\hat{d}+1)$ than it does to $\text{weight}(\hat{d})$. Any other node contributes no more. Hence the capacity of the boundary is exactly $\text{weight}(\hat{d}+1) - \text{weight}(\hat{d})$. It follows by choice of $\hat{d}$ that

$$\text{CAP}(B_T) \leq (SD_t/4C)\text{weight}(\hat{d}) \leq (SD_t/4C)\text{weight}(T). \tag{11}$$

Finally we delete the nodes of $T$ and its boundary, and repeat, constructing another tree, until all sources and sinks have been deleted.

**What the tree decomposition achieves:** Now suppose that all sources and sinks have been deleted. We have assigned short paths to every commodity whose source and sink lay in the same tree. Let $D_{t+1}$ denote the sum of demands of remaining commodities. We prove that $D_{t+1} \leq D_t/2$.

Each commodity not assigned a short path by the end of stage $t$ has one endpoint in a tree $T$ and one outside $T$. It follows that the union $\bigcup_T B_T$ of the boundaries of the trees separates all these commodities. Hence by the definition (1) of $S$ we have

$$SD_{t+1} \leq \text{CAP}\left(\bigcup_T B_T\right).$$

By summing (11) over all trees in the decomposition, we obtain

$$\text{CAP}\left(\bigcup_T B_T\right) \leq (SD_t/4C)\sum_T \text{weight}(T). \tag{12}$$

After we constructed each truncated tree, we deleted its nodes before forming the next, so the truncated trees are disjoint. Hence the sum of weights in the right hand side of (12) is at most the total weight in the graph, which is in turn at most $2C$ by (9). That is, we have

$$(SD_t/4C)\sum_T \text{weight}(T) \leq (SD_t/4C)2C = SD_t/2.$$

By combining these inequalities and dividing by $S$, we show that the sum of demands yet to be assigned paths is at most one-half of the sum of demands that had not been assigned paths at the beginning of the stage.

### 2.3. Turning the proof into an algorithm

In Theorem 2.1, we stated that the algorithm would find a set $W$ such that

$$\sum_i dist(s_i, t_i)d(i) = O(\log C \log D)\frac{\text{DEM}(W)}{\text{CAP}(W)}.$$

In fact, we only proved that $\sum_i dist_\ell(s_i, t_i)d(i) = O(\log C \log D)/S$. Moreover, we assumed that the value of $S$ was exactly known in advance. We show how to remedy these drawbacks in this subsection.

In the above argument we considered only $O(\log D)$ node-separators, one for each stage; namely, for each tree decomposition, we considered the node-separator consisting of the union of the tree boundaries $\bigcup_T B_T$. We therefore define the minimum "observed" cut value $S_{\text{obs}}$ for a given execution of the algorithm to be $\min_W \frac{\text{CAP}(W)}{\text{DEM}(W)}$ where $W$ ranges over these $O(\log D)$ node-separators.

All that was required in showing that each stage halves the remaining demand was that the value used as $S$ in (10) be no more than $S_{\text{obs}}$. In fact, if the value

used is only slightly more than $S_{\text{obs}}$, say no more than $(3/2)S_{\text{obs}}$, then each stage reduces the remaining demand by a factor of $4/3$.

We use this observation as follows. We maintain an overestimate $S_{\text{est}}$ of $S$, and a specific node-separator $W_{\text{est}}$ achieving a capacity-to-demand ratio of $S_{\text{est}}$. We initialize $S_{\text{est}}$ and $W_{\text{est}}$ based on an arbitrary cut in the graph that separates some demand. We run the algorithm, and compare the resulting value of $S_{\text{obs}}$ with the estimate $S_{\text{est}}$.

