An Approximate Max-flow Min-cut Relation for Multicommodity Flow and its Applications

Philip Klein, Ajit Agrawal and R. Ravi[1]
Satish Rao[2]

Technical Report No. CS-91-17
March 1991

[1]Department of Computer Science,Brown University, Providence, RI 02912
[2]Aiken Computation Laboratory, Harvard University,
Cambridge, MA 02138

An approximate max-flow min-cut relation for multicommodity flow and its applications

Philip Klein[1] [2] Ajit Agrawal[†] R. Ravi[†] Satish Rao[3]

[1]Research supported by an NSF grant CCR-9012357

[2]Brown University, Providence, RI 02912. Reseach 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, Amendment 1.

[3]Aiken Computation Laboratory, Harvard University, Cambridge, MA. Research supported by ONR Grant # N0014-88-k-0243.

Abstract

In this paper, we prove the first approximate max-flow min-cut theorem for general 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 edge 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 approximation algorithms for minimum deletion of clauses of a 2-CNF$\equiv$ formula, via minimization, and other problems.

1 Introduction

Max-flow min-cut theorems

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 [3] and of Elias, Feinstein and Shannon [2] 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 [6] 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. As was demonstrated fairly early, however, no such theorem can hold in general for all multicommodity flow instances.

Leighton and Rao [10] 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.

Our approximate max-flow min-cut theorem for arbitrary multicommodity flow

In this paper, we have gone a step further. We have proved an approximate max-flow min-cut theorem that holds for *all* multicommodity flow problems. Our theorem is the first for arbitrary multicommodity flow.

Theorem 1.1 *Consider a multicommodity flow problem where the sum of the demands is D and the sum of the 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 its demand 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. By appropriate choices of source-sink pairs, one can use this algorithm to find cuts that are (approximately) minimum subject to certain criteria. Thus we significantly enhance the generality of the cut-based approach to approximation algorithms.

New approximation algorithms: Deleting 2-CNF≡ clauses to achieve satisfiability, and applications

As a consequence, we obtain new approximation algorithms. Our first result is the following theorem:

Theorem 1.2 *Given a 2-CNF≡ formula with weighted clauses of the form $(P \equiv Q)$, one can in polynomial time find an approximately minimum-weight set of clauses whose deletion yields a satisfiable formula.*[1] *The performance ratio is $O(\log^3 n)$.*

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 2-CNF≡ clause deletion provides a good basis for such algorithms; one can sometimes state the structural property in the language of 2-CNF≡ 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 [16]. Using our variant of 2-CNF≡ in which each clause has the form $(P \equiv Q)$, it is easy to state 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.

Using similar techniques, we can solve some other problems. The *via minimization problem* (discussed in [1]) 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.

Theorem 1.3 *One can in polynomial time approximately solve*

- *the edge-deletion graph bipartization problem,*
- *the via minimization problem.*

[1] A complementary result was obtained by Johnson [7] in 1974; he showed that 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 could 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 [12] of expressing NP-complete problems with quantified sentences in order to study their approximability.

The performance guarantee is the same as that in Theorem 1.2.

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

2.1 Preliminaries

In this section we prove an approximate max-flow min-cut theorem that forms the basis for the 2-CNF$\equiv$ clause deletion algorithm of Section 4 and the applications: bipartisation and via minimisation.

Consider a multicommodity flow problem, where we have a network G with edge-capacities $\textsc{cap}(vw)$, 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 *concurrent flow* [15] of *capacity utilization* u is a multicommodity flow satisfying the demands and subject to the edge-capacities $u \cdot \textsc{cap}(vw)$. The concurrent flow problem is to find a flow f with minimum capacity utilisation. 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.

It is useful to think of each pair s_i, t_i of commodity endpoints as a *demand edge* s_it_i in the graph G, with a weight $\textsc{dem}(s_it_i)$ equal to the demand associated with the commodity. Thus G has two kinds of edges, demand edges and ordinary capacity edges.

Each subset V of the node set of G defines a *cut*, namely the set $\Gamma(V)$ of edges with exactly one endpoint in V. For a subset E of the edge set of G, let $\textsc{cap}(E)$ denote the sum of capacities of capacity edges in E, and let $\textsc{dem}(E)$ denote the sum of weights of demand edges in E.

