The Network Inhibition Problem

(extended abstract)

Cynthia A. Phillips *

ABSTRACT

In the network inhibition problem, we wish to find the most cost-effective way to reduce the ability of a network to transport a commodity. Potential applications include disabling polluting pipe systems, military supply lines, and illicit drug networks. We wish to compute a fixed-budget attack strategy which will maximally inhibit a network's capability.

In this paper we introduce the network inhibition problem, prove several NP-completeness results, and give algorithms for planar and outerplanar graphs. We prove the problem is strongly NP-complete for general graphs, even of degree at most 3, and weakly NP-complete for series-parallel graphs, grid graphs, bandwidth-k graphs with $k \geq 3$, and Halin graphs (and thus for k-outerplanar graphs with $k \geq 2$). We describe three pseudopolynomial-time algorithms for planar graphs and show how to convert them into fully polynomial-time approximation schemes (FPTAS). Techniques used in these algorithms also give a FPTAS for the restricted shortest path problem in general graphs that is a factor of $|E|$ faster than the best previous scheme whose running time does not depend on the input numbers and the fastest overall for moderately dense graphs and reasonable error parameters. Finally, we describe a polynomial-time algorithm for inhibition of outerplanar graphs. Thus, network inhibition separates outerplanar graphs from series-parallel graphs, separates 1-outerplanar from 2-outerplanar graphs, and separates bandwidth-2 graphs from bandwidth-3 graphs.

*Algorithms and Discrete Mathematics Department (Dept. 1423), Sandia National Laboratories, P. O Box 5800, Albuquerque, NM 87185-5800 Email caphill@cs sandia.gov. This work was supported by the U.S Department of Energy under Contract DE-AC04-76DP00789

25th ACM STOC '93-5/93/CA,USA

© 1993 ACM 0-89791-591-7/93/0005/0776...$1.50

1 INTRODUCTION

Suppose a system of pipes from a sewer or factory is discharging toxic waste into a lake. The cost of filtering or chemically neutralizing the waste is pipe-dependent. How do you expend a fixed budget modifying the pipe system so as to minimize the flow of waste?

We model this flow-interdiction problem as the choice of attack strategy upon a transportation network. In the classical network flow problem, the owner of a network computes a strategy for optimally utilizing a capacitated transportation network. In the network inhibition problem, we assume the role of an adversary who wishes to optimally utilize limited destructive resources so as to maximally inhibit a network's capability. Other potential applications include disabling military supply lines, communication networks, a national energy system, or an illicit drug network. In this paper we describe the rich complexity structure of this new problem.

Before defining network inhibition, we describe the networks to be attacked. *Flow networks* model the movement of material through a static system of channels, for example water through pipes, electronic communication over wires, or supplies over roadways. Informally, in the *max flow* problem, we wish to maximize the material flowing from a source node to a sink node where the flow obeys capacity constraints on each edge and conservation constraints on each non-source/sink node. The max flow problem can be solved in polynomial time. The asymptotically best deterministic algorithms to date are due to King, Rao and Tarjan [28] for dense graphs ($|E| > |V|^{1+\epsilon}$, for any constant $\epsilon > 0$) and Goldberg and Tarjan [16] for sparser graphs. The best randomized algorithm to date is due to Cheriyan, Hagerup, and Mehlhorn [8]. Reif [38], Hassin and Johnson [23], and Fredrickson [11, 12] have faster algorithms for planar graphs.

Problem Statement: In the network inhibition problem, in addition to a *capacity* c_{uv}, each edge (u, v) of a flow network has a *destruction cost* d_{uv}, which corresponds to the cost an attacker incurs removing the

edge. We assume the destruction is linear, so that an attacker can pay αd_{uv} units of budget for $0 \leq \alpha \leq 1$ and remove αc_{uv} units of capacity from the edge. Edges can be indestructible ($d_{uv} = \infty$). In section 6 we describe extensions to other models such as not allowing partial cuts and modeling remaining capacity of individual edges by any piecewise-linear convex function of budget. The latter case can model the situation where it is relatively easy to destroy some of an edge, but relatively hard to assure removal of the final units of capacity.

Suppose we are given a capacitated flow network G with destruction costs and a fixed *budget* B. Let $\mathcal{G}_B$ be the family of reduced-capability networks obtained by spending at most B units of budget reducing capacities of edges in graph G. The *network inhibition problem* is to choose the member of the family $\mathcal{G}_B$ with the minimum max flow. This corresponds to choosing an attack strategy that minimizes the capability left in the resulting damaged network.

If the nodes of a flow network are partitioned into two sets, S and T with $s \in S$ and $t \in T$, the set of edges that go between sets S and T is called a *cut*. The *capacity* of a cut is the sum of the capacities of the edges in the cut. A *minimum cut* is a cut with smallest capacity (or the capacity of this cut depending upon context). By Ford and Fulkerson's classic max flow/mincut theorem [10], the value of a maximum flow is equal to the minimum cut. Thus the network inhibition problem can be framed as a cut problem: spend the budget to minimize the minimum cut. All our algorithms are based upon this version of the problem. We refer to the capacity left after attacking a cut (network) with some budget as the *residual capacity* of the cut (network).

In this paper we prove that the network inhibition problem is strongly NP-complete for general graphs, even of degree at most 3, and fully resolve the complexity of the network inhibition problem for all subclasses of planar graphs described in Johnson's 16th NP-completeness column[25]. Figure 1, a slightly modified version of the figure in Johnson's column[25, page 441], summarizes our complexity results. We show that network inhibition is weakly NP-complete for series-parallel graphs, grid graphs, bandwidth-k graphs with $k \geq 3$, and Halin graphs (thus for k-outerplanar graphs with $k \geq 2$). Since network inhibition is weakly NP-complete for subclasses of planar graphs, inhibition of planar networks is at least weakly NP-complete.

We show that these classes are no harder than weakly NP-complete by presenting several pseudopolynomial-time algorithms for planar graphs. These algorithms take advantage of the one-to-one correspondence between cuts in a planar graph and cycles in its dual. Exploiting special structure of optimal solutions, we can use dynamic programming on paths in the dual to solve the original cut problem. This general technique has been used before [4, 19] to find polynomial-time algorithms for planar graphs for NP-complete cut problems, and to speed up computation of min cuts for planar graphs [11, 12, 23, 24, 38].