If $S_{\text{est}}$ is no more than $(3/2)S_{\text{obs}}$, then the algorithm will have proved equation (7) with $S_{\text{est}}$ substituted for $S$. Thus we will have found a node-separator $W$, namely $W_{\text{est}}$, satisfying (5). If $S_{\text{est}}$ is greater than $(3/2)S_{\text{obs}}$, we replace $S_{\text{est}}$ with $S_{\text{obs}}$, replace $W_{\text{est}}$ with the cut whose ratio is $S_{\text{obs}}$, and repeat. Since $S_{\text{obs}}$ cannot be less than $1/D$ and we reduce $S_{\text{est}}$ by a constant factor in each iteration, termination will occur after at most $O(\log CD)$ iterations. This proves Theorem 2.1. ∎

## 3. Balanced separators

In this section, we discuss the application of concurrent-flow based methods to finding balanced separators. In Subsection 3.1, we show how to apply our Theorem 2.1. These results are used in Section 4.

### 3.1. Cutting a fraction of the demand

The cuts that are found by the algorithm of the previous section tend to achieve a low cut ratio. However, such a cut may only separate a small fraction of the *total* demand in the flow problem. In this subsection, we consider the problem of finding a small cut that cuts a large fraction of the total demand.

In this context, capacity is interpreted as a measure of removal cost. Let $G$ be a multicommodity flow instance. Let $D$ be the total demand. Our goal is to find a minimum-cost set of nodes or edges that separates a given fraction of $D$. This problem arises in evaluating a network's resilience to worst-case faults. Note that nodes and edges may be assigned infinite cost, effectively precluding their inclusion in a separator.

For a constant $0 < \beta \leq 1$, let $OPT_\beta$ be the minimum cost of a set of nodes or edges that separates a $\beta$ fraction of the demand.

**Theorem 3.1.** *Suppose $0 < \alpha < \beta \leq 1$ are constants. There is a polynomial-time algorithm to find a set $\mathcal{W}$ of nodes or edges separating at least a fraction $\alpha$ of the demand such that the cost of $\mathcal{W}$ is $O(\log^2 n)OPT_\beta$.*

Thus the separator found by the algorithm only separates an $\alpha$ fraction of the total demand, but we compare its cost to the cheapest set of nodes or edges separating a larger fraction $\beta$.

The following method of analysis closely resembles that of [26, 28].

**Proof.** We use a greedy application of the min-cut algorithm of Theorem 2.1. That is, use the algorithm to find an approximate min-cut. Discard the nodes and

edges belonging to the cut, discard the separated demands, and then repeat on the resulting graph, until the sum of the discarded demands exceeds $\alpha D$.

Now we analyze the greedy algorithm. Let $\mathcal{E}_{opt}$ be the minimum-cost set of nodes and edges separating $\beta D$ demand. Let $\rho$ be the performance ratio of the approximate min-cut algorithm. By Theorem 2.1, $\rho = O(\log C \log D)$.

Let $Q_i$ be the total demand separated after $i$ steps of the greedy algorithm. Then

(13) $\quad Q_{i+1} = Q_i + (\text{amount of additional demand separated in } i^{th}\text{step}).$

If we removed $\mathcal{E}_{opt}$ from the original graph, we would separate at least $\beta D$ demand. Consider instead first executing $i$ steps of the greedy algorithm, and then removing $\mathcal{E}_{opt}$ from the resulting graph. The total demand separated would be at least $\beta D$, but up to $Q_i$ of that demand might have been separated by the $i$ greedy steps. Thus removing $\mathcal{E}_{opt}$ would still separate at least $\beta D - Q_i$ additional demand. It follows that the graph remaining after $i$ steps has a separator of cost no more than $OPT_\beta$ that separates at least $\beta D - Q_i$ demand.

