NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN ALGORITHM FOR ENUMERATING THE NEAR-MINIMUM WEIGHT *S-T* CUTS OF A GRAPH

by

Ahmet Balcioglu

December 2000

Thesis Co-Advisors:	R. Kevin Wood Craig W. Rasmussen
Second Reader:	Gerald G. Brown

Approved for public release; distribution is unlimited.

20010307 140

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE December 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE: An Algorithm for Enumerating The Near-Minimum Weight *s-t* Cuts of a Graph	5. FUNDING NUMBERS
6. AUTHOR(S) Balcioglu, Ahmet	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (***maximum 200 words***)

We provide an algorithm for enumerating near-minimum weight s-t cuts in directed and undirected graphs, with applications to network interdiction and network reliability. "Near-minimum" means within a factor of $1+\varepsilon$ of the minimum for some $\varepsilon \geq 0$. The algorithm is based on recursive inclusion and exclusion of edges in locally minimum-weight cuts identified with a maximum flow algorithm. We prove a polynomial-time complexity result when $\varepsilon = 0$, and for $\varepsilon > 0$ we demonstrate good empirical efficiency. The algorithm is programmed in Java, run on a 733 MHz Pentium III computer with 128 megabytes of memory, and tested on a number of graphs. For example, all 274,550 near-minimum cuts within 10% of the minimum weight can be obtained in 74 seconds for a 627 vertex 2,450 edge unweighted graph. All 20,806 near-minimum cuts within 20% of minimum can be enumerated in 61 seconds on the same graph with weights being uniformly distributed integers in the range [1,10].

14. SUBJECT TERMS Near-Minimum Cuts, Cut Enumeration, Minimum Cuts, Network Interdiction	15. NUMBER OF PAGES 64
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

AN ALGORITHM FOR ENUMERATING THE NEAR-MINIMUM WEIGHT *S-T* CUTS OF A GRAPH

Ahmet Balcioglu
First Lieutenant, Turkish Army
B.S., Turkish Army Military Academy, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
December 2000

Author: _____
Ahmet Balcioglu

Approved by: _____
R. Kevin Wood, Thesis Co-Advisor

_____
Craig W. Rasmussen, Thesis Co-Advisor

_____
Gerald G. Brown, Second Reader

_____
James N. Eagle, Chairman
Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

We provide an algorithm for enumerating near-minimum weight s-t cuts in directed and undirected graphs, with applications to network interdiction and network reliability. "Near-minimum" means within a factor of $1+\varepsilon$ of the minimum for some $\varepsilon \geq 0$. The algorithm is based on recursive inclusion and exclusion of edges in locally minimum-weight cuts identified with a maximum flow algorithm. We prove a polynomial-time complexity result when $\varepsilon = 0$, and for $\varepsilon > 0$ we demonstrate good empirical efficiency. The algorithm is programmed in Java, run on a 733 MHz Pentium III computer with 128 megabytes of memory, and tested on a number of graphs. For example, all 274,550 near-minimum cuts within 10% of the minimum weight can be obtained in 74 seconds for a 627 vertex 2,450 edge unweighted graph. All 20,806 near-minimum cuts within 20% of minimum can be enumerated in 61 seconds on the same graph with weights being uniformly distributed integers in the range [1,10].

THIS PAGE INTENTIONALLY LEFT BL ANK

TABLE OF CONTENTS

I. INTRODUCTION 1

A. NEAR-MINIMUM CUTS AND THEIR APPLICATIONS 1

B. BACKGROUND 4

C. THESIS OUTLINE 8

II. DEFINITIONS AND NOTATION 9

III. THEORETICAL RESULTS 11

A. BASIC ALGORITHMS 11

B. AN IMPLEMENTABLE ALGORITHM 16

C. CORRECTNESS OF THE ALGORITHM 19

D. COMPLEXITY OF THE ALGORITHM 21

1. Complexity Analysis of Minimum-Cut Enumeration 21

2. Complexity Analysis of Near-Minimum Cut Enumeration 23

IV. COMPUTATIONAL RESULTS 25

A. IMPLEMENTATION DETAILS 25

1. Efficient Implementation of Algorithm B 25

2. Problem Generators 27

B. THE EXPERIMENTS 30

1. Experiments on Unweighted Graphs 30

2. Experiments on Weighted Graphs 33

V. CONCLUSIONS AND RECOMMEDATIONS 37

LIST OF REFERENCES 41

INITIAL DISTRIBUTION LIST 45

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. Algorithm A1: A generic partitioning algorithm for enumerating all near-minimun *s-t* cuts 12

Figure 2. Sample Graph. 13

Figure 3. Enumeration Tree for the Graph of Figure 2 14

Figure 4. Algorithm A2: A "relaxed" version of Algorithm A1. 15

Figure 5. Algorithm B: An approximate implementation of Algorithm A2 17

Figure 6. Quasi-inclusion and -exclusion of an edge from a cut. 18

Figure 7. An unweighted directed grid graph generated by GGFGEN 28

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Problem groups for GGF.. 28

Table 2. Problem types for DBLCYCLEGEN and AD.. 29

Table 3. Run times (in seconds) for Algorihtm B solving AMCP-st on unweighted instances of GGF-square and GGF-long graphs 31

Table 4. Run times (in seconds) for ANMCP-st solutions of Algorithm B on unweighted GGF-square instances with $\varepsilon = 0.05, 0.10, 0.15$ 32

Table 5. Computational results on unweighted, acyclic dense (AD) graphs with various threshold levels ε 33

Table 6. Computational results for minimum-cut enumeration (AMCP-st) on weighted, GGF-square problems 33

Table 7. Computational results for near-minimum cut enumeration (ANMCP-st) on weighted GGF-square problems 34

Table 8. Computational results for near-minimum cut enumeration (ANMCP-st) on weighted DBLCYC-I problems.. 35

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

We consider the problem of enumerating all near-minimum weight *s-t* cuts in a directed graph $G=(V,E)$. A minimal *s-t* cut is a minimal set of edges whose removal results in disconnection of the "source vertex" *s* and "sink vertex" *t*. A cut is "near minimum" if its weight is within a multiplicative factor of $1+\varepsilon$ times the minimum cut weight for some $\varepsilon \geq 0$.

This thesis is motivated by network interdiction problems in which a decision maker selects edges to interdict based on a primary criterion, e.g., resource consumption, and some secondary criteria, e.g., collateral damage, exposure to enemy fire, etc. However, the results may also be useful for evaluating graph reliability and other problems.

Our enumeration algorithm is based on a recursive "inclusion-exclusion" method. The algorithm identifies a (minimal) minimum-weight *s-t* cut by exploiting the duality between max flow and minimum cut, and then systematically partitions the space of minimal (and possibly some non-minimal) cuts by attempting to include and exclude the edges in that cut. The *quasi-exclusion* of an edge is accomplished by setting its weight to infinity, so that the edge will not be identified as a part of any finite-weight cut in subsequent solutions. We *quasi-include* an edge (u,v) by implicitly adding the infinite weight edges (s,u) and (v,t) to the graph so that every finite-weight cut in G must now contain (u,v). The weight of the locally minimum cut is monotonically non-decreasing, so the recursive algorithm can backtrack as soon as the weight of a locally minimum cut exceeds $(1+\varepsilon)W_{MIN}$, where W_{MIN} is the minimum weight of a cut in G.

The quasi-inclusion technique can, unfortunately, lead to enumeration of non-minimal cuts, i.e., cuts that contain more edges than necessary to disconnect s from t. We identify non-minimal cuts by performing two searches beginning from s and t and not traversing cut edges: If any endpoint of a cut edge is not reached, then the cut is non-minimal. The algorithm simply ignores non-minimal cuts and continues.

The algorithm is correct because the sequence of weights of (locally) minimum cuts is monotonic non-decreasing as we descend along any path in the inclusion-exclusion enumeration tree and because we never eliminate or duplicate any minimal, near-minimum cut during enumeration.

The complexity of the algorithm for finding only minimum cuts (when $\varepsilon = 0$) is $O(f(|V|,|E|)+|V||E||\boldsymbol{C}_0(G)|)$ where $f(|V|,|E|)$ is the complexity of solving a maximum flow problem on $G = (V,E)$ from scratch and $\boldsymbol{C}_0(G)$ is the set of minimum cuts in G. Unfortunately, this polynomial complexity result is not valid for near-minimum cut enumeration, because we cannot properly bound the number of non-minimal cuts identified during the enumeration. Thus the worst-case complexity of the algorithm for near-minimum cut enumeration remains unknown when $\varepsilon > 0$.