Using a Floyd-Warshall-style all-pairs-shortest-paths algorithm, we can compute optimal attack strategies for planar networks in time $O(n^3C)$, where n is the number of nodes in the graph and C is the optimal residual capacity of the network (minimum max flow). An algorithm based upon Dijkstra's shortest path algorithm runs in time $O(n^2C \lg(nC))$. Adding techniques for searching in Monge arrays improves this bound to $O(nkC \lg nC)$, where k is the minimum number of nodes on a path in the dual from any node representing a face adjacent to source s to any node representing face adjacent to sink t. By interchanging the roles of destruction costs and capacity in the above algorithms, we can replace the optimal residual capacity C in the running times by the budget B. If either of these quantities is bounded by a polynomial in the graph size n, then the algorithms run in polynomial time and the one based on Dijkstra's algorithm and Monge arrays is asymptotically superior. This case should dominate in practice since capacities and/or destruction costs will be fixed-precision estimates. If both these quantities are exponential, the Floyd-Warshall-based algorithm is superior, but it runs in exponential time.

We can take the process one step further by converting the pseudopolynomial algorithms into *fully polynomial-time approximation schemes (FPTAS)*. A FPTAS accepts an error parameter $\epsilon > 0$ and after running for time polynomial in the input size and $1/\epsilon$, returns an answer within $(1+\epsilon)$ times the optimal. Thus as the error parameter approaches 0, the approximation is better, but the running time increases; conversely, as the error parameter is allowed to grow away from 1, the approximation gets worse, but the running time decreases. Strongly NP-complete problems cannot have a FPTAS (unless $P = NP$)[34]. The FPTAS based upon Monge arrays runs in time $O(n^2k(1+1/\epsilon)(\lg n + \lg(1+1/\epsilon)))$ and the Floyd-Warshall version runs in time $O(n^4(1+1/\epsilon))$.

To complete the complexity characterization of the network inhibition problem, we give a trivial $O(n^3)$-time algorithm for inhibition of outerplanar networks. Thus network inhibition separates outerplanar graphs from both 2-outerplanar and series-parallel graphs, and separates bandwidth-2 graphs from bandwidth-3 graphs. We know of no other problem which does this.

To our knowledge, the network inhibition problem has never been studied before. In particular, no one has studied network inhibition in a model which allows partial attacks upon edges or cuts. Cunningham [9] has studied the notion of network inhibition using different models of network degradation, destroying full network cuts only, and computing an attack with minimum cost/benefit ratio. Computing this ratio is fundamen-

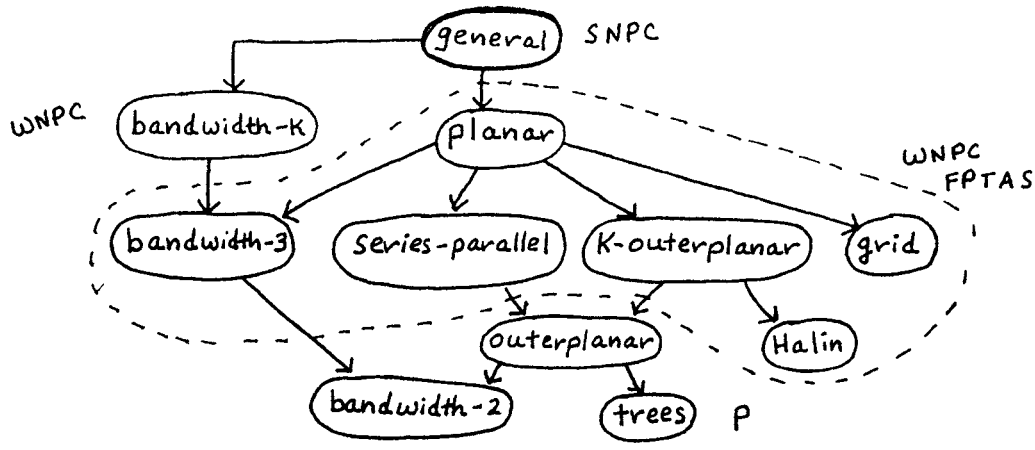

Figure 1: Summary of our complexity results. Directed edges indicate proper inclusion. General graphs are strongly NP-complete. Bandwidth-k graphs ($k \geq 4$), are at least weakly NP-complete (WNPC). Classes inside the dashed circle are weakly NP-complete (WPNC) and have a fully polynomial-time approximation scheme. Classes under the dashed line are in P.

tally easier than computing a fixed-budget attack[1], but is less useful since the attack with smallest ratio could have insufficient benefit or exorbitant cost. Our model of network inhibition provides an attacker with a strategy that optimally uses exactly the amount of resources he is willing and able to commit to the attack. It also assigns some benefit to partial attacks upon edges and partial attacks upon cuts. Thus the transport capability of the network degrades gracefully as a function of attack budget.

Other related work also considers only total destruction of cuts. Klein *et al* [30] give an $O(\lg^3 n)$-approximation algorithm for finding a minimum-weight set of edges (nodes) whose removal disconnects each pair of a given set of vertex pairs ($n = |V|$). The error factor drops to $O(\lg^2 n)$ for disconnecting a fraction of the pairs (possibly weighted). Bienstock [5] computes minimum cuts in planar networks where edges "close" to each other in an embedding can be removed together at a single cost. Hammer *et al* [20] examine the structure of *cut-threshold graphs*, where removal of a set of edges disconnects a network if and only if the set has sufficient weight.

Using techniques from the Dijkstra-based FPTAS for planar network inhibition, we give a FPTAS for the *restricted shortest path problem*, defined as follows. Given a graph $G = (V, E)$ with designated vertices s and s', *length* $l_{uv} \in \mathcal{Z}^+$ and *transit time* $t_{uv} \in \mathcal{Z}^+$ for each edge (u, v), and deadline T, we wish to find a path P from s to s' in the graph such that $\sum_{(u,v)\in P} t_{uv} \leq T$ and $\sum_{(u,v)\in P} l_{uv}$ is minimum. This problem has been studied extensively beginning as early as 1966 [26] and culminating in Hassin's two FPTAS[22]. The first runs in time $O((mn^2/\epsilon)\lg(n/\epsilon))$, where $m = |E|$ and $n = |V|$, and the second runs in time $O(\lg\lg U((mn/\epsilon)+\lg\lg U))$ where U is an upper bound on the optimal path length. Our FPTAS runs in time $O(mn/\epsilon + (n^2/\epsilon)\lg(n/\epsilon))$ for $\epsilon < 1$, which is asymptotically superior to Hassin's first scheme by at least a factor of m, and to the second when $m > cn\lg(n/\epsilon)/\lg\lg U$, for some constant c. For example, if $\epsilon > 1/n^k$, for some constant k, then our scheme is better when $m > cn\lg n/\lg\lg n$. We get a lower bound on U because a FPTAS has error at most $\epsilon \cdot$ opt $\leq \epsilon U$. Since $l_{uv} \in \mathcal{Z}^+$, $\epsilon U < 1$ requires optimality. If $\epsilon > 1/n^k$ then we have $U \geq n^k$, since smaller values of U require optimality and we are better off using the pseudopolynomial-time algorithm from which the FPTAS is derived.