Let $R_i = \beta D - Q_i$. By the above argument, if $R_i > 0$ then the min-cut has a capacity-to-demand ratio of at most $OPT_\beta / R_i$. Suppose the $i+1^{st}$ step finds a cut of capacity $cap_{i+1}$ separating demand $dem_{i+1}$. This cut is within a $\rho$ factor of optimal, so

(14) $$\frac{cap_{i+1}}{dem_{i+1}} \leq \rho \cdot \frac{OPT_\beta}{R_i}.$$

The algorithm terminates when the demand separated exceeds $\alpha D$, say after $L$ iterations. Hence $Q_i \leq \alpha D$ for each $i < L$, and we obtain $R_i \geq \beta D - \alpha D$. Substituting into (14) and multiplying both sides by $dem_{i+1}$ we get

$$cap_{i+1} \leq \rho \cdot \frac{OPT_\beta}{(\beta - \alpha)D} dem_{i+1}.$$

Summing over all $i$ and using the fact that $\sum_i dem_i \leq D$, we obtain

$$\sum_i cap_i \leq \rho \frac{1}{\beta - \alpha} OPT_\beta$$

as desired.

In general, we have shown only that $\rho = O(\log C \log D)$. To prove Theorem 3.1 which claims an $O(\log^2 n)$ performance ratio, we resort to a rounding technique. Let $\sigma = (\beta - \alpha)D/2n^2$. Before running the greedy algorithm, we round all demands up to the nearest multiple of $\sigma$. This increases the sum of demands by at most $(\beta - \alpha)D/2$. Moreover, these increased demands can only reduce the cost $OPT'_\beta$ of separating demands totaling $\beta D$. Because each demand is a polynomial multiple of $\sigma$, we can apply Observation 1.3 to obtain a performance guarantee $\rho' = O(\log^2 n)$ for the min-cut algorithm.

Now run the greedy algorithm but stop only after the sum of separated demands is at least $\alpha' D$, where $\alpha' = (\alpha + \beta)/2$. By the above analysis, the cost of

the separator found is at most $O(\rho')OPT'_\beta$, which is $O(\log^2 n)OPT_\beta$. However, we must account for the separated demands that were overestimated in the rounding process. We separated at least $\alpha' D$ demand, of which at most $(\beta-\alpha)D/2$ was added demand due to rounding. We are thus left with at least $\alpha' D-(\beta-\alpha)D/2$ demand having been separated. Substituting the value of $\alpha'$, we see that this is at least $\alpha D$ demand separated. ∎

Thus we have an algorithm to find a small cut that separates many of the demands. In fact, by repeated use of this algorithm, we can find a small cut that separates all demands. The following corollary provides an approximation algorithm for the generalized multiway cut problem, proving Theorem 1.4. Note that it also provides an approximation algorithm for the generalized multiway *node*-cut problem. We use the latter algorithm in the next section.

**Corollary 3.2.** *There is a polynomial-time algorithm to find an approximately minimum-cost set $\mathcal{E}$ of nodes and edges separating all commodities. The performance ratio is $O(\log^3 n)$, where $n$ is the number of nodes.*

**Proof.** Apply the algorithm of Theorem 3.1 iteratively in a greedy fashion with $\beta=1$ and $\alpha=1/2$. Each iteration halves the number of commodities remaining, so there are $O(\log n)$ iterations.

Let $OPT$ be the minimum cost of a set of nodes and edges separating all demands in the original graph. For each of the graphs arising in the above greedy iteration, the minimum cost of a set of nodes and edges separating all demands is certainly at most $OPT$, so the cost of the nodes and edges selected in each iteration is $O(\log^2 n)OPT$. This proves the theorem. ∎

## 4. Deleting CNF≡ clauses to achieve satisfiability

A CNF≡ formula $F$ is a conjunction of (possibly weighted) clauses of the form $(p_1 \equiv p_2 \equiv \cdots \equiv p_k)$, where the $p_i$'s are *literals*, i.e. either variables or negations of variables. Even in the case where there are only two literals per clause, it is NP-hard to find a minimum-weight set of clauses whose deletion yields a satisfiable formula [11]. We use a well-known construction to reformulate this problem as a problem of deleting nodes in a graph, and apply Corollary 3.2.