The analogue of a minimum cut is a cut $\Gamma(A)$ that minimizes the ratio of capacity to demand. That is, define the *minimum cut ratio* S as follows:

$$S = \min_A \frac{\textsc{cap}(\Gamma(A))}{\textsc{dem}(\Gamma(A))} \tag{1}$$

It is easy to see that the value of the capacity utilisation is at least $1/S$. In other words, if the minimum cut ratio is less than one, then the multicommodity flow problem is infeasible.

Let ℓ be any length function assigning lengths to the capacity edges of the graph G, and let $dist_\ell(v, w)$ be the resulting shortest path distance between v and w using capacity edges. The linear programming dual to the problem of minimising the capacity utilization u is maximising the sum

$$\sum_{\text{demand edge } st} dist_\ell(s, t)\textsc{dem}(st) \tag{2}$$

subject to the constraint

$$\sum_{\text{capacity edge } vw} \textsc{cap}(vw)\ell(vw) = 1 \tag{3}$$

In particular, the value of (2) is always at most the value of the capacity utilisation u, and, when ℓ maximizes (2) and f minimises u, then 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 subset A such that*

$$\sum_i dist_\ell(s_i, t_i)d(i) = O(\log C \log D)\frac{\textsc{dem}(\Gamma(A))}{\textsc{cap}(\Gamma(A))}$$

where C is the sum of all capacities and D is the sum of all demands.

Corollary 2.1
(Approximate Max-flow Min-cut Theorem)
We have

$$\frac{S}{O(\log C \log D)} \leq 1/u \leq S$$

In particular, for a multicommodity flow to be feasible, it is necessary that the min cut ratio S be at least 1, and it is sufficient that it be at least $O(\log C \log D)$.

Corollary 2.2
(Approximation Algorithm for Min Cut)
The cut $\Gamma(A)$ found by the algorithm of Theorem 2.1 approximately minimizes the ratio

$$\frac{\textsc{cap}(\Gamma(A))}{\textsc{dem}(\Gamma(A))}$$

to within a factor of $O(\log C \log D)$.

The following theorem is useful in applications where either the capacities or demands are exponentially large.

Theorem 2.2 *If either C or D is polynomial in n, the number of nodes of G, then the performance guarantee can be improved to be $O(\log^2 n)$.*

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 ℓ 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

$$\sum_i \text{length}(P(s_i,t_i)) \cdot \text{DEM}(s_i t_i) = O((1/S)\log C \log D) \quad (5)$$

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. Let D_{t+1} be 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 (5) is $O((1/S)\log C))$. It remains to describe a procedure for decomposing G into trees of small height such that $D_{t+1} \le D_t/2$, for then the number of stages is at most $\log D$, and the total contribution of all stages to the left-hand side of (5) is $O((1/S)\log C \log D)$.

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

$$\ell_t(vw) := \lceil C\ell(vw) \rceil \quad (6)$$

for each edge vw. Notice that since $\sum \ell(vw)\text{CAP}(vw) \le 1$ and that rounding up adds only one extra token for each edge, we have that the total token weight is

$$\sum \ell_t(vw)\text{CAP}(vw) \le 2C \quad (7)$$

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, as well as all incident capacity edges and demand edges, 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 $\lceil (9C\ln 2C)/SD_t \rceil + 1$, then let T be this tree. Otherwise, we must select a distance $\hat{d} \le \lceil (9C\ln 2C)/SD_t \rceil + 1$ at which to truncate the tree.

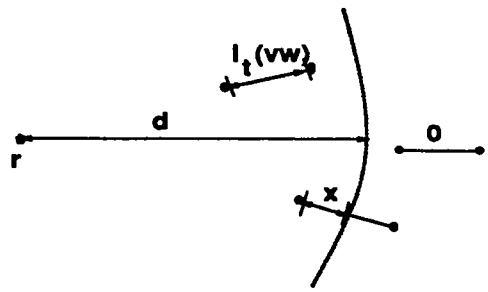

Figure 1: If one of the endpoints of edge vw is within distance $d - \ell_t(vw)$ of r, then all $\ell_t(vw)$ of the edge's tokens are within distance d. If neither endpoint is within distance d, then none of the edge's tokens is within distance d. If one endpoint is at a distance $d - x$ of r, and $x \le \ell_t(vw)$, then x of the edge's tokens are within distance d.