The enumeration algorithm is implemented in Java (JDK 1.2.2) and tested using a 733 MHz Pentium III computer with 128 megabytes of RAM operating under Microsoft Windows 98 SE. For example, in a 25 by 25 grid graph with 627 vertices, 2,450 edges and unit weights, all 24 minimum-weight cuts ($\varepsilon = 0$) are enumerated in 0.22 seconds, all 1,128 cuts with $\varepsilon = 0.05$ are enumerated in 3.91 seconds, and all 3,621,978 cuts with $\varepsilon = 0.15$ are enumerated in 973.29 seconds.

ACKNOWLEDGMENTS

I would like to acknowledge my advisors Prof. R. Kevin Wood and Prof. Craig W. Rasmussen for their counsel and patience throughout the research and writing of this thesis. Their wisdom and the guidance significantly enhanced my education at the Naval Postgraduate School.

I would also like to acknowledge Prof. Gerald G. Brown, Prof. Richard Rosenthal, Prof. Gordon H. Bradley, and the rest of the of the Operations Research lecturers for the inspirational classes that helped us all to think beyond the state of the art.

Finally, I would like to thank my wife Hacer and my daughter Ecem for the love and meaning they have brought to my life. I could not have made it through school without their understanding and perseverance.

Excerpt from Walden
I went to the woods because I wished to live deliberately, to front only the essential facts of life, and see if I could not learn what it had to teach, and not, when I came to die, discover that I had not lived. I did not wish to live what was not life, living is so dear, nor did I wish to practice resignation, unless it was quite necessary. I wanted to live deep and suck all the marrow of life, to live so sturdily and Spartan-like as to put to rout all that was not life, to cut a broad swath and shave close, to drive life into a corner, and reduce it to its lowest terms, and if it proved to be mean, why then to get the whole and genuine meanness of it, and publish its meanness to the world; or if it were sublime, to know it by experience, and be able to give a true account of it in my next excursion. For most men, it appears to me, are in a strange uncertainty about it, whether it is of the devil or of God, and have somewhat hastily concluded that it is the chief end of man here to "glorify God and enjoy him forever."

- Henry David Thoreau

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Enumeration of various types of minimal cuts in graphs has been well studied, and both the operations research and reliability engineering communities have presented efficient enumeration algorithms. In this thesis, we deal with a particular type of cut that has not received the same attention as other types, specifically, near-minimum-weight minimal *s-t* cuts.

We develop and implement an empirically efficient algorithm to enumerate all near-minimum-weight minimal *s-t* cuts in a directed or undirected graph. (Note: "cuts" will be used synonymously with "minimal cuts" from here on unless stated otherwise. Also, note that "minimal" refers to the set-theoretic property that a set of edges that disconnects all *s-t* paths is minimal, and "minimum" refers to the total weight of a cut.)

A. NEAR-MINIMUM CUTS AND THEIR APPLICATIONS

The network to be considered, primarily, is a *directed graph* $G = (V, E)$, with positive edge weights and two special, distinct vertices, a source s and a sink t. An *s-t cut* is a minimal set of edges whose removal breaks all directed paths from s to t. We define the problem of finding an *s-t* cut of minimum weight among all possible *s-t* cuts in G as the *minimum s-t cut problem* (MCP-st). We also define two extensions of MCP-st, the problem of *enumerating all minimum s-t cuts* in G (AMCP-st) and the problem of *enumerating all near-minimum weight s-t cuts* (ANMCP-st) whose weight is within a factor of $1+\varepsilon$ of the minimum *s-t* cut weight in the given network. The last problem is the focus of this thesis.

The analogs of AMCP-st and ANMCP-st in undirected weighted graphs G are easily solved using the techniques we develop, also. An *s-t* cut in an undirected network is simply a minimal set of edges C whose deletion breaks all (undirected) *s-t* paths. If we replace each undirected edge by two directed, anti-parallel edges, each with the weight of the original undirected edge—this is a standard transformation, of course—each *s-t* cut in the new directed graph corresponds in a one-to-one fashion with an *s-t* cut in the original graph. Thus, an efficient technique to enumerate *s-t* cuts in a directed graphs leads to an efficient technique to enumerate *s-t* cuts in an undirected graph.

Another type of cut can be defined in an undirected graph G which we call a *general cut*; this is simply any minimal disconnecting set of edges in G. The problems of finding and enumerating general cuts are somewhat related to our problems and will be discussed briefly, so we make these definitions: (a) The problem of finding a minimum-weight general cut is the *general min-cut problem* (MCP), (b) the problem of enumerating all minimum-weight general cuts is the *general all min-cuts problem* (AMCP), and (c) the problem of enumerating all near-minimum-weight general cuts is the *general all near-min-cuts problem* (ANMCP).

In the remainder of this thesis, we will use the following notation and define more as needed. For *s-t* cuts, we denote a minimum cut as C_0 and a near-minimum cut as C_ε with their respective cardinalities $|C_0|$ and $|C_\varepsilon|$. We also let $\boldsymbol{C}_0(G)$ and $\boldsymbol{C}_\varepsilon(G)$ denote the set of minimum and near-minimum cuts in G, respectively.

One application of ANMCP-st arises in network interdiction problems (Wood 1993). A network user (adversary) attempts to "communicate" between source s and sink t in a directed network while the interdictor (decision maker), using limited

resources (aerial sorties, cruise missiles, etc.), tries to break all paths between s and t. By treating the amount of resource required to destroy an edge as its weight, a decision maker can use a maximum flow minimum-cut algorithm to identify a minimum-weight cut, i.e., minimum-resource cut, to disrupt the communication between s and t. (Hereafter, following common practice, the term *max* is substituted for maximum and *min* for minimum.)

There may be secondary criteria that the decision maker wishes to consider when determining the best interdiction plan, e.g., collateral damage, risk to attacking forces, etc. In this case, near-optimal solutions with respect to the primary criterion can be obtained by solving ANMCP-st; then those solutions can be evaluated against the secondary criteria for suitability. One of those near-optimal "good solutions" might produce more desirable results than an "optimal solution" obtained by solving MCP-st or AMCP-st (Boyle 1998, Gibbons 2000.)

Another application of ANMCP-st arises in assessing the reliability and connectivity of networks (Colbourn 1987, Provan and Ball 1983). Consider a probabilistic graph G with two specified vertices s and t, in which each edge fails independently with probability p. The *s-t connectedness* (or *two-terminal reliability* in the undirected case) of the graph is the probability that there exists a working *s-t* path in G. This probability can be computed as $1-\sum_{k=k_{\min}}^{|E|} \lambda_k p^k (1-p)^{|E|-k}$, where k_{min} is the cardinality of the minimum cut and λ_k is the number of edge sets of cardinality k whose deletion disconnects s from t. When p is sufficiently small, this value can be approximated by $1-\lambda_{k\min} p^{k_{\min}} (1-p)^{|E|-k_{\min}}$, which equals $1-|\boldsymbol{C}_0(G)|\, p^{|C_0|}(1-p)^{|E|-|C_0|}$ when

edge weights are all defined to be 1 (i.e., weight of a cut corresponds to the number of edges in the cut) (Colbourn 1987, pp. 53-54). This approximation (which is also an upper bound) can be made more accurate by including terms from the original formula for $k_{\min} \le k \le k'$, where k' is "not very large." (Intuitively, only cuts with small cardinality can have a significant probability of failing, and thus disconnecting *s* and *t*; Karger 1999) When $k = k_{\min}$, λ_k counts only minimal cuts, but when $k > k_{\min}$, non-minimal cuts must be counted in addition to minimal ones. However, non-minimal cuts of cardinality k are easily counted given the minimal cuts of cardinality $k-1$, $k-2$, etc. Thus, enumeration of minimum and near-minimum cuts is essential for accurately estimating *s-t* connectedness of *G*.

B. BACKGROUND

As a result of their widespread applicability in network reliability and in many other combinatorial problems (Picard and Queyranne 1982), AMCP, AMCP-st and the problem of enumerating all minimal *s-t* cuts have been intensively studied, but ANMCP-st has not received the same attention.