The remainder of the paper is organized as follows. Section 2 considers special cases of the network inhibition problem, describes the structure of an optimal solution, and provides a simple exponential-time algorithm. Section 3 discusses the NP-completeness of network inhibition (strongly NP-complete for general graphs, even of bounded-degree, and weakly NP-complete for series-parallel graphs, Halin graphs, grid graphs, and bandwidth-k graphs for $k \geq 3$). Section 4 gives a trivial polynomial-time algorithm for outerplanar graphs and describes the pseudopolynomial-time algorithms for undirected planar graphs. Section 5 converts the pseudopolynomial-time algorithms into FPTAS. Section 6 gives the improved FPTAS for the restricted shortest path problem, and describes extensions of our inhibition algorithms to include different models of inhibition of a single edge and to allow inhibition of planar multigraphs and directed planar graphs. Finally, in section 7, we give some open problems, primarily concerning the modeling of network inhibition, and offer some concluding remarks.

[1] Cunningham's algorithms are strongly polynomial-time. His problems become NP-complete in a fixed-budget model [33] A ratio attack in our model can be computed in weakly polynomial time[33].

2 STRUCTURE OF AN OPTIMAL SOLUTION

In this section we explore the structure of an optimal solution to the network inhibition problem. We describe special cases of the problem that can be solved in polynomial time for any network. We show that applying a greedy strategy to some cut of the network yields an optimal solution and use this observation to obtain a simple exponential-time network inhibition algorithm. Finally we describe the sensitivity of the optimal attack strategy to small changes in budget.

The first special case is budget $B = 0$. This is simply the polynomially-computable min cut problem. The second special case is at the opposite extreme: the *minimum cost maximum destruction* problem. If there is a cut where all the destruction costs are finite, then we wish to compute the minimum cost way to sever the source s from the sink t (*min cost total destruction*). If no cut has finite cost, then we wish to compute the minimum cost necessary to make the connection between nodes s and t as small as possible. We can compute min cost max destruction by finding a min cut using capacities c'_{uv} defined as follows:

$$c'_{uv} = \begin{cases} d_{uv} & \text{if } d_{uv} < \infty \\ Dc_{uv} & \text{otherwise} \end{cases}$$

The parameter D is guaranteed to be bigger than the finite destruction cost of any cut (*eg.* $D = |E| \max d_{uv}$) Computing the minimum cut for the network with capacities c'_{uv} finds a cut with the smallest possible indestructible capacity. Among all such cuts, it finds the one with the smallest cost to achieve that capacity.

To minimize the minimum cut in the damaged network, the set of edges that we attack should be a subset of some mimimal cut of the original network. We now argue that for a fixed cut, the optimal attack strategy is greedy. The optimal attack strategy upon a fixed cut with edge set $E_c \subseteq E$ is the one that removes as much of the cut's fixed capacity as possible, expending exactly B units of budget. Quantity $\delta_{uv} \equiv c_{uv}/d_{uv}$ corresponds to the amount of capacity we remove for one unit of budget expended on edge (u, v). An optimal attack strategy on the cut is as follows: sort all edges $(u, v) \in E_c$ by δ_{uv}. If there is insufficient budget to remove the first edge then expend all the budget partially cutting this edge. Otherwise, remove the edge and continue to the next edge with the remaining budget. At most one edge is partially cut.

This observation leads to a simple algorithm for network inhibition: enumerate all cuts, compute the result of a greedy attack, and select the best one. There are 2^n ways to partition the nodes of an n-node graph into two sets. Not all of these partitions give minimal cuts (cuts where no proper subset of edges is also a cut). Unfortunately, there are still exponentially many minimal cuts in general, so this algorithm is not likely to be efficient, even using the algorithm by Tsukiyama et. al. [40] for enumerating all minimal cutsets of a graph in linear time per cutset.

We now examine the sensitivity of optimal inhibition strategies to changes in budget. For a particular cut, if we plot the optimal residual capacity as a function of budget (found greedily), we get a piecewise-linear convex function (nondecreasing slopes). Each piece of this *cut function* corresponds to a budget range where an edge (u, v) is partially cut and thus has slope $-\delta_{uv}$. Edges of the graph in figure 2 are labeled by capacities with destruction costs in parentheses. Two cut functions are plotted below the graph with each piece labeled by the corresponding edge.

If we plot the cut functions for all minimal cuts on the same set of axes, the lower envelope gives the optimal residual capacity as a function of budget. As illustrated in figure 2, two cut functions can cross each other multiple times. In fact the lower envelope can have exponentially many pieces. We can view this lower envelope as a function from budget to cut. For example, for just the two cuts shown in figure 2, the solid cut is lowest for budget 15. We achieve the lower-envelope capacity by cutting edges appearing in the cut function before budget 15 (edges $(s, v_1), (s, v_3)$, and part of (s, v_2)). If we increase the budget to 17, the dashed cut becomes optimal. We attack edges of the dashed cut targeted with the first 17 units of budget, namely $(v_2, v_5), (v_4, t), (s, v_3)$, and part of (v_4, v_3). Thus a small change in budget can dramatically change the optimal strategy (though the optimal residual capacity is a continuous function). Given this extreme sensitivity, the lower bound results in the next section are not surprising.

3 NP-COMPLETENESS RESULTS

In this section we state our NP-completeness results for the network inhibition problem.

Theorem 1 *The network inhibition problem is strongly NP-complete.*

Proof. Reduction from graph bisection. Omitted in this abstract.

Network inhibition is strongly NP-complete even for graphs of degree at most 3.

Theorem 2 *Inhibition of series-parallel networks is weakly NP-complete.*

Proof. (due to Steve Vavasis of Cornell) Reduction from subset sum. Omitted in this abstract.

Theorem 3 *Network inhibition is weakly NP-complete for grid graphs, bandwidth-3 graphs, and Halin graphs.*

Proof. Modifications to the proof of theorem 2. Omitted in this abstract.