There is a natural notion of how much of an edge's tokens lies within a given distance d of the root r, as illustrated in Figure 1. Denote this value by amount(vw, d). We determine the total *weight* of the subgraph within distance d of r, defined as follows.

$$\text{weight}(d) = \sum_{\text{edge } vw} \text{amount}(vw, d) \cdot \text{CAP}(vw)$$

We use binary search on the range of distances $d = 1, 2, 3 \ldots \lceil (9C\ln 2C)/SD_t \rceil + 1$, to find a $\hat{d}$ such that

$$\text{weight}(\hat{d}+1) \le (1 + \frac{SD_t}{8C})\text{weight}(\hat{d}) \quad (8)$$

The binary search to find such a $\hat{d}$ proceeds as follows. First, we set d_l to 1 and d_r to $\lceil (9C\ln 2C)/SD_t \rceil + 1$. Notice, that weight$(d_l)$ is at least $(1 + SD_t/8C)^{d_l - 1}$ and hence at least 1 since every edge has capacity one. Also notice that weight(d_r) is strictly less than $(1 + SD_t/8C)^{d_r - 1}$ since

$$(1 + SD_t/8C)^{d_r - 1} \ge (1 + SD_t/8C)^{(9C\ln 2C)/SD_t} \ge 2C,$$

and $2C$ is the *total* weight in the whole graph by equation 7. Thus, there must be a level between d_l and d_r where the weight does not expand by $(1 + SD_t/8C)$. Now, we consider the level $d' = \lfloor (d_l + d_r)/2 \rfloor$. If weight$(d')$ is at least $(1 + SD_t/8C)^{d' - 1}$ then set $d_l = d'$, otherwise set $d_r = d'$. Notice 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 T be the portion of the shortest-path tree spanning nodes of distance $\leq \hat{d}$ from the root, and define weight(T) = weight$(\hat{d})$. Let $\mathcal{E}_T$ denote the set of edges currently incident to T. Note that

$$\begin{aligned} \text{CAP}(\mathcal{E}_T) &\leq \text{weight}(\hat{d}+1) - \text{weight}(\hat{d}) \\ &\leq \frac{SD_t}{8C}\text{weight}(T) \end{aligned} \quad (9)$$

Finally we delete the nodes of T and all incident edges, and repeat, constructing another tree.

What the tree decomposition achieves: Now suppose that all sources and sinks have been deleted. We've 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 $D_{t+1} \leq D_t/2$. Let D_T denote the sum of demands of commodities i with source s_i in T and sink t_i outside T. The trees partition D_{t+1}. Let $\Gamma(T)$ denote the boundary of T, i.e. the set of edges with exactly one endpoint in T. We have the following

$$SD_{t+1} = \sum_T SD_T \leq \sum_T \text{CAP}(\Gamma(T))$$

where the inequality follows from the definition of S. Since each edge belongs to the boundary of at most two trees and belongs to at least one $\mathcal{E}_T$, we have

$$\sum_T \text{CAP}(\Gamma(T)) \leq 2\sum_T \text{CAP}(\mathcal{E}_T).$$

Using (9), we infer

$$2\sum_T \text{CAP}(\mathcal{E}_T) < \frac{2SD_t}{8C}\sum_T \text{weight}(T)$$

In constructing the tree decomposition, once we count an edge towards the weight of the tree, we delete it. Hence $\sum_T \text{weight}(T)$ is no more than the total weight in the whole graph, which was shown to be $2C$ in (7). Putting these inequalities together, we obtain

$$SD_{t+1} \leq SD_t/2.$$

2.3 Turning the proof into an algorithm

In Theorem 2.1, we stated that the algorithm would find a set A such that $\sum_i dist(s_i,t_i)d(i) = O(\log C \log D)\frac{\text{DEM}(\Gamma(A))}{\text{CAP}(\Gamma(A))}$. In fact, we only proved that $\sum_i dist_l(s_i,t_i)\text{DEM}(s_it_i) = O(\log C \log D)/S$. Moreover, we assumed that the value of S was exactly known in advance.