### 4.1. Modeling the problem

Given a weighted CNF≡ formula $F$, we first define a second weighted formula $F'$ equivalent to $F$. For each clause $c=(p_1 \equiv \cdots \equiv p_k)$ in $F$, there is an equivalent clause $\hat{c}$ in $F'$ with the same weight but with all literals negated: $\hat{c}=(\neg p_1 \equiv \cdots \equiv \neg p_k)$. We assume without loss of generality that no clauses of $F'$ appear in $F$.

Next we define the corresponding bipartite graph $G$ as follows. The node-set of $G$ consists of two disjoint sets, one consisting of all variables and their negations, and the other consisting of the clauses of $F \wedge F'$. For each literal appearing in a clause, there is an edge in $G$ between the literal and the clause.

Note that $G$ has a "symmetry" property: for each clause $c$ of $F$, $G$ contains both $c$ and $\hat{c}$. This property is crucial in the proof of the following well-known lemma (cf. [14]).

**Lemma 4.1.** *A* CNF$\equiv$ *formula $F$ is satisfiable iff no connected component of the corresponding graph $G$ contains both a literal and its negation.*

### 4.2. The algorithm

We can use the graph formulation of Subsection 4.1 in an algorithm for approximately minimum-weight deletion of clauses to achieve satisfiability, proving Theorem 1.5.

**Proof of Theorem 1.5.** The algorithm is as follows. For any weighted CNF$\equiv$ formula $F$, construct the graph $G$ as described above. For each clause, the weight of the clause is assigned as the cost to remove the clause/node from $G$. All other nodes and all edges receive infinite cost. Then apply the algorithm for generalized multiway node cut to find an approximately minimum-weight set $N$ of nodes whose removal separates each variable from its negation. We show how to derive from $N$ a solution to the clause-deletion problem.

Each node in $N$ is a clause in $F \wedge F'$. Let $N' = \{\hat{c} : c \in N\}$. Then the graph $G-(N \cup N')$ has the "symmetry" property, so Lemma 4.1 applies. Since $N$ was a multiway cut, so is $N \cup N'$. Hence no component of $G-(N \cup N')$ contains both a literal and its negation, so the formula $F$ with clauses $N \cup N'$ deleted is satisfiable. Note that only half the clauses of $N \cup N'$ appear in $F$; the others appear in $F'$. Hence the total weight of the clauses removed from $F$ is at most the weight of $N$.

The analysis is as follows. Consider a minimum-weight set $C$ of clauses whose removal renders the formula satisfiable. Let $C' = \{\hat{c} : c \in C\}$. By Lemma 4.1, the graph $G-(C \cup C')$ has the property that each variable is separated from its negation. This shows that the cost of the optimal solution to the generalized multiway cut instance is at most twice the weight of the optimal solution to the clause-deletion instance. Hence the cost of the clauses deleted by the algorithm described above is $2 \cdot O(\log^3 n)$ times optimal. ∎

## 5. Via minimization

Via minimization problems have received much attention [1, 2, 3, 4, 21, 22, 23, 25]. The general problem is as follows. One is given a layout of a circuit. The goal is to assign wire segments to layers of the VLSI chip or PC-board. Wire segments that cross in the layout must be assigned to distinct layers. Wire segments that should be electrically connected (i.e. belong to the same wire) but lie on different layers must be connected using a *via* or *contact*. One is therefore given (implicitly or explicitly), in addition to the layout, a set of points (called *junctions*) that are candidates for locations of vias. For PC boards, vias correspond to holes that must be drilled. For VLSI chips, a contact requires a certain amount of area than wires; moreover, the yield of the fabrication process depends inversely on the number of

contacts [25]. In both cases, therefore, it is desirable to minimize the number of vias.

However, the problem is NP-hard if the number of layers can exceed two or the junction degree can exceed three [4]. Researchers have proposed algorithms for cases of the via minimization problems, e.g. when the number of layers is limited to two and the maximum number of wire segments incident to a junction is limited to three. Pinter [25] also points out that in VLSI the layers are not all equivalent in performance — some layers are to be preferred to others — and he considers the problem of maximizing the amount of wire assigned to a preferred layer.