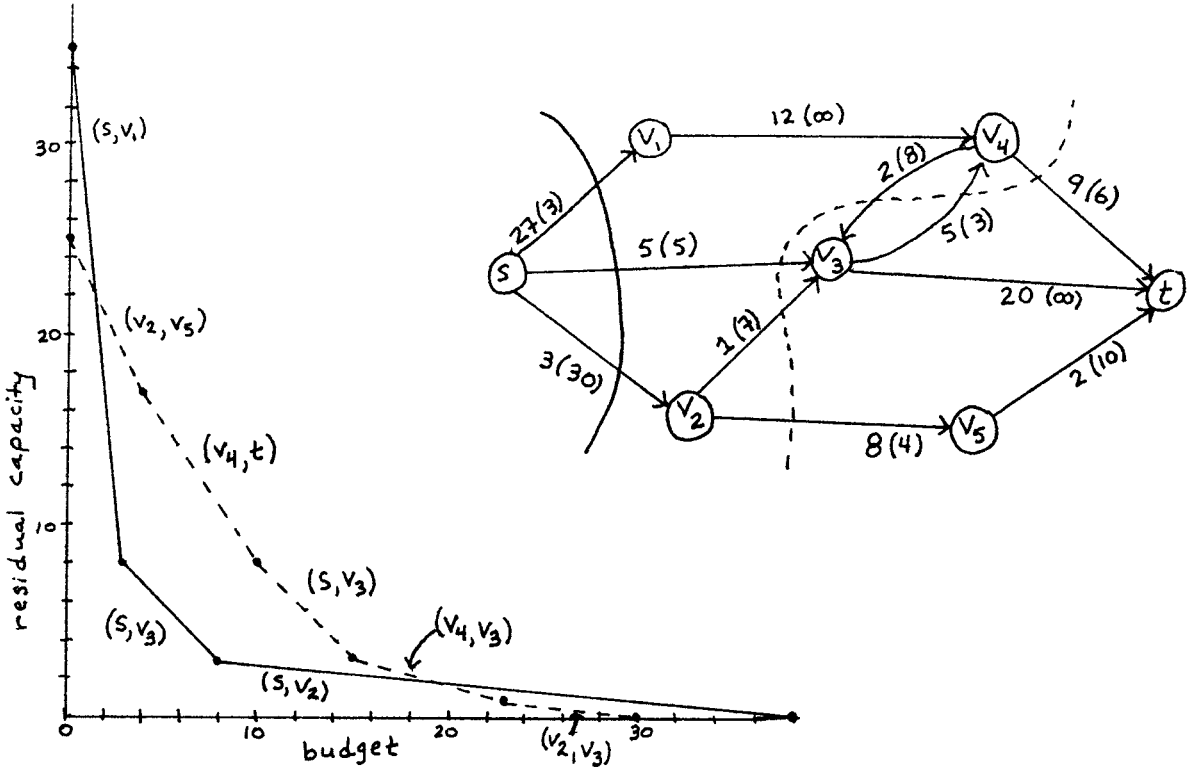

Figure 2: For a fixed cut, a greedy attack strategy is optimal. Cut functions (residual capacity as a function of budget) for the cuts indicated with dashed and solid lines are plotted below the graph. Labels on the cut functions indicate the edge partially cut for budgets in the given range.

4 PSEUDOPOLYNOMIAL-TIME ALGORITHMS FOR PLANAR GRAPHS

In this section we present an $O(n^3)$ algorithm for attacking outerplanar networks and then present and analyze three pseudopolynomial-time algorithms for inhibition of undirected planar networks (section 6 describes extensions to planar multigraphs and directed planar graphs). We characterize an optimal cycle in the dual (which correspond to cuts in the primal) and give three dynamic programming algorithms for finding such a cycle. The first, based upon a Floyd-Warshall approach, runs in time $O(n^3C)$, where n is the number of nodes in the graph and C is the optimal residual capacity. The second algorithm, based upon Dijkstra's algorithm, runs in time $O(n^2C \lg(nC))$. The third, based upon Dijkstra's algorithm and searching in Monge arrays, runs in time $O(nkC \lg(nC))$ time, where k is the minimum number of nodes on a path in the dual from a node representing a face adjacent to source s to a node representing a face adjacent to sink t. For example, if nodes s and t share a face, then we have $k = 1$. The residual capacity C in any of the above running times can be replaced by the budget B by reversing the roles of budget and capacity in the dynamic programming. The third algorithm is asymptotically superior to the first if either the optimal residual capacity or the budget is at most $2^{O(n^2/k)}$.

We can compute an optimal fixed-budget attack strategy for n-node outerplanar graphs in time $O(n^3)$. The key observation is that outerplanar graphs have $O(n^2)$ minimal s-t cuts, which are easily generated because of special structure.

Given a planar graph $G = (V, E)$, the *dual* graph $G_D = (F, E')$ is constructed by placing a node in each face and connecting two nodes by edge e' if their faces share an edge e (edges e and e' are *dual edges*). The dual is planar, possibly a multigraph, with $O(n)$ nodes and edges. There is a one-to-one correspondence between minimal s-t cutsets in a planar graph and simple cycles in the dual that separate s from t. See [21, 31] for more details.

We now describe how to identify cycles in the dual that separate the source s from the sink t. Let P be any path in graph G from vertex s to vertex t. By the Jordan curve theorem, a cycle in the dual separates vertices s and t if and only if it crosses path P an odd number of times, or equivalently if and only if it contains an odd number of edges from the set P' of the dual edges of path P. The parity of an edge $e' \in E'$ is 1 if $e' \in P'$ and 0 otherwise. Define the parity of a path or cycle to be the exclusive OR of the parities of the edges it contains. A cycle separates vertices s and t if and only if it has odd parity. Other algorithms for finding min cuts in planar graphs using cycles in the dual ([24, 38, 23, 11, 12]) do not compute the parity of cycles. They observe that a separating cycle crosses a minimum

capacity path from s to t exactly once. Because path length depends upon both budget and capacity in the network inhibition problem, the observation does not hold for us and we must compute parity explicitly.