One brute-force approach for ANMCP-st would be to find all minimal *s-t* cuts and then eliminate the ones that are not minimum or near-minimum. All minimal *s-t* cuts of a graph can be enumerated in time that is polynomial in the size of *G* and in the number of cuts enumerated (Tsukiyama et al., 1980, Abel and Bicker 1982, Karzanov and Timofeev 1986, Shier and Whited 1986, Sung and Yoo 1992, Ahmad 1990, Prasad et al., 1992, Nahman 1995, Patvardhan et al., 1995 and Fard and Lee 1999). Unfortunately, this cannot lead to an efficient general approach for AMCP-st or ANMCP-st because the

number of cuts in a graph may be exponential in the size of that graph while the number of minimum and near-minimum cuts may be polynomial. For instance, if G is a complete directed graph with edge weights of 1, the total number of minimal cuts is $2^{|V|-2}$, the number of minimum cuts is 2 and the number of (minimal) cuts of the next largest size is $2(|V|-2)$.

All minimum *s-t* cuts (AMCP-st) can be enumerated efficiently. Picard and Queyranne (1980) present a characterization of all minimum *s-t* cuts to solve AMCP-st by exploiting "max flow-min cut" duality. They find a maximum *s-t* flow in G, define a binary relation (Picard and Queyranne 1980, Theorem 1) on the vertices of the resulting "residual graph" and show how to find all minimum *s-t* cuts by efficiently generating all closures of this graph. Gusfield and Naor (1993) and Curet (1999) give efficient algorithms for AMCP-st using this idea. Ball and Provan (1983) identify all "*s*-directed minimum cuts" by solving AMCP-st between s and i, for each $i \in V - \{s\}$, and upon identification of the solution to AMCP-si, they collapse vertices s and i into a single vertex s to find the remaining *s*-directed cuts (which break all paths between s and $T = V - \{s\}$). Provan and Shier (1996) define a general paradigm to solve AMCP-st by extending the Picard and Queyranne approach. Their paradigm lists *s-t* cuts in $O(|V|)$ time per cut in directed and undirected graphs. They also show that the algorithms of Picard and Queyranne (1980) and Ball and Provan (1983) may not be polynomial in the case of directed graphs. Kanevsky (1993) uses Provan and Shier's paradigm to find all minimum-size "separating vertex sets" instead of edge sets.

Before going any further in discussing algorithms to solve ANMCP and ANMCP-st efficiently, we must define some notation. We let $n = |V|$ and $m = |E|$ from here on.

Ramanathan and Colbourn (1987) solve ANMCP by giving an efficient algorithm to count "almost-minimum cardinality *s-t* cuts." They bound the number of cuts enumerated by $O(m^k n^{k+2})$, where $k \geq 1$ is a constant that determines the cardinality of almost-minimum cut together with the cardinality of the minimum cut. This algorithm is applicable only to undirected graphs and polynomial complexity is guaranteed only if k is fixed.

Karger and Stein (1996) introduce a randomized algorithm for solving ANMCP by using edge contraction (identify an edge that does not cross the minimum cut and merge its endpoints into a single new vertex without losing the minimum cut). Their algorithm enumerates all general cuts whose weight is within a factor α of the minimum cut in $O(n^{2\alpha} \log^2 n)$ expected time with high probability. They also derive an upper bound $O(n^{2\alpha})$ on the number of cuts. Karger (2000) later improves this upper bound to $O(n^{\lfloor 2\alpha \rfloor})$. But this is a Monte Carlo algorithm and gives the right answer with high probability, not with certainty. Nagamochi et al. (1997) give a deterministic algorithm similar to that of Karger and Stein (1996) for solving ANMCP. They show that all general cuts of weight less than k times the general minimum cut weight can be enumerated in $O(m^2 n + n^{2k} m)$ time. Unfortunately, both algorithms apply only to undirected graphs and, furthermore, they are unlikely to be extendable to enumeration problems involving *s-t* cuts (Karger and Stein 1996).

Vazirani and Yannakakis (1992) propose a polynomial-time algorithm for solving ANMCP and ANMCP-st in a directed or undirected graph with weighted or unweighted edges. In their extended abstract, they give an algorithm to find all general cuts and all

s-t cuts in ascending order with respect to their weight. They show that the k-th minimum weight *s-t* cuts in this order can be enumerated using $O(n^{2k})$ maximum flow computations, for any fixed *k*. They list the *s-t* cuts in increasing order of the weight by using an idea similar to that of Picard and Queyranne (1980) (shrinking the strongly connected components to single vertices in the residual graph) and provide a bound of $O(n^{3k-1})$ for the number of *k*th minimum weight cuts in an undirected graph. The technique of Vazirani and Yannakakis may be useful, but it is not as easy to describe as ours and is based on an unproven assertion. In particular, they state: "Given a partially specified cut, we can find with one max flow computation a minimum weight *s-t* cut consistent with it." We leave it to others to determine the validity of their approach.

Boyle's algorithm for solving constrained network-interdiction problems on undirected graphs (Boyle 1999) can be modified to enumerate near-minimum cuts. But the algorithm is only applicable to planar graphs and, because the technique is based on the planar dual of a graph, no generalization to non-planar graphs seems likely. (A generalization to directed planar graphs might be possible.) Gibbons (2000) gives an algorithm for solving ANMCP-st in a directed and undirected graph, but he may enumerate a cut more than once. Empirically, the running time and number of cuts enumerated grow rapidly as the size of graph and ε increase.

The current approaches for solving ANMCP-st are either computationally prohibitive (Gibbons 2000), not truly efficient (Ramanathan and Colbourn 1987), or restricted to specific types of graphs, e.g., undirected (Ramanathan and Colbourn 1987), planar (Boyle 1999). There are some efficient techniques to enumerate all near-minimum general cuts, i.e., to solve ANMCP (Karger 2000, Nagamochi et al. 1997), but these do

not seem to be extendable to ANMCP-st. Only the algorithm due to Vazirani and Yannakakis (1992) may be efficient for solving ANMCP-st (and ANMCP), but their technique is complicated and based on an unproven assertion. Therefore, we propose a new algorithm to solve all instances of ANMCP-st efficiently, and provide computational results for a simple implementation of the algorithm.

Our algorithm for solving ANMCP-st first identifies a minimum weight *s-t* cut by solving a maximum flow problem. Given a minimum cut, the algorithm recursively partitions the space of possible cuts by forcing inclusion and exclusion of cut edges from subsequent cuts (which are possible solutions) to enumerate all possible near-minimum cuts in G. An edge (u,v) is excluded from a cut by setting its weight to infinity. Edge inclusion is accomplished by implicitly adding two infinite-weight edges to G. One edge connects s to u and the other connects v to t so that they create a simple path from s to t via that edge. The technique for including edges, unfortunately, can cause the algorithm to identify non-minimal cuts, i.e., cuts containing a minimal cut as a subset.

C. THESIS OUTLINE

The remainder of this thesis is organized as follows: Chapter II introduces more definitions and notation. In Chapter III, we explain our algorithm theoretically and prove its correctness. Chapter IV contains computational results from testing the algorithm on different types of graphs with different sizes. Finally, Chapter V summarizes our findings and gives conclusions.

II. DEFINITIONS AND NOTATION

Let $G = (V, E)$ be an *edge-weighted directed graph* with a finite set V of *vertices* and a set $E \subseteq V \times V$ of *edges*. A directed edge $e = (u,v)$ is an *incoming* edge to v and an *outgoing* edge from u. An *undirected graph* is defined similarly, except that its edges are unordered pairs from V (with no orientation). We may use e or (u,v) as shorthand for $e = (u,v)$. We denote the number of vertices by $n = |V|$ and the number of edges by $m = |E|$. We distinguish two separate vertices s and t in V as the *source* and *sink*, respectively. Edge weights are specified by the *weight function* w: $E \rightarrow Z^+$, where Z^+ is the set of non-negative integers. We denote the weight of an edge as w_e or $w(u,v)$ and the vector of edge weights as $\mathbf{w} = (w_{e_1}, w_{e_2}, \ldots, w_{e_m})$.