We observe that the two-layer problem can be formulated as a CNF$\equiv$ clause-deletion problem, and can therefore be approximated by the algorithm of Theorem 1.5. There is a variable for each wire segment indicating to which layer the segment is to be assigned. For each pair of crossing wire segments, there is a clause of infinite weight stating that the segments must be assigned to distint layers. For each junction, there is a clause of weight 1 stating that the incident wire segments must be assigned to the same layer. Let $F$ be the resulting CNF$\equiv$ formula. Each selection of $k$ vias and associated layer assignment corresponds to a selection of $k$ weight-1 clauses of $F$ to delete and a truth assignment that satisfies the remaining clauses.

This formulation also allows us to incorporate Pinter's suggestion. We can assign weights to the wire segments indicating the cost of not assigning them to the preferred layer, and then minimize a weighted sum of number of vias and total weight of segments not assigned to the preferred layer. We simply introduce a clause for each segment stating that the variable for that segment is assigned 1; the weight for the clause is the weight assigned to the segment.

## 6. Final remarks

We have not addressed running times of algorithms described in this paper, but we note that the algorithms of [17] and [18] can be used to quickly find approximate solutions to the concurrent flow problems.

Since the research described here was completed, there have been several improvements and related results [10, 15, 27, 31]. Garg, Vazirani, and Yannakakis [10] have improved the performance ratio in Theorem 1.4 to $O(\log k)$, where $k$ is the number of commodities. This improvement yields similar improvements in the performance ratio for minimum-weight deletion of clauses of a CNF$\equiv$ formula, for the edge-deletion graph bipartitization problem, and for the via minimization problem.

**Acknowledgements.** We gratefully acknowledge helpful conversations with Tom Leighton, John Reif, David Shmoys, and Éva Tardos.

## References


[1] F. BARAHONA: On via minimization, *IEEE Trans. Circuits Syst.* **37**, 410–416.

[2] F. BARAHONA, M. GRÖTSCHEL, M. JÜNGER, and G. REINELT: An application of combinatorial optimization to statistical physics and circuit layout design, *Operations Research* **36** (1988), 493–513.

[3] R.-W. CHEN, Y. KAJITANI, and S.-P. CHAN: A graph-theoretic via minimization algorithm for two-layer printed circuit boards, *IEEE Trans. Circuits Systems, CAS-30* (1983), 284–299.

[4] H. CHOI, K. NAKAJIMA, and C.S. RIM: Graph bipartization and via minimization, *SIAM J. of Discrete Math.* **2** (1989), 38–47.

[5] V. CHVÁTAL: Tough graphs and Hamiltonian circuits, *Discrete Mathematics* **5** (1973), 215–228.

[6] E. DAHLHOUS, D. S. JOHNSON, C. H. PAPADIMITRIOU, P. D. SEYMOUR, and M. YANNAKAKIS: The complexity of multiway cuts, *Proc. 24th ACM Symposium on Theory of Computing* (1992), 241–251.

[7] P. ELIAS, A. FEINSTEIN, and C.E. SHANNON: A note on the maximum flow through a network, *IRS Trans. Information Theory IT* **2** (1956), 117–119.

[8] L. R. FORD, JR., and D. R. FULKERSON: *Flows in Networks*, Princeton University Press, Princeton, New Jersey (1962).

[9] A. FRANK: Packing paths, circuits, and cuts — a survey, in: *Paths, Flows, and VLSI Layout*, ed. B. Korte, L. Lovász, H. J. Prömel, A. Schrijver, Springer-Verlag (1990), 47–100.

[10] N. GARG, V. V. VAZIRANI, and M. YANNAKAKIS: Approximate max-flow min-(multi)cut theorems and their applications. *Proc. 25th ACM Symposium on the Theory of Computing* (1993), 698–707.

[11] M. R. GAREY, and D. S. JOHNSON: *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman, San Francisco (1979).

[12] T. C. HU: Multicommodity network flows, *Operations Research* **11**, (1963), 344–360.

[13] D. S. JOHNSON: Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences* **9** (1974), 256–278.