The structure of an optimal attack allows solution by dynamic programming. Given a planar graph with capacity c_{uv} and destruction cost d_{uv} for each edge (u, v), we form the dual G' with each edge assigned the capacity and destruction cost of its dual edge. From section 2 the best attack is a greedy attack upon a simple cycle, and therefore at most one edge is partially cut. A best attack always expends exactly the budget B. If the edge that is "partially cut" is indestructible, inspecting the attack strategy yields a smaller budget which causes the same amount of damage. We refer to an attack upon a set of edges as *all/none* if no edge is partially cut. If the best attack expends exactly $0 \leq b \leq d_{uv}$ units of budget partially cutting edge (u, v), then it is completed by an all/none attack expending exactly the remaining budget $B - b$ on the best (lowest residual capacity) path from vertex v back to vertex u of parity $\overline{p_{uv}}$. To avoid the case where the best such path from node u to node v is edge (u, v) itself, we require the all/none path to have at least 2 edges. The minimum of all such cycles taken over all legal values of budget b and all edges (u, v) is the optimal solution. More formally, if $c(u, v, b)$ is the residual capacity of edge (u, v) after spending budget b on the edge, and $C(u, v, b)$ is the minimum residual capacity of a parity-$\overline{p_{uv}}$, length-at-least-2 path from vertex u to vertex v expending budget b and allowing no partial cuts, then the optimal residual capacity c_r is

$$c_r = \min_{(u,v)\in E'} \{ \min_{0\leq b\leq d_{uv}} \{c(u, v, b) + C(v, u, B - b)\}\}.$$

The first two algorithms find such a best cycle by first building a table T of all-pairs shortest paths with no partial cuts. Table entry $T(u, v, c, p)$ contains the minimum budget required to reduce the residual capacity of a u-to-v path of parity p to exactly $0 \leq c \leq C$. Given this $O(n^2C)$-size table, we can find the optimal cycle in time $O(nC)$, by examining $O(C)$ table entries for each of $O(n)$ edges. For edge (v, u) having parity p_{vu}, we consider completing a cycle with each of the paths represented in table entries $T(u, v, c, \overline{p_{vu}})$ for all $0 \leq c \leq C$. If the budget used to achieve a table entry exceeds budget B, the resulting cycle is illegal. Otherwise, edge (v, u) is cut with the remaining budget $B - T(u, v, c, \overline{p_{vu}})$. If the table size is too small, then all cycles are illegal. Thus we can use repeated doubling to choose the table size parameter C to be at most 2 times the optimal residual capacity, without *a priori* knowledge of the optimal value, and without asymptotic penalty in running time.

We now decribe how to build the table T. The first method is based on Dijkstra's single-source-shortest-paths algorithm. Given a graph $G_D = (F, E')$, with capacity c_{uv} and destruction cost d_{uv} for each edge $(u, v) \in E'$, we construct a *search graph* $G'_D = (F', A)$ as follows. Node set $F' = \{(f, c, p) : f \in F, 0 \leq c \leq C, p \in \{0, 1\}\}$. The shortest path from a start node $(r, 0, 0)$ to node $(f, c, p) \in F'$ corresponds to the minimum budget, parity-p path in the dual from node r to node f with residual capacity exactly c and no partial cuts. The edge set $A = A_{\text{cut}} \cup A_{\text{leave}}$. $A_{\text{cut}} = \{((u, c, p), (v, c, p \oplus p_{uv})) : (u, v) \in E'\}$, where the symbol $\oplus$ represents the exclusive or bit operation. Each such edge has length d_{uv}. $A_{\text{leave}} = \{((u, c_i, p), (v, c_j, p \oplus p_{uv})) : (u, v) \in E' \text{ and } c_i + c_{uv} = c_j \leq C\}$. Each such edge has length 0. Path $P = \{(r, 0, 0), (u_1, c_1, p_1), \ldots, (u_k, c_k, p_k)\}$ in the search graph corresponds to path $\{r, u_1, \ldots, u_k\}$ in the dual. The residual capacity is c_k and the budget is the length of the path. Traversing an edge in set A_{cut} corresponds to cutting the edge (capacity same, budget increases). Traversing an edge in set A_{leave} corresponds to not cutting the edge (capacity increases, budget same).

To build table T, we run a modified Dijkstra's algorithm from node $(f, 0, 0)$ for each $f \in F'$. The modifications assure that table entry $T(u, v, c, p)$ does not correspond to the length-1 path (u, v). At the start of the algorithm, when node $(f, 0, 0)$ is removed from the heap, we perform a normal decrease-key on all its neighbors, but we mark each neighbor to indicate that the current budget entry corresponds to a length-1 path. Whenever a marked node is removed from the heap, we perform decrease-key on its neighbors as usual, but instead of considering the node completed, we return it to the heap (unmarked) with a budget of ∞. The next time the node is removed, the budget will correspond to the best path containing at least 2 edges. Since each node is removed from the queue at most twice, there is no penalty in running time for this modification.

The search graph has $O(nC)$ nodes and $O(nC)$ edges. The cost for each search using Dijkstra's algorithm is $O(nC \lg(nC))$ for each of $O(n)$ searches, for a total cost of $O(n^2C \lg(nC))$. The search graph is not planar in general, but if it coincidentally stays planar, these bounds can be improved using the techniques of [11]. An alternative method for building table T, based upon the Floyd-Warshall algorithm, runs in time $O(n^3C)$.

The final algorithm for computing an optimal cycle uses search techniques for Monge arrays to avoid computing table T. Let $P = \{f_1, \ldots, f_k\}$ be a path in the dual, where f_1 is a face adjacent to source s, f_k is a face adjacent to sink t, and k is minimum ($k = 1$ if s and t share a face). Itai and Shiloach [24] observe that any cycle in the dual that separates s from t must include a node from path P. Thus it suffices to compute the best cycle passing through each f_i.

We can compute the optimal cycle passing through a specified vertex f in time $O(nC \lg(nC))$ using Dijkstra's algorithm and searches in Monge arrays. First we use Dijkstra's algorithm as described above to find the shortest (cheapest) path from node f to all other nodes for all values of residual capacity c and both parities using no partial cuts. If the optimal cycle through node

f partially cuts edge (u, v) with budget b, then the rest of the cycle consists of a path from node f to node u and a path from node f to node v of the appropriate parity using a total budget of $B - b$. To form a cycle, neither of these paths can contain edge (u, v). Because there are $\theta(C)$ paths from node f to each other node, a naive algorithm for finding an optimal cycle that combined two such paths requires $\theta(C^2)$ time per edge. We exploit special structure to compute the optimal cycle for an edge in time $O(C)$.

A matrix A is *Monge* if for every 2×2 submatrix representing rows $i < i'$ and columns $j < j'$ we have $A[i, j'] + A[i', j] \geq A[i, j] + A[i', j']$. An array is *inverse Monge* if for each such submatrix the inequality is reversed. Aggarwal et. al. [2] give an $O(m+n)$-time algorithm for computing all the column or row minima of an $m \times n$ (inverse-)Monge array[2]. We construct a $C \times C$ inverse-Monge array whose minimum entry overall corresponds to the cycle with minimum residual capacity. We then compute all row minima in time $O(C)$ and the overall minimum in time $O(C)$.

For a fixed node f and a fixed edge (u, v), consider the shortest path from $(f, 0, 0)$ to (u, c, p) for some capacity c and parity p. If this path is $[(f, 0, 0), (v_1, c_1, p_1), \ldots, (v_k, c_k, p_k), (u, c, p)]$, then the path cannot be used to form cycles with edge (v_k, u) unless $v_k = f$ (the path is a single edge).