We remedy this by making two observations. First, note that the proof still holds when S is replaced by the "observed value of S," which we denote by S_{obs}. The value of S_{obs} is defined to be the minimum value of $\frac{\text{CAP}(\Gamma(T))}{\text{DEM}(\Gamma(T))}$ over all trees T actually appearing in some decomposition during the algorithm. Thus if we knew S_{obs} before running the algorithm, we could substitute it for S, and proceed as before. We would end up showing that $\sum_i dist_l(s_i,t_i)\text{DEM}(s_it_i) = O(\log C \log D)/S_{\text{obs}}$. By letting A be the nodes spanned by the tree that determined the value of S_{obs}, we obtain the bound in Theorem 2.1.

Of course, we can't expect to know the value of S_{obs} before running the algorithm. However, if the value substituted for S in (8) is no more than $(3/2)S_{\text{obs}}$, then we are still guaranteed that the remaining demand diminishes by a factor of 3/4 in each stage. We use this observation as follows. Start with some estimate of S, derived from looking at an arbitrary cut in the graph. Run the algorithm, and compare the resulting value of S_{obs} with the estimate. If the estimate is no bigger than $(3/2)S_{\text{obs}}$, then we are done, otherwise, reduce the estimate by a factor of 3/2, and repeat.

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 all 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 section, we present some results about finding a small cut that cuts a large fraction of the total demand.

In particular, we consider a cut that cuts half the demand pairs.

Consider a multicommodity flow problem specified by $G = (V, E)$. We denote the capacity edges of G by E_C and denote the demand edges by E_D. Now we define E_{opt} to be a minimum capacity edge set such that if $\{H_1, ..., H_l\}$ are the connected components of $G_{\text{opt}} = (V, E_C - E_{\text{opt}})$ then

$$\sum_{s_i \in H_j, t_i \in H_k, j \neq k} d(i) \geq D/2.$$

That is, the edge set E_{opt} is the smallest cut that separates half the demand in G.

We can prove the following result about approximating such a cut.

Theorem 3.1 *There is a polynomial time algorithm, A, that will find a set of edges E_A such that if $\{H_1, ..., H_{l'}\}$ are the connected components of $G' = (V, E_C - E_A)$ then*

$$\sum_{s_i \in H_j, t_i \in H_k, i \neq k} d(i) \geq D/3,$$

and

$$\text{CAP}(E_A) \leq O(\log C \log D)\text{CAP}(E_{\text{opt}}).$$

Proof Sketch: We simply find an approximate sparsest cut using the algorithm of the previous section, remove the edges in the cut from the graph and repeat until $D/3$ demand sources and sinks are separated. We can show that the union of the cuts have small cost by using the fact that they have small sparsest cut cost. □

Thus we have an algorithm that finds a small cut that separates many of the demands. In fact, using the above theorem, we can find a small cut that separates all demands. First, we say an edge set, E_A, *completely cuts* the demand in a flow problem if there is no commodity whose source and sink are in the same connected component of $G' = (V, E_C - E_A)$. Now we state the following lemma.

Lemma 3.1 *There is a polynomial time algorithm, A, that given a flow problem on G will find a set of edges E_A that completely cuts the demand in G such that*

$$\text{CAP}(E_A) = O(\log C \log^2 D)\text{CAP}(E_{\text{opt}}),$$

where E_{opt} is the smallest capacity edge set that completely cuts G.

4 Approximately minimizing unsatisfied clauses in a 2-CNF≡ formula

Given a 2-CNF≡ formula F with weighted clauses of the form $(p \equiv q)$, finding a minimum weighted set of clauses whose deletion yields a satisfiable formula is known to be NP-complete [4]. We use a well-known construction to reformulate this problem as a problem of deleting edges in a graph. We then show how to approximate this edge-deletion problem by defining a flow function in this graph and using the ideas presented in Section 3.

Defn: We define a weighted 2-CNF≡ formula to be a conjunction of weighted clauses of the type $(p \equiv q)$ where p and q are literals, and $\equiv$ refers to logical equivalence.

4.1 Modeling the problem

Given a weighted 2-CNF≡ formula F, we define the corresponding boolean graph $G(F)$ as follows. Let the node-set $V(G)$ be the set of all the literals that occur in F. For every clause $(p \equiv q)$ in F, include the *undirected* edge (p, q) in $G(F)$. This edge is assigned a weight equal to the weight of the corresponding clause $(p \equiv q)$ in F. We shall henceforth use the terms "literal" and "vertex" interchangeably.