An *s-t cut* in G is a set of edges whose removal disconnects s from t in G, i.e., breaks all (directed) paths from s to t. An *s-t* cut is *minimal* if it does not contain a subset which itself is an *s-t* cut of the graph. We will use "*s-t* cut" and "cut" interchangeably for "minimal *s-t* cut" in the remainder of the thesis; a cut is denoted by C. (Note: Whenever we need to mention cuts other than *s-t* cuts, we may refer such cuts as simply *general cuts*.) The *weight* $w(C)$ of an *s-t* cut is the sum of the edge weights in the cut, that is, $w(C) = \sum_{e \in C} w_e$. A *minimum cut* (min cut) is an *s-t* cut whose weight is minimum among all *s-t* cuts; such a cut is denoted C_0. A *near-minimum cut* (near-min cut) is an *s-t* cut whose weight is at most $1 + \varepsilon$ times the minimum cut weight for a given $\varepsilon \geq 0$; such a cut

is denoted C_ε. We use $\boldsymbol{C}_0(G)$ to denote the set of minimum cuts, and $\boldsymbol{C}_\varepsilon(G)$ to denote the set of near-min cuts.

A *flow* in a directed G is a function $f : E \rightarrow Z$, where $f(e) \leq w(e)$ for each edge $e \in E$, and $\sum_{(v,u)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u)$ for all $u \in V - \{s,t\}$. The value of flow from s to t is $\mathrm{F} = \sum_{u\in V} f(s,u)$. A flow F* with the maximum value among all flows is called *maximum flow* between s and t. In the *maximum flow problem* we wish to find a flow of maximum value from s to t.

As a result of the max flow min-cut theorem and its proof (e.g., Ahuja et al pp.184-185), we know that $w(C_0) = \mathrm{F}^*$, where $C_0 \in \boldsymbol{C}_0(G)$. Given any maximum flow F* together with $f^*(e)$ for every $e \in E$, we can identify a minimum cut C_0 in $O(m)$ time.

In describing our algorithm, we use W_{MIN} and W_{NEW} to denote the weight of minimum-cut in G and the weight of a newly identified cut (which will be a min cut in a modified graph), respectively.

A rooted tree, T, is a free tree (a connected, acyclic, undirected graph) in which one node (called the "root" and denoted by r) is distinguished from others. A node i in T with root r is said to be in level (depth) l if the length of unique path $P_i = (r, v_1, v_2, .., v_{k-1}, i)$ from root r to node i is k. Every node along path P_i (except node i) is called a (proper) ancestor of i, and if i is ancestor of j, then j is descendant of i. Let h and i be the last two nodes of P_i, then h is the *parent* of i and i is a *child* of h. Nodes with the same parents are called *siblings*. A rooted tree, and *enumeration tree*, will be used to describe the recursive partitioning process that we use to solve ANMCP-st.

III. THEORETICAL RESULTS

In this chapter, we introduce our algorithm (Algorithm B) for solving ANMCP-st and AMCP-st. We first describe the generic partitioning algorithm, Algorithm A1, from Gibbons (2000) as the basic approach. This is modified into Algorithm A2, which is then approximately implemented as Algorithm B.

A. BASIC ALGORITHMS

Algorithm A1 (Figure 1) provides a basic framework for enumerating near-min *s-t* cuts although it may be difficult or impossible to implement efficiently. It begins by finding a minimum cut C_0 and its weight $w(C_0)$. Then, the algorithm calls the procedure *ENUMERATE* which attempts to find a new minimum cut by *processing* the edges of originally identified cut such that the edges are forced into (*included in*) or out of (*excluded from*) any new near-min cuts. For example, let $C_0 = \{e_1, e_2, \ldots, e_k\}$ be the edges of the initial minimum cut. Based on this cut, $C_\varepsilon(G)$ can be partitioned as

$$C_\varepsilon(G) = [C_\varepsilon(G) \cap (e_1)] \cup [C_\varepsilon(G) \cap e_1 \cap (e_2)] \cup [C_\varepsilon(G) \cap e_1 \cap e_2 \cap (e_3)] \cup$$

$$\ldots \cup [C_\varepsilon(G) \cap e_1 \cap e_2 \cap \ldots \cap e_{k'-1} \cap (e_k)] \cup [C_\varepsilon(G) \cap e_1 \cap \ldots \cap e_k],$$

where $C_\varepsilon(G) \cap e_1 \cap e_2 \cap \ldots e_{k'-1} \cap (e_{k'})$ is interpreted to mean all near-min cuts containing e_1 through $e_{k'-1}$ but not $e_{k'}$. The cuts in this partition, except for the unique cut of the last term which has already been found as C_0, are enumerated by calling *ENUMERATE* recursively with the argument sets E^+ and E^-, where E^+ denotes the set of

Algorithm A1

INPUT: A directed network $G = (V, E)$, s, t, $\mathbf{w}$ and ε.

OUTPUT: All minimal s-t cuts C such that $w(C) \leq (1 + \varepsilon) \times w(C_0)$ where C_0 is a minimum weight s-t cut of G.

```
begin
    Find a min cut C0 in G and let W_MIN ← w(C0);
    W_MAX ← (1 + ε) × W_MIN ;
    E+ ← Ø;          /* set of edges to be included */
    E− ← Ø;          /* set of edges to be excluded */
    ENUMERATE (G, s, t, w, E+ , E− , W_MAX) ;
end
```

Procedure *ENUMERATE* $(G, s, t, \mathbf{w}, E^+, E^-, W_{MAX})$

```
begin
    if (∃ a cut C′ such that E+ ⊂ C′ and E− ∩ C′ = Ø)
        then let C_ε ← C′_0 and W_NEW ← w(C′_0) where C′_0 is a min
                cut satisfying the conditions for C′ ;
        else return;
    if (W_NEW > W_MAX) then return;
    print (W_NEW, C_ε);
    for (each edge e ∈ { C_ε \ E+ })
        E− ← E− ∪ {e};
        ENUMERATE (G, s, t, w, E+ , E− , W_MAX);
        E− ← E− \ {e};
        E+ ← E+ ∪ {e};
    endfor
    return;
end
```

Figure 1. Algorithm A1: A generic partitioning algorithm for enumerating all near-minimum s-t cuts with a weight no greater than $1 + \varepsilon$ times the minimum cut weight.

edges to be included and E^- denotes the set of edges to be excluded. (This essentially results in calling *ENUMERATE* with $\{(e_1)\}$, $\{e_1, (e_2)\}$, $\{e_1, e_2, (e_3)\}$,...,$\{e_1, e_2, \ldots, (e_k)\}$ for the just-identified cut $\{e_1, e_2, \ldots, e_k\}$, where, as above, edges in parentheses are forced out of possible solutions while the others are forced in.) The procedure calls itself recursively for every "unforced edge" of the locally minimium cut that is identified within each call to *ENUMERATE*, unless it is determined that no acceptable cuts remain and the algorithm must backtrack.

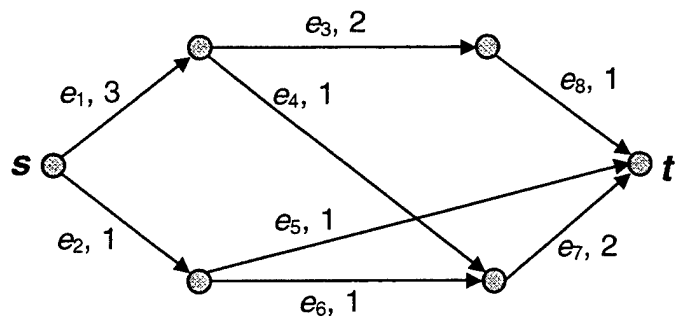


Figure 2. Sample Graph. Numbers represent edge weights.

To illustrate how the enumeration works, consider the enumeration tree (which is an instance of a rooted tree) given in Figure 3, which corresponds to solving ANMCP-st on the graph of Figure 2. The algorithm first finds a minimum cut, $\{e_2, e_4, e_8\}$ at the root node at level 0, then recursively partitions the solution space via $\{(e_2)\}$, $\{e_2, (e_4)\}$ and $\{e_2, e_4, (e_8)\}$. Once an edge of a cut at some node k has been processed, then it will never be processed again at any descendant node of k, because its status as "included" or "excluded" with respect to the current cut has been fixed at node k. The branches with $\{(e_2)\}$, $\{e_2, (e_4)\}$ and $\{e_2, e_4, (e_8)\}$ correspond to searches for a new minimum cut by

processing the edges as described. If a search is successful, it leads to a productive node where the edges of a new cut, whose status has not been fixed, are processed. Otherwise, the search leads directly to a terminal node, along an unproductive branch, where the algorithm backtracks. The correctness of this partitioning scheme follows from the fact that $w(C_0)$ is a monotonic, non-decreasing function of edge weights in G and the partitioning is based on a straightforward inclusion-exclusion technique. The actual implementation of the algorithm could be difficult, and efficiency poor, because edge inclusion may not be so simple to ensure (although exclusion is).