Fix some parity p. Let $l_u^{(1)}[i]$ be the regular shortest-paths results from node f (the minimum budget for a path from node f to node u with residual capacity exactly i for $0 \leq i \leq C$). If $l_u^{(1)}[i]$ corresponds to a path with final edge (w, u), then let $l_u^{(2)}[i]$ correspond to the minimum budget for a path from f to u which doesn't use edge (w, u) and has residual capacity i. We can compute both arrays in the asymptotic time required to run Dijkstra's algorithm once. When a node is removed from the heap for the first time, the budget gives the entry for array $l_u^{(1)}[i]$. If the last edge is (w, u) for $w \neq f$, we record w, mark the node, and return it to the heap with budget reverted to its value before the last update (we must keep two previous budgets, but only the current one is used as the heap key). Each node (u, c, p) has at most two edges corresponding to any neighbor w in the original dual graph, namely $(w, c, p \oplus p_{wu})$ of length d_{wu} and $(w, c - c_{wu}, p \oplus p_{wu})$ of length 0. Therefore, after at most one more visit, we will encounter the node from a new vertex. The budget at this time, when the marked node is removed from the heap, gives the entry for array $l_u^{(2)}[i]$.

For a given edge (u, v), let $l_u[i]$ correspond to the minimum budget path from f to u of capacity i suitable for completing a cycle with edge (u, v). This array can be formed in time $O(C)$ by checking the next-to-last node on the shortest path and using the entry in either array $l_u^{(1)}[i]$ or $l_u^{(2)}[i]$. It is important for Mongité that array $l_u[i]$ be nonincreasing. "Spikes" in the array correspond to nonoptimal paths, since an earlier entry dominates in both budget and capacity. Define a new array $b_u[i] = min_{0 \leq j \leq i} l_u[j]$, which corresponds to the minimum budget for a path from node f to node u with capacity *at most* i. Define arrays $l_v[i]$ and $b_v[i]$ similarly[3].

Define array A as follows: $A[i, j] = i + j + c_{uv} - \delta_{uv}(B - b_u[i] - b_v[j])$ if $b_u[i] + b_v[j] \leq B$, and $A[i, j] = +\infty$ otherwise. Recall $\delta_{uv} \equiv c_{uv}/d_{uv}$ is the capacity removed per dollar spent on edge (u, v). Entry $A[i, j]$ is the residual capacity of the following cycle: a path from node u to node f with residual capacity at most i and budget $b_u[i]$, a path from node f to node v with residual capacity at most j and budget $b_v[j]$, and edge (u, v) attacked with the remaining budget. A cycle that requires too much budget is illegal ($+\infty$). The minimum entry in array A is the minimum residual capacity of a legal cycle (in practice, we subtract the constant $c_{uv} - \delta_{uv}B$ from each entry). It is critical for our running time that the matrix A is represented implicitly since the matrix has C^2 entries.

The $+\infty$ entries in array A all appear in the upper left corner in a *staircase* pattern (a rectilinear line drawn between the finite and infinite entries looks like a series of possibly irregular stairs). That is, if an entry $A[i, j] = +\infty$, then $A[k, l] = +\infty$ for all $k \leq i$ and $l \leq j$, which follows from the nonincreasing property of the b arrays. All finite entries in array A are the sum of functions which depend only upon the column or only upon the row (*separable* functions). It is well known that a matrix whose entries are separable functions is both Monge and inverse Monge, and that adding staircase $+\infty$ values to any corner, or pair of opposite corners, of such a matrix produces a matrix that is either Monge or inverse Monge. The matrix A is inverse Monge and therefore we can use the algorithm of Aggarwal et. al. to find all row minima in time $O(C)$.

We can find the best cycle through a given vertex in time $O(nC \lg(nC))$, since the $O(nC)$ time to search a Monge matrix for each edge is dominated by the shortest paths calculation. We find such a cycle for each of the k nodes on the path P, so the total running time is $O(nkC \lg(nC))$.

We have versions of the above three algorithms which interchange the roles of capacity and budget, useful when the budget is small. The algorithm that runs in time $O(nkB \lg(nB))$ uses similar techniques as the algorithm above, but has a different argument for inverse Mongité.

We now consider multiedges in the dual. For shortest path computations, k multiedges with capacities

[2] Their result is for totally monotone arrays, but every Monge array is totally monotone

[3] There are two arrays for each of nodes u and v to account for parity The four arrays are paired according to the parity of edge (u, v) to form two arrays and the minimum of the two arrays is the overall minimum

$c_1, \ldots, c_k$ and destruction costs $d_1 \ldots d_k$, are replaced by a single edge having capacity $\min_i c_i$ and destruction cost $\min_i d_i$. Multiedges correspond to chains in the original graph. Capacity is determined by the bottleneck, and removal of any edge breaks the chain.

Constructing cycles with a partial cut of some multiedge between node u and node v is more subtle. We can simply treat each multiedge as a regular edge. This leaves the asymptotics unchanged, since there are $O(n)$ total edges including multiedges in the dual of a planar graph. Alternatively, in time $\theta(k \lg k)$ we can compute the effective residual capacity of a set of k multiedges. We then can compute cycles faster (eg. in time $O(C)$ rather than $O(kC)$ for the table-based schemes). This is better overall for large k.

5 PLANAR FULLY POLYNOMIAL-TIME APPROXIMATION SCHEMES

In this section we convert the pseudopolynomial-time algorithms for inhibition of planar networks from section 4 into fully polynomial-time approximation schemes (FPTAS). We begin by converting the pseudopolynomial-time algorithms into approximation algorithms which run faster. The main idea is to compute only one shortest path for all residual capacities in a range of values. We give a bound on the absolute error of the approximation algorithms as a function of range size and choose the range size as a function of desired error to yield a FPTAS, namely an algorithm that runs in time polynomial in n, the network size, and ϵ, a specified error parameter, and returns an answer with a relative error at most ϵ.

We now describe the approximation algorithms. We divide the interval $[0, C']$ into chunks of size r, for some integer r. Let $I_i = [ir, (i+1)r)$ for $i = 0, \ldots, \lceil C'/r \rceil$. In the approximation algorithms, we compute only one path for each interval I_i. The shortest path computed for each range is at most $2nr$ longer than the optimal shortest path in that range, where n is the number of nodes in the original graph (see below). Thus if we wish to find the optimal residual capacity C by doubling, we must allow paths of residual capacity up to $C' = C + 2nr$. After computing shortest paths for the smaller problem, we construct the cycles as before using the $O(n^2C'/r)$-size table or the $C'/r \times C'/r$ Monge array. The running time of each approximation algorithm is the same as the pseudopolynomial algorithm it is based upon except that table size C is replaced by $\lceil C'/r \rceil = \lceil C/r \rceil + n$. Computing shortest paths using ranges instead of exact values requires a few technical modifications to the procedures described in section 4. For example, the search graph must be built as the search proceeds.