We can easily prove the following lemma [8].

Lemma 4.1 *A weighted 2-CNF≡ formula F is satisfiable iff no connected component of the graph G(F) constructed as above contains both a literal and its negation.*

By virtue of Lemma 4.1, we have the following corollary.

Corollary 4.1 *Given a weighted 2-CNF≡ formula F, finding a minimum weight set of clauses whose deletion yields a satisfiable formula is equivalent to finding a minimum weight set of edges in the corresponding boolean graph G(F) such that its deletion from G(F) leaves the graph with no connected component containing both a literal and its negation.*

4.2 The algorithm

Define a general concurrent flow problem on the boolean graph $G(F)$ by assigning one unit of a commodity to flow from each literal x_i to $\neg x_i$ occuring in the same connected component. From Lemma 3.1, we can find a set of edges in $G(F)$ whose cost is at most within a factor of $O(\log^3 n)$ of the minimum cost separator that completely cuts all the demands in $G(F)$. From Corollary 4.1, such a separator corresponds to the minimum weight set of clauses that we are looking for and we have the following theorem.

Theorem 4.1 *There exists an algorithm that finds an approximately minimum-weight set of clauses to delete from a given weighted 2-CNF≡ formula to leave it satisfiable. The performance ratio is $O(\log^3 n)$.*

5 Final remarks

We have presented an approximate min-max theorem for general multicommodity flow. Recently, we have been able to generalise this theorem to apply to hypergraph networks; using this theorem, we can handle CNF≡ clauses with an arbitrary number of literals per clause.

We have not addressed running times of algorithms described in this paper, but we note that the algorithm of [9] can be used to quickly find approximate solutions to the concurrent flow problems.

Our approximate min-max theorem, while more general than that in [10], does not guarantee as good an approximation. In contrast to [10], we have no example demonstrating that our bound is existentially tight. It is therefore an open problem to improve our bound or show it cannot be improved.

Acknowledgements

We gratefully acknowledge helpful conversations with Tom Leighton, John Reif, and David Shmoys.

References

[1] H. Choi, K. Nakajima and C.S. Rim, "Graph bipartisation and via-minimisatiuon", *SIAM J. of Discrete Maths Vol. 2, No. 1* (1989), pp. 38-47.

[2] P. Elias, A. Feinstein and C.E. Shannon, "A note on the maximum flow through a network", *IRS Trans. Information Theory IT 2* (1956), pp. 117-119.

[3] L. R. Ford, Jr., and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, New Jersey (1962).

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

[5] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York (1980).

[6] T. C. Hu, "Multicommodity network flows", *Operations Research 11*, (1963), pp. 344-360.

[7] D. S. Johnson, "Approximation algorithms for combinatorial problems", *Journal of Computer and System Sciences 9* (1974), pp. 256-278.

[8] N. D. Jones, Y. E. Lien and W. T. Lasser, "New problems complete for nondeterministic log space", *Math. Systems Theory, 10* (1976), pp. 1-17.

[9] P. Klein, C. Stein, and É. Tardos, "Leighton-Rao might be practical: faster approximation algorithms for concurrent flow with uniform capacities", *Proceedings, 22nd ACM Symposium on Theory of Computing* (1990), pp. 310-321.

[10] 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), pp. 422-431.

[11] F. T. Leighton, F. Makedon, and S. Tragoudas, personal communication, 1990

[12] C. H. Papadimitriou and M. Yannakakis, "Optimisation, approximation, and complexity classes", *Proceedings, 20th ACM Symposium on Theory of Computing* (1988), pp. 229-234.

[13] P.D. Seymour, "Matroids and multicommodity flows", *European Journal of Combinatorics 2* (1981), pp. 257-290.

[14] P.D. Seymour, "On odd cuts and planar multicommodity flows", *Proc. London Math. Soc. 42* (1981), pp. 178-192.

[15] F. Shahrokhi and D. Matula, "The maximum concurrent flow problem," *Journal of the ACM* 37:2 (1990), pp. 318-334

[16] M. Yannakakis, "Edge-Deletion problems", *SIAM J. Computing 10*, (1981), pp. 297-309.