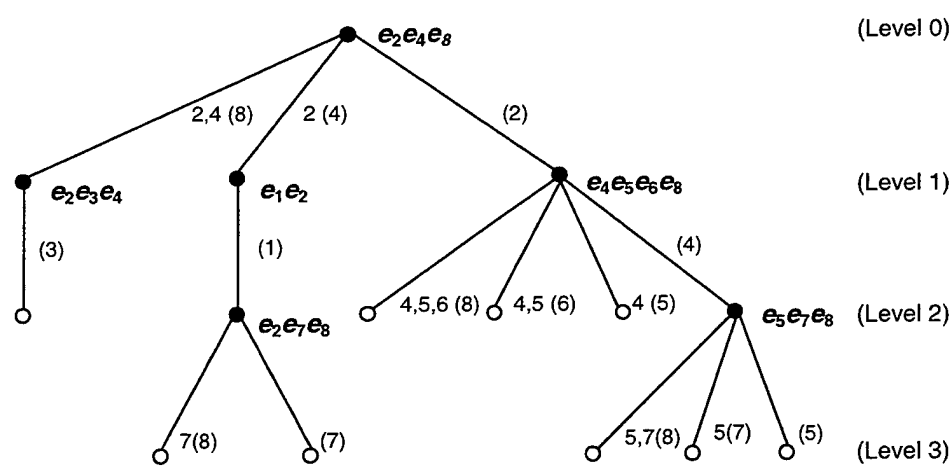


Figure 3. Enumeration Tree for the Graph of Figure 2. The enumeration scheme is represented from top to bottom and right to left. Each black node corresponds to a cut (edges given in bold next to the node) whose weight is no larger than $1+\varepsilon = 1+0.4$ times the min-cut weight, i.e., $W_{MAX} = \lfloor(1+0.4)\times 3\rfloor = 4$. Numbers in parentheses represent the number of the edge excluded from a cut; whereas other numbers represents edges to be included. White nodes are terminal nodes where the algorithm backtracks. The branches arriving into a terminal node are called "unproductive branches."

We next present a "relaxed" version of Algorithm A1, denoted "Algorithm A2," which may waste time working with non-minimal cuts: It partitions the space of minimal and non-minimal cuts but only prints out the minimal cuts. Our final algorithm, Algorithm B (Figure 5), closely mimics Algorithm A2 (Figure 4), although Algorithm B need not enumerate all near-min, non-minimal cuts.

Algorithm A2

INPUT: A directed network $G = (V, E)$, s, t, $\mathbf{w}$ and ε

OUTPUT: All minimal s-t cuts C such that $w(C) \leq (1 + \varepsilon) \times w(C_0)$ where C_0 is a minimum weight s-t cut of G.

```
begin
    Same as Algorithm A1;
end

Procedure ENUMERATE (G, s, t, w, E+, E-, W_MAX)
  begin
6     if (∃ a minimal or non-minimal cut C' s.t. E+ ⊂ C' and E- ∩ C' = ∅)
7         then let C_ε ← C'_0 and W_NEW ← w(C'_0) where C'_0 is a min-weight
                  minimal or non-minimal cut satisfying the conditions for C';
8         else return;
9     if (W_NEW > W_MAX) then return;
10    if (C_ε is minimal) then print (W_NEW, C_ε);
11    for (each edge e ∈ { C_ε \ E+ })
12        E- ← E- ∪ {e};
13        ENUMERATE (G, s, t, w, E+, E-, W_MAX);
14        E- ← E- \ {e};
15        E+ ← E+ ∪ {e};
      endfor
16    return;
  end
```

Figure 4. Algorithm A2: A "relaxed" version of Algorithm A1.

B. AN IMPLEMENTABLE ALGORITHM

Algorithm B, given in Figure 5, implements a variant of Algorithm A2, which may or may not produce non-minimal cuts. We *quasi-exclude* an edge $e = (u,v)$ from possible cuts at subsequent levels of the enumeration tree by simply setting $w(e) = \infty$. (In fact, we use a large number z instead of ∞, i.e., $z > \sum_{e \in E} w_e$, but for notational convenience we use ∞.) It is clear that every near-minimum cut in G should have a finite weight, and thus e with $w(e) = \infty$ cannot be contained in $C_\varepsilon(G)$. This means that quasi-exclusion implements true exclusion. The graph G with edge e quasi-excluded is denoted $G - e$.

We *quasi-include* $e = (u,v)$ by adding two additional edges to G as follows: One edge (s,u) connects the source to u, and the other edge (v,t), connects v to the sink; both new edges have infinite weights. The graph G with edge e quasi-included is denoted $G + e$. It is clear that if $G + e$ contains a finite-weight cut, all such cuts must contain $e = (u,v)$ because, by construction, u must be on the s side of such a cut, v must be on the t side of such a cut, and that implies that e is a forward arc in the cut. Actually, we implement quasi-inclusion in Algorithm B by temporarily treating u as an additional source and v as an additional sink.

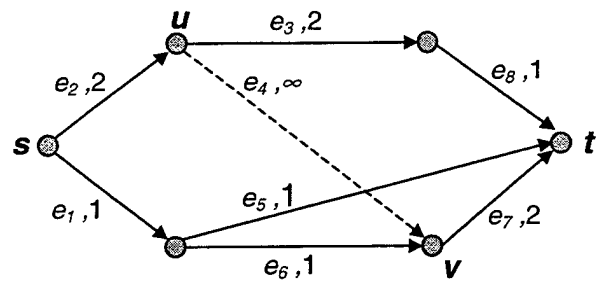
Figure 6 illustrates the quasi-inclusion and -exclusion of edges. $G + E^+ - E^-$ will denote G with the set of edges E^+ quasi-included and the set of edges E^- quasi-excluded.

ALGORITHM B

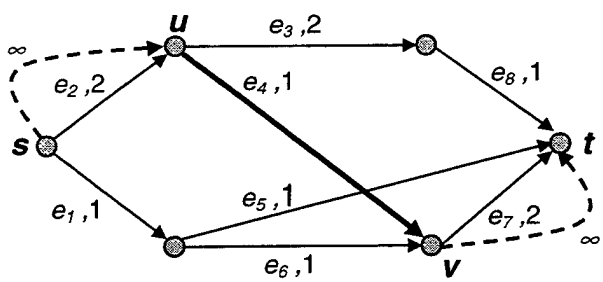
```
INPUT:    A directed network G = (V, E), s, t, w and ε
OUTPUT:   All minimal s-t cuts C in G such that w(C) ≤ (1 + ε) × w(C0)
  begin
1     [W_MIN, C0] ← MAXFLOW (G, s, t, w);
2     W_MAX ← (1 + ε) × W_MIN ;
3     E+ ← Ø;          /* set of edges to be included */
4     E- ← Ø;          /* set of edges to be excluded */
5     ENUMERATE (G, s, t, w, E+, E-, W_MAX) ;
  end

Procedure ENUMERATE (G, s, t, w, E+, E-, W_MAX)
  begin
6     w' ← w;
7     for (each edge e=(u, v) in E-)  w'(u, v) ← ∞;
8     for (each edge e=(u, v) in E+)
9         add artificial edge (s, u) to G and let w'(s, u) ← ∞;
10        add artificial edge (v, t) to G and let w'(v, t) ← ∞;
      endfor
      /* w' is now interpreted to include artificial edges */
11    [W_NEW, C_ε] ← MAXFLOW (G, s, t, w');
12    if (W_NEW > W_MAX) then return;
13    if (C_ε is minimal) then print (W_NEW, C_ε);
14    for (each edge e ∈ { C_ε \ E+ })
15      E- ← E- ∪ {e};
16      ENUMERATE (G, s, t, w, E+, E-, W_MAX);
17      E- ← E- \ {e};
18      E+ ← E+ ∪ {e};
19    endfor
20    return ;
  end
```

Figure 5. Algorithm B: An approximate implementation of Algorithm A2.