The approximation schemes have good error bounds:

Theorem 4 *The solution returned by any of the approximation schemes using range size r has residual capacity at most $2nr$ greater than the optimal residual capacity.*

Proof. Omitted in this abstract. Note: pessimistic bound in practice.

By theorem 4, a choice of range size $r = \epsilon C/4n$ converts all the approximation schemes into FPTAS. Recall that ϵ is an error parameter specified as input and C is the table size which grows by doubling until it is no more than twice the value of the optimal residual capacity. With the above choice of r, all approximation algorithms have error at most $\epsilon \cdot$ opt by theorem 4, thus meeting the error criterion of a FPTAS. The approximation scheme using Dijkstra's algorithm and searches in Monge arrays runs in time $O(nk(n + C/r) \lg(n(n + C/r)))$. Substituting $r = \epsilon C/4n$ and simplifying gives a running time of $O(n^2k(1+1/\epsilon)(\lg n + \lg(1+1/\epsilon)))$, which is a polynomial function of n and $1/\epsilon$, and therefore meets the time criterion of a FPTAS. We can convert the other approximation schemes into FPTAS using the same value of range r.

6 EXTENSIONS

In this section we give an improved FPTAS for the restricted shortest path problem, and extend our network inhibition algorithms to different models of inhibition of a single edge and to inhibition of planar multigraphs and directed planar graphs. The new models of edge inhibition are 1) no partial cuts allowed and 2) residual capacity on individual edges a piecewise-linear convex function of budget. The latter case models the situation where it is relatively easy to destroy some of an edge, but relatively hard to assure removal of the final units of capacity.

The Dijkstra-based path algorithms of section 4 can be used to compute restricted shortest paths. Given a general graph $G = (V, E)$ with transit time t_{uv} and length l_{uv} for each edge (u, v), and transit deadline T, we construct an nT-node, mT-edge search graph $G' = (V', E')$ as follows. Let $V' = \{(v, t) : v \in V, 0 \le t \le T\}$ and $E' = \{((u, t), (v, t + t_{uv})) : (u, v) \in E, 0 \le t \le T - t_{uv}\}$. Each such edge has length l_{uv}. The shortest path from node $(s, 0)$ to node (v, t) in graph G' corresponds to the shortest path from node s to node v with transit time exactly t in graph G. Running Dijkstra's algorithm from node $(s, 0)$ and choosing the shortest path to node (s', t) for $t \le T$ solves the restricted paths problem exactly in time $mT + nT \lg(nT)$. The techniques of sections 4,5 convert this algorithm into a FPTAS with running time $O(mn(1+1/\epsilon) + n^2(1+1/\epsilon)(\lg n + \lg(1+1/\epsilon)))$, the fastest FPTAS for this problem among those with running time independent of input numbers, and fastest overall for appropriate ϵ and m (see section 1).

Allowing no partial cuts does not change the complexity of network inhibition. The reductions of section 3 do not require partial cuts (or can be changed to need

none). The algorithms for planar graphs are simpler but do not improve asymptotically. Because we do not allow partial cuts we do not need to use Monge techniques to achieve time $O(nlC\lg(nC))$, where l is the minimum number of edges on a path from source s to sink t in the original planar graph.

In a real application, it is usually easier to remove the initial units of capacity upon first attacking an edge than it is to remove the final units of capacity late in the attack. Thus a more realistic model of attack upon a single edge represents residual capacity of an edge as a convex function of budget spent attacking it. Denote this function $f_{uv}(b)$. If we assume the budget expended upon any edge is integral, then any such edge-inhibition function can be modeled by at most d_{uv} linear pieces, where d_{uv} is the destruction cost of edge (u, v). Thus we assume functions f_{uv} are piecewise-linear. In practice, many fewer than d_{uv} pieces may suffice.

We can compute an optimal attack strategy for a planar network in this edge-inhibition model using the algorithms described in sections 4 and 5 on a modified graph. Replacing edge (u, v) in the original planar graph by k multiedges can simulate a convex inhibition function f_{uv} with k linear pieces. If the linear pieces of f_{uv} have endpoints at (b_i, c_i) for $0 \leq i \leq k+1$, then multiedge $(u, v)_i$ has capacity $c_i - c_{i-1}$ and destruction cost $b_i - b_{i-1}$. Since the function f_{uv} is convex, the slopes of multiedges $(u, v)_i$ increase monotonically. By the arguments in section 2, the budget is spent upon the multiedges in order.

We pay a penalty in running time which depends upon the total number of linear pieces in the functions representing edge inhibition. The dual of a planar multigraph with n nodes and e edges has $\theta(e)$ nodes. Each occurrence of n in the running time of our algorithms is replaced by e. If each function piece is part of the input, then these new running times are still functions only of input size and the value of the optimal solution. Thus the FPTAS are unchanged. If these functions are implicit functions of budget with, for example, d_{uv} pieces, the pseudopolynomial-time algorithms cannot be converted to FPTAS using the techniques of section 5.

An optimal inhibition strategy for a *directed* planar network $G = (V, E)$ can be computed in time $O(nlC\lg(nC))$, where l is the minimum number of edges on a path from source s to sink t, and C is the value of the optimal residual capacity. Because some edges in a cut now go backward across the cut (and therefore do not count), the table-based algorithms do not have obvious extensions, and the Monge-based scheme has several modifications.

7 CONCLUSIONS

In this section we present some open modeling questions and offer some concluding remarks.

Combining node with edge inhibition and modeling bridges offer challenges. In a directed graph, one can model node inhibition using edge inhibition only, but this transformation destroys planarity. In a general graph, a physical bridge is modeled by crossing edges. Attacking a bridge inhibits the transport capability of the edge running underneath it at no extra cost.

Another extension is attacking networks where the value of a flow is a linear function of flows into multiple sinks. For example, lexicographic flow can be computed in polynomial time [32, 13], but it is not clear how to minimize the maximum lexicographic flow.

Because network inhibition is strongly NP-complete, we seek a good approximation algorithms, perhaps even a PTAS. We have developed an exponential-time heuristic for general graphs based upon a relaxation of a mixed-integer program and specialized branch-and-bound[35]. The relaxation gives the convex hull of the lower envelope of cut functions described in section 2. The heuristic can be used as an approximation scheme, since it provides a lower bound and constructive upper bound at each step. We can approximate to within a constant factor if we are willing to relax bounds on both residual capacity and budget[39].