[14] N. D. JONES, Y. E. LIEN, and W. T. LASSER: New problems complete for nondeterministic log space, *Math. Systems Theory*, **10** (1976), 1–17.

[15] P. KLEIN, S. PLOTKIN, and S. RAO: Planar graphs, multicommodity flow, and network decomposition. *Proc. 25th ACM Symposium on the Theory of Computing* (1991), 682–690.

[16] P. KLEIN, S. PLOTKIN, S. RAO, and É. TARDOS: Bounds on the max-flow min-cut ratio for directed multicommodity flows, Brown University Technical Report CS-93-30 (1993).

[17] 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*, to appear.

[18] F. T. LEIGHTON, F. MAKEDON, S. PLOTKIN, C. STEIN, É. TARDOS, S. TRAGOUDAS: Fast approximation algorithms for multicommodity flow problems, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing* (1991), 101–111.

[19] F. T. LEIGHTON, and S. RAO: An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms, *Proceedings, 29th Symposium on Foundations of Computer Science* (1988), 422–431.

[20] F. T. LEIGHTON, F. MAKEDON, and S. TRAGOUDAS: personal communication, 1990
[21] P. MOLITOR: On the contact-minimization problem, *Proc. 4th Annual Symposium on Theoretical Aspects of Computer Science* (1987), published as *Lecture Notes in Computer Science 247*, Springer-Verlag, New York, Berlin (1987), 420–431.
[22] N. NACLERIO, S. MASUDA, and K. NAKAJIMA: Via minimization for gridless layouts, *Proc. 24th ACM/IEEE Design Automation Conference* (1987), 159–165.
[23] N. NACLERIO, S. MASUDA, and K. NAKAJIMA: The via minimization problem is NP-complete, *IEEE Trans. Comput.* **38**, 1604–1608.
[24] C. H. PAPADIMITRIOU, and M. YANNAKAKIS: Optimization, approximation, and complexity classes, *Proceedings, 20th ACM Symposium on Theory of Computing* (1988), 229–234.
[25] R. PINTER: Optimal layer assignment for interconnect, *Journal of VLSI and Computer Systems* **1** (1984), 123–137.
[26] D. PLAISTED: A heuristic algorithm for small separators in arbitrary graphs, *SIAM J. Comput. 19* (1990), 267–280.
[27] S. PLOTKIN, and É. TARDOS: Improved bounds on the max-flow min-cut ratio for multicommodity flows. *Proc. 25th ACM Symposium on the Theory of Computing* (1993), 691–697.
[28] S. RAO: Finding near optimal separators in planar graphs, *Proceedings, 28th Annual Symposium on Foundations of Computer Science* (1987), 225–237.
[29] A. SCHRIJVER: Min-max results in combinatorial optimization, *Mathematical Programming, the state of the art* Springer-Verlag, Bonn (1983), 439–500.
[30] F. SHAHROKHI, and D. MATULA: The maximum concurrent flow problem, *Journal of the ACM* **37**:2 (1990), 318–334
[31] S. TRAGOUDAS: VLSI partitioning approximation algorithms based on multicommodity flow and other techniques, PhD thesis, University of Texas at Dallas (1991).
[32] M. YANNAKAKIS: Edge-Deletion problems, *SIAM J. Computing* **10**, (1981), 297–309.

Philip Klein

*Department of Computer Science*
*Brown University*
*Providence, RI 02912, U.S.A.*
klein@cs.brown.edu

Satish Rao

*NEC Research Institute*
*4 Independence Way*
*Princeton, NJ 08540, U.S.A.*
satish@research.nj.nec.com

Ajit Agrawal

*Exa Corporation*
*125 Cambridge Park Drive*
*Cambridge, MA 02140*
agarwal@exa.com

R. Ravi

*DIMACS Center*
*P. O. Box 1170*
*Rutgers University*
*Piscataway, NJ 08855–1179, U.S.A*
rravi@dimacs.rutgers.edu