(a)



(b)

Figure 6. Quasi-inclusion and -exclusion of an edge from a cut. (a) Edge e_4 is quasi-excluded from a cut by setting $w(e_4)=\infty$. (b) Edge e_4 is quasi-included in a cut by adding infinite weight edges (s,u) and (v,t) to the graph.

C. CORRECTNESS OF THE ALGORITHM

Algorithm A2 would correctly find all near-minimum minimal cuts if it were partitioning the set of all minimal cuts in G along with some, rather than all, of the non-minimal cuts. That is essentially what Algorithm B does. Algorithm B begins by finding a minimum cut in G and determining W_{MAX} using a max-flow algorithm, all in an obviously correct manner. That is, the main routine of Algorithm B correctly implements the main routine of Algorithm A2. Then, where Algorithm A2 finds a min cut, minimal or non-minimal, that includes edges in E^+ and excludes edges in E^-, Algorithm B solves a max-flow problem and finds a min cut, minimal or non-minimal, in $G + E^+ - E^-$. Algorithm B is clearly finite and will be correct as long as (a) the weight of the locally minimum cut is monotonically non-decreasing, and (b) it correctly partitions the space of all minimal and possibly some of the non-minimal cuts in G. We first show point (a).

Lemma 3.1 $w(C_0)$ is monotonically non-decreasing for each call to *ENUMERATE* in Algorithm B.

Proof: When *ENUMERATE* is called, the finite weight of one edge in G has been increased to infinity, and some infinite-weight edges may have been added to G. Clearly, this cannot decrease the maximum flow (which equals the minimum cut weight) in G and thus $w(C_0)$ is monotonically non-decreasing. ■

Now, point (b) above will be satisfied if no non-minimal cuts are lost in the calls to *ENUMERATE* and none are repeated. The following two lemmas suffice to prove this.

Lemma 3.2 Let C be a finite-weight set of edges in G and let E^+ and E^- be quasi-inclusion and quasi-exclusion sets, respectively, produced while running Algorithm B.

Suppose that $C \cap E^+ = E^+$ and $C \cap E^- = \varnothing$. Then C is a minimal cut of G only if C is also a finite-weight minimal cut of $G + E^+ - E^-$.

Proof: So, we suppose that C is a minimal cut of G. The minimality of this cut is not affected by the weight of any edge, so C is also minimal cut of $G - E^-$. C is also a cut of $(G - E^-) + E^+ = G - E^- + E^+$ because the only way to stop this, while transforming $G - E^-$ into $G - E^- + E^+$, would be to add one or more edges from the s side of C to the t side of C. But, quasi-inclusion of E^+ only adds edges between nodes on the same side of the cut. So, C is definitely a cut $G - E^- + E^+$. Is it a minimal cut? No paths in $G - E^-$ have been deleted by adding the edges associated with quasi-inclusion of E^+ (to obtain $G - E^- + E^+$ from $G - E^-$). Thus, all edges of C must be deleted from $(G - E^-) + E^+ = G - E^- + E^+$ in order to disconnect s from t in that graph; this means that C is a minimal cut in $G - E^- + E^+$. Finally, C must be a finite-weight minimal cut in $G - E^- + E^+$ because it contains no infinite-weight edges. ■

Lemma 3.3: Let C be a set of edges in G and let E^+ and E^- be quasi-inclusion and quasi-exclusion sets, respectively, produced while running Algorithm B. Suppose that $C \cap E^+ \neq E^+$ or $C \cap E^- \neq \varnothing$. Then, C is a not a finite-weight minimal cut of $G + E^+ - E^-$.

Proof: From a previous argument, we know that quasi-exclusion properly implements edge exclusion. Thus, C cannot be a finite-weight minimal cut of $G + E^+ - E^-$ if some edge of C has been excluded, i.e., if $C \cap E^- \neq \varnothing$. Also from a previous argument, we know that all finite-weight minimal cuts of $G + E^+$ must contain E^+, i.e., C cannot be a finite-weight minimal cut of $G + E^+ - E^-$ if $C \cap E^+ \neq E^+$. ■

Theorem 3.1 Algorithm B correctly solves ANMCP-st.

Proof: Lemmas 3.1 through 3.3 show that Algorithm B correctly partitions the set of minimal cuts in G (along with, possibly, some non-minimal cuts): The partitioning process never loses a minimal cut by Lemma 3.2. Also, the process never repeats a minimal cut by Lemma 3.3 and the fact that the algorithm backtracks if the "min-weight cut" has infinite weight. Lemma 3.1 shows that the weight of the locally minimum cut in Algorithm B is monotonically non-decreasing and, therefore, the algorithm is finite and cannot miss enumerating a minimal cut because certain cut weights are "skipped" due to backtracking ■

D. COMPLEXITY OF THE ALGORITHM

Algorithm B can be used to solve both AMCP-st and ANMCP-*s-t*, depending on the choice of ε. We first give a complexity analysis for all min-cut enumeration, AMCP-st.

1. Complexity Analysis of Minimum-Cut Enumeration

Consider the enumeration tree given in Figure 3. Every node in that tree is either productive, and defines a new cut, or it is a terminal node and is immediately backtracked from. Unfortunately, the quasi-inclusion technique might result in the identification of one or more non-minimal cuts, i.e., some productive nodes might correspond to non-minimal cuts. Fortunately, any non-minimal cut encountered while solving AMCP-st cannot be productive and an efficient procedure results.

We know that the worst-case complexity of solving, from scratch, an initial max flow problem on $G = (V,E)$ is $O(f(n,m))$, where $f(n,m)$ is a polynomial function of $n = |V|$ and $m = |E|$. At each non-root node of the enumeration tree, the local max flow is

obtained by attempting to perform flow augmentations starting with the feasible flow from the parent node. (This will be shown to be correct later.) Each flow augmentation requires $O(m)$ work using breadth-first search in a standard fashion. It then turns out that the total amount of work performed at each node is $O(m)$, because (a) if the first search does not find a flow-augmenting path, a new min cut has been identified, and (b) if a flow-augmenting path is found, the locally maximum flow is at least $W_{MIN} + 1$ and the algorithm can backtrack immediately. (The algorithm must be modified slightly to do this.) Now a non-minimal cut must have a weight of at least $W_{MIN} + 1$, so no such cuts correspond to productive nodes. Thus the number of productive nodes is $|\boldsymbol{C}_0(G)|$.

Now, each productive node can generate at most n non-productive nodes (assuming G has no parallel arcs), so the total number of nodes generated is bounded by $(n + 1)\ |\boldsymbol{C}_0(G)|$. As described above, the amount of work to generate each node except the first is $O(m)$, and the amount of work to generate the first node is $O(f(n,m))$, so we have the following result.

Theorem 3.3. Algorithm B finds all minimum cuts (solves AMCP-st) in $O(f(n,m) + mn\,|\boldsymbol{C}_0(G)|)$ time. ■

This shows that the time complexity of Algorithm B is polynomial in the number of minimum cuts enumerated and the size of graph. Algorithm B is somewhat less efficient for solving AMCP-st than are some other algorithms from the literature: The algorithm of Picard and Queyranne (1980) solves ANMCP-st in $O(f(n,m) + (m+n)(|\boldsymbol{C}_0(G)|))$ time and the algorithm of Ball and Provan (1983) solves the problem in $O(nm + m\,|\boldsymbol{C}_0(G)|)$ time. Nevertheless, our algorithm has several

advantages in that (a) it is easy to implement, (b) empirically, it tends to run in $O(f(n,m)+m\,|C_0(G)|)$ time and, (c) it extends trivially to near-min cut enumeration, i.e., to solving ANMCP-st.