ACKNOWLEDGEMENTS

Thanks to Ernie Brickell (Sandia) for first suggesting that dynamic programming on cycles works in this metric, to James Park (Sandia) for his help in shortest paths issues and Monge arrays and for pointing out the relevance of this work to the general restricted shortest path problem, to David Greenberg (Sandia), James Park, Tandy Warnow (Sandia), Sorin Istrail (Sandia), and Ernie Brickell for comments on drafts of this paper, to Joel Wein (Polytechnic University) for pointing out Cunningham's paper, to Andrew Goldberg (Stanford) and Cliff Stein (MIT) for pointing me to references for lexicographic flow, to Steve Vavasis (Cornell) for the base proof of weak NP-completeness, to Gene Aronson (Sandia) for bringing me the problem, to Bill Cunningham (Waterloo) for references, to Valerie King (U. Victoria) for suggesting the environmental application, and to Wolf Bein (American Airlines Decision Technologies), and Bruce Hendrickson (Sandia) for helpful technical discussions.

REFERENCES

[1] A. Aggarwal and M. M. Klawe, "Applications of generalized matrix searching to geometric algorithms", *Discrete Applied Mathematics*, 27(1,2):3–23, May, 1990.

[2] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber, "Geometric applications of a matrix-searching algorithm", *Algorithmica*, 2(2):195–208, 1987.

[3] A. Aggarwal and J. K. Park, "Improved Algorithms for Economic Lot-Size Problems",*Oper. Res.*, to appear, (Report RC 15626, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1990).

[4] M. O. Ball and J. S. Provan, "Calculating bounds on reachability and connectedness in stochastic networks", *Networks*, 13:253–278, 1983.

[5] D. Bienstock, "Some generalized max-flow min-cut problems in the plane," *Mathematics of Operations Research*, 16(2):310–333, May 1991.

[6] P. Z. Chinn, J. Chvátalová, A. K. Dewdney, N.E. Gibbs, "The bandwidth problem for graphs and matrices — a survey", *J. Graph Theory*, Vol. 6, 1982, pp. 223–254.

[7] J. Chvátalová, A. K. Dewdney, N. E. Gibbs, R. R. Korfhage, "The bandwidth problem for graphs: a collection of recent results", Res. report #24, Dept. of Comput. Sci., UWO, London, Ont, 1975.

[8] J. Cheriyan, T. Hagerup, and K. Mehlhorn, "Can a maximum flow be computed in $o(mn)$ time?", Proceedings of ICALP, 1990.

[9] W. Cunningham, "Optimal attack and reinforcement of a network", *JACM*, 32(3):549–561, July 1985.

[10] L. R. Ford Jr. and D. R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, NJ, 1962.

[11] G. Fredrickson, "Fast algorithms for shortest paths in planar graphs, with applications", *SIAM J. Comput.*, 16(6):1004–1022, Dec. 1987.

[12] G. Fredrickson, "Planar graph decomposition and all pairs shortest paths", *JACM*, 38(1):162–204, 1991.

[13] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, "A fast parametric maximum flow algorithm and applications," *SIAM J. Comput.* 18(1):30–55, Feb. 1989.

[14] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems", *Theoretical Computer Science*, 1:237–267, 1976.

[15] M. R. Garey, and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, NY, 1979.

[16] A. Goldberg, R. Tarjan, "A new approach to the maximum flow problem", *JACM*, 35:921–940, 1988.

[17] O. Goldschmidt and D. Hochbaum, "Polynomial algorithm for the k-cut problem", *Proc. 29th FOCS*, 1988, pp. 444–451.

[18] D. Gusfield, "Computing the strength of a graph," *SIAM J. Comput.*, 20(4):639–654, Aug. 1991.

[19] F. Hadlock, "Finding a maximum cut of a planar graph in polynomial time", *SIAM J. Comput.*, 4(3):221–225, Sept. 1975.

[20] P. L. Hammer, R. Maffray, and M. Queyranne, "Cut-threshold graphs", *Discrete Applied Mathematics*, 30:163–179, 1991.

[21] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1972.

[22] R. Hassin, "Approximation schemes for the restricted shortest path problem", *Math. Oper. Res.*, 17(1):36–42, Feb. 1992.

[23] R. Hassin and D. Johnson, "An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks", *SIAM J. Comput.*, 14(3):612–624, Aug. 1985.

[24] A. Itai and Y. Shiloach, "Maximum flow in planar networks", *SIAM J. Comput.*, 8(2):135–150, 1979.

[25] D. Johnson, "The NP-completeness column: an ongoing guide (16th)", *J. Alg.*, 6(3):434–451, 1985.

[26] H. C. Joksch, "The shortest route problem with Constraints", *J. Math. Anal. Appl.*, 14:191–197, 1966.

[27] R. Karp, "Reducibility among combinatorial problems", in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher eds., Plenum Press, NY, 1972, pp. 85–103.

[28] V. King, S. Rao, and R. Tarjan, "A faster deterministic maximum flow algorithm", *Proc. 3rd ACM-SIAM SODA*, 1992, pp. 157–164.

[29] M. M. Klawe and D. J. Kleitman, "An almost linear time algorithm for generalized matrix searching" *SIAM J. Disc. Math.*, 3(1):81–97, Feb. 1990.

[30] P.N. Klein, A. Agrawal, R. Ravi, and S. Rao, "Approximation through multicommodity flow," *Proc. 31st Annual FOCS*, 1990, pp. 726–737.

[31] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, 1976.

[32] E. Minieka, "Maximal, lexicographic, and dynamic network flows", *Oper. Res.*, 21(2):517–527, 1973.

[33] J. K. Park and C. A. Phillips, manuscript in preparation.

[34] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc, Englewood Cliffs, NJ, 1982.

[35] C. A. Phillips, manuscript in preparation.

[36] F. Preparata and M. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, NY, 1985.

[37] J. S. Provan and M. O. Ball, "The complexity of counting cuts and of computing the probability that a graph is connected", *SIAM J.Comput.*, 12(4):777–788, Nov. 1983.

[38] J. Reif, "Minimum s-t cut of a planar undirected network in $O(n \log^2(n))$ time", *SIAM J. Comput.*, 12(1):71–81, Feb. 1983.

[39] Éva Tardos, priviate communication.

[40] S. Tsukiyama, I. Shirakawa, and H. Ozaki, "An algorithm to enumerate all cutsets of a graph in linear time per cutset", *JACM*, 27(4):619–632, Oct. 1980.