2. Complexity Analysis of Near-Minimum Cut Enumeration

By the arguments of the preceding section, the number of productive nodes in enumeration tree should be bounded by $(n+1)(|C_\varepsilon(G)|+|C'_\varepsilon(G)|)$, where $C'_\varepsilon(G)$ denotes the set of near-minimum, non-minimal cuts identified as productive nodes of Algorithm B's enumeration tree. The test for non-minimality, which utilizes a breadth-first search, takes $O(m)$ time at each node. The search for a local max flow might require multiple flow augmentations and might be as hard as solving for a max flow from scratch. Therefore, the work expended at every node is $O(f(n,m)+m)=O(f(n,m))$. If we could backtrack whenever a non-minimal cut was identified, then we could say $|C'_\varepsilon(G)|\leq n\,|C_\varepsilon(G)|$ because no node could have more than n descendants and the resulting complexity for the whole algorithm would be $O(nf(n,m)\,|C_\varepsilon(G)|)$. In near-minimum cut enumeration, however, non-minimal cuts are acceptable if their weight does not exceed $(1+\varepsilon)W_{MIN}$, and empirical tests show that backtracking when a non-minimal cut is encountered can result in the loss of some valid minimal cuts. Thus, Algorithm B must ignore non-minimal cuts and continue until it can backtrack based on cut weight. This results in a complexity of $O(nf(n,m)(|C_\varepsilon(G)|+|C'_\varepsilon(G)|))$, which may not be polynomial if $|C'_\varepsilon(G)|$ is exponentially larger than $|C_\varepsilon(G)|$. Therefore, the worst-case complexity of Algorithm B for near-minimum cut enumeration is not well determined. We leave this complexity issue as a topic for future research.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. COMPUTATIONAL RESULTS

In this chapter, we report computational experiments with Algorithm B to demonstrate its correctness and efficiency for solving both AMCP-st and ANMCP-st. We test Algorithm B on both weighted and unweighted grid graphs and on various problem instances from the literature.

Algorithm B is written and compiled using the Java 1.2.2 (Sun Microsystems 1998) programming language. All tests are performed on a personal computer with a 733 MHz Pentium III processor and 128 MB of RAM, running under the Windows 98 SE operating system.

A. IMPLEMENTATION DETAILS

1. Efficient Implementation of Algorithm B

Here, we discuss some improvements of Algorithm B that empirically yield a more efficient algorithm for solving both AMCP-st and ANMCP-st.

The *MAXFLOW* routine of Algorithm B is implemented to solve an "incremental" max flow problem rather than solving for a max flow "from scratch." This can be done because the max flow F* in G is a feasible flow F in $G + E^{+} - E^{-}$. Consider the enumeration tree introduced in Chapter III, and let G_k and F_k* be the modified graph and its corresponding max flow value, respectively, at node k. Each max flow problem at any descendant node of k is a relaxation (with respect to edge weights) of the max flow problem at node k. Therefore, F_l* in G_l, at any immediate child of k, can be reoptimized from F_k* in G_k by using an augmenting-path algorithm. (See Ahuja et al., 1993, pp. 180-

184 for details.) A complete max flow problem is only solved at root r; the remaining max flow problems at any node of the enumeration tree can be obtained by using F* of parent nodes. In fact, it takes at most $O(m)$ work at each node when solving AMCP-st, because we need to attempt only a single flow augmentation before we find a new max flow or discover that the local max flow is greater than W_{MAX} which is equal to W_{MIN} in AMCP-st. Unfortunately, for ANMCP-st, multiple flow augmentations might be needed to re-optimize the flow and this might require as much work as solving a complete max flow problem. In practice, much less work is required, however.

The test for the non-minimality of a cut C_ε identified in Algorithm B is performed only while solving ANMCP-st. The algorithm checks for non-minimal cuts right after the identification of each cut in MAXFLOW. It performs a (breadth-first) search starting at s trying to reach as many vertices as possible without traversing any cut edges and keeps track of the vertices that have been reached. Then, a similar search is performed backward from t. A cut is identified as minimal if for every edge $e = (u,v) \in C_\varepsilon$, u is reached from s and v is reached from t. Otherwise, the cut is non-minimal and the algorithm does not print it.

One other issue in efficient implementation is edge inclusion. In theory, we quasi-include an edge (u,v) by adding infinite-weight edges (s,u) and (v,t) to the graph, but in practice we simulate this by simply treating u as an additional source and v as an additional sink.

The rest of the implementation is straightforward. We use forward and reverse star representation as our data structure (Ahuja et al. 1993, pp. 35-38) and the shortest

flow-augmenting-path algorithm of Edmonds and Karp (1972) for solving max flow problems.

2. Problem Generators

In our literature search, we have not seen any particular problem family designed to generate instances for testing algorithms for AMCP-st and ANMCP-st, except for Grid Graph Families (GGFs) (Curet 1999, Gibbons 2000). The problem generators, however, for MCP, MCP-st (e.g., Levine 1997) and the max flow problems at DIMACS (The Center for Discrete Mathematics and Theoretical Computer, DIMACS 1991) are easy to modify for our purposes. Therefore, we use GGF problems and several problem classes from Levine (1997) and DIMACS to test Algorithm B.

We code a GGF Generator (GGFGEN) in Java to generate grid graphs. The width W of the graph, and its length L, determine the size of the generated graph. Other parameters are ε and q, where ε determines the near-minimum weight criterion and q indicates whether the graph is weighted ($q = 1$) or unweighted ($q = 0$). For both weighted and unweighted graphs, the edges beginning and ending in s and t, respectively, have infinite weights. Every grid vertex u (other than s and t) is connected to each adjacent (vertically and horizontally, assuming it exists) vertex v with two directed edges, (u,v) and (v,u). Edge weights are 1 in the unweighted case, and pseudo-random, uniformly distributed integer weights in range [1,10] otherwise. GGFGEN produces a (connected) directed graph with $(WL+2)$ vertices and $2(W+(2WL-W-L))$ edges. Figure 7 shows a grid graph generated by GGFGEN with inputs $W = 3$, $L = 4$, $q = 0$ and any $\varepsilon \geq 0$.

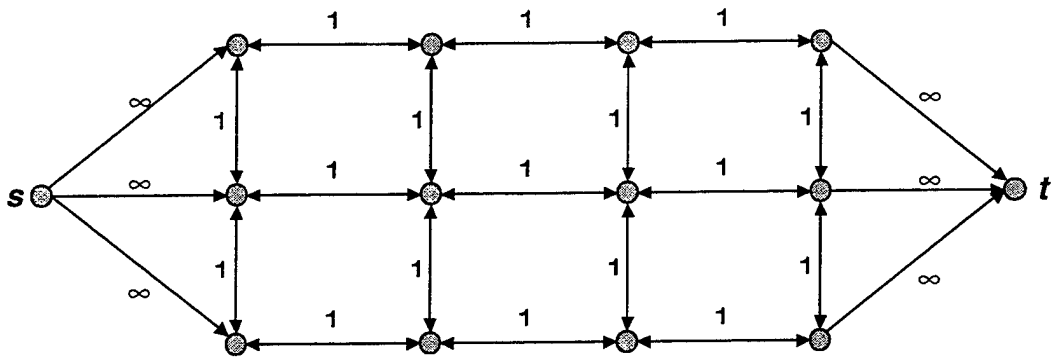


Figure 7. An unweighted, directed grid graph generated by GGFGEN with inputs $W = 3$, $L = 4$, $q = 0$ and any $\varepsilon \geq 0$. Edges incident to s and t have infinite weights. Other weights are all 1. Bi-directional edges between u and v represent two directed edges, (u,v) and (v,u).

We consider different weighted and unweighted instances of GGF with $W = L$ (GGF-square) and $W << L$ (GGF-long) for unweighted graphs. Table 1 gives the data for the generated problem graphs.

Problem Name	W	L	ε	q
GGF-square	5,10,15,20,25,30, 40,50,60,70,80,90	5,10,15,20,25,30, 40,50,60,70,80,90	0.0, .05, .10, .15	0, 1
GGF-long	25	100,125,150,175, 200,225,250	0.0	0

Table 1. Problem groups for GGF. A $W \times L$ grid of vertices is generated and adjacent vertices are connected by bi-directional, vertical and horizontal edges. For $q = 0$, the graph is unweighted (all edge weights are 1), and for $q = 1$, the graph is weighted with pseudo-random integer weights in the range [1,10].

We have also chosen two other generators from the literature, implemented in the C language and available via Internet for research use. The first is the Double-Cycle Generator (DBLCYCLEGEN) (Levine 1997). The single input parameter for DBLCYCLEGEN is n, the number of vertices. DBLCYCLEGEN generates two interleaved cycles such that the outer cycle has n vertices with the edge weights of 1000 and 997, and the inner cycle connects every third vertex of the outer cycle with the edges of weights 1 or 4. A minimum cut lies in the middle of the graph with a weight of 2000 and there are many near-min cuts of the weight 2006.

The second generator is the Acyclic Dense (AD) graph generator from DIMACS (1991). AD takes n as its input parameter and generates a fully dense, directed acyclic graph with n vertices and $m = n\ (n-1)/2$ edges. We replace the pseudo-randomly generated edge weights in AD with unit weights to observe the behavior of our algorithm on the underlying topologic structure.

Table 2 gives the generated problem types for DBLCYCLEGEN and AD.

Problem Name	n	ε
DBLCYC-I (Levine 1997)	500	0.00, 0.10, 1.25, 1.50, 1.75, 2.00
AD-I (DIMACS 1991)	50	0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70

Table 2. Problem types for DBLCYCLEGEN and AD. Graphs of type DBLCYC-I are generated with DBLCYCLEGEN and have $n = 500$ vertices and $m = 1000$ edges on two, interleaved, directed cycles. The outer cycle has edge weights of 1000 and 997, and the inner one has weights of 1 and 4. The edges between the cycles are weighted so as to hide a min-cut with a weight of 2000 in the middle of the graph. Graphs of type AD-I are fully dense, directed acyclic graphs with $n = 50$ vertices and $m = 1225$ unweighted edges.

B. THE EXPERIMENTS

Run times are specified as the total of the time (a) spent in *MAXFLOW* (MF TIME) including the initial max flow at the root node, (b) the time for identifying and searching for a new cut (ID TIME), and (c) the time for determining if the new cut is minimal (CHECK TIME). All times are given in seconds. Graph input time is short, and not reported.

1. Experiments on Unweighted Graphs

Table 3 presents run times of Algorithm B on GGF instances for solving AMCP-st. It takes less than 1 second for Algorithm B to identify all minimum cuts in grid graphs with sizes up to 402 vertices and 1,560 edges. The number of calls to *MAXFLOW* increases about linearly with n. Because no non-minimal cuts are enumerated, most of the running time is spent identifying and searching for a new cut. Also, note that CHECK TIME increases as graph size increases. This is due to the $O(m)$ running time of breadth-first search used to check for cut minimality.

Problem Name	$\|V\|$	$\|E\|$	$\|C_0\|$	$\|C_0(G)\|$	Non-min. Cuts[1]	Calls to MF[2]	MF TIME[3] (sec.)	CHECK TIME[4] (sec.)	ID TIME[5] (sec.)	TOTAL TIME[6] (sec.)
GGF10×10	102	380	10	9	0	92	0.05	0.00	0.05	0.10
GGF20×20	402	1560	20	19	0	382	0.00	0.05	0.00	0.05
GGF30×30	902	3540	30	29	0	872	0.00	0.00	0.05	0.05
GGF40×40	1602	6320	40	39	0	1562	0.10	0.00	0.05	0.15
GGF50×50	2502	9900	50	49	0	2452	0.11	0.00	0.17	0.28
GGF60×60	3602	14280	60	59	0	3542	0.21	0.00	0.28	0.49
GGF70×70	4902	19460	70	69	0	4832	0.48	0.05	1.56	2.09
GGF80×80	6402	25440	80	79	0	6322	1.15	0.10	4.58	5.83
GGF25×100	2502	9800	25	99	0	2477	0.00	0.12	0.05	0.17
GGF25×125	3127	12250	25	124	0	3102	0.06	0.33	0.77	1.16
GGF25×150	3752	14700	25	149	0	3727	0.06	0.61	2.30	2.97
GGF25×175	4377	17150	25	174	0	4352	0.11	0.90	4.64	5.65
GGF25×200	5002	19600	25	199	0	4977	0.11	1.55	7.34	9.00
GGF25×225	5627	22050	25	224	0	5602	0.11	1.88	10.97	12.96
GGF25×250	6252	24500	25	249	0	6227	0.17	2.13	15.34	17.64

1 Non-Min. Cuts : Number of the non-minimal cuts enumerated.
2 Calls to MF : Number of calls to *MAXFLOW*.
3 MF TIME : Total amount of the time spent in *MAXFLOW* including the initial complete max flow problem.
4 CHECK TIME : Total time for determining the if each new cut is minimal.
5 ID TIME : Total time for identifying and searching for a new cut.
6 TOTAL TIME : Total running time in seconds.

Table 3. Run times (in seconds) for Algorithm B solving AMCP-st on unweighted instances of GGF-square and GGF-long graphs. No non-minimal cuts are enumerated because the weight of a non-minimal cut is at least $W_{MIN} + 1$, and the algorithm backtracks immediately if a cut with weight greater than W_{MIN} is found. GGFW×L denotes a GGF graph with a $W \times L$ grid of vertices.

Table 4 summarizes the results for ANMCP-st on GGF-square instances with $\varepsilon = 0.05$, 0.10, and 0.15. Solution times are expected to increase as ε increases, because the number of cuts in any graph might be exponential in the size of the graph. The algorithm is quite efficient for modest-size grid graphs with reasonable ε values. Compared with Gibbons' results for near-minimum cut enumeration (Gibbons 2000), our results indicate an exponential decrease in calls to *MAXFLOW* and run times.

Problem Name	$\|V\|$	$\|E\|$	$\|C_0\|$	$\|C_\varepsilon\|$	$\|C_\varepsilon(G)\|$	Non-min. Cuts	Calls to MF	MF TIME (sec.)	CHECK TIME (sec.)	ID TIME (sec.)	TOTAL TIME (sec.)
ε = 0.05											
GGF5×5	27	90	5	5	4	0	22	0.05	0.00	0.05	0.10
GGF10×10	102	380	10	10	9	0	92	0.00	0.00	0.06	0.06
GGF15×15	227	870	15	15	14	0	212	0.00	0.06	0.00	0.06
GGF20×20	402	1560	20	21	703	0	7906	0.77	0.00	0.71	1.48
GGF25×25	627	2450	25	26	1128	0	15506	1.62	0.15	2.14	3.91
GGF30×30	902	3540	30	31	1653	0	26856	3.92	0.49	7.55	11.96
ε = 0.10											
GGF5×5	27	90	5	5	22	0	22	0.05	0.00	0.05	0.10
GGF10×10	102	380	10	11	956	0	956	0.00	0.06	0.05	0.11
GGF15×15	227	870	15	16	3306	0	3306	0.23	0.00	0.27	0.50
GGF20×20	402	1560	20	22	113090	0	113090	9.11	1.35	9.63	20.09
GGF25×25	627	2450	25	27	274550	0	274550	30.98	3.83	39.62	74.43
GGF30×30	902	3540	30	33	8911698	378	8911698	1553.89	182.78	2798.42	4535.09
ε = 0.15											
GGF5×5	27	90	5	5	22	0	22	0.05	0.00	0.05	0.10
GGF10×10	102	380	10	11	956	0	956	0.05	0.05	0.06	0.16
GGF15×15	227	870	15	17	35905	0	35905	1.69	0.42	2.77	4.88
GGF20×20	402	1560	20	23	1202033	153	1202033	102.92	13.70	98.70	215.32
GGF25×25	627	2450	25	28	3621978	253	3621978	420.08	56.83	496.38	973.29
GGF30×30	902	3540	30	34	13465371	21843	113463496	18537.49	2609.65	25248.65	46395.79

Table 4. Run times in seconds for ANMCP-st solutions of Algorithm B on unweighted GGF-square instances with ε = 0.05, 0.10, 0.15. As expected, the solution times increase as ε increases. For example, 8,911,698 near-min cuts together with 378 non-minimal cuts are enumerated in 4,535.09 seconds in GGF30×30 for ε = 0.10. For ε = 0.15 , on the same graph, 13,465,371 near-min cuts and 21,843 non-minimal cuts are enumerated in 46,395.79 seconds.

Table 5 presents results for unweighted AD graphs. Interestingly, Algorithm B does not enumerate any non-minimal cuts even though ε is relatively large in some instances. Given our concern about enumerating non-minimal cuts, this invites further investigation.

ε	$\|C_0\|$	$\|C_\varepsilon\|$	$\|C_\varepsilon(G)\|$	Non-min. Cuts	Calls to MF	MF TIME (sec.)	CHECK TIME (sec.)	ID TIME (sec.)	TOTAL TIME (sec.)
0.0	49	49	49	0	1275	0.21	0.00	0.11	0.32
0.10	49	53	544	0	13650	2.28	0.00	0.85	3.13
0.20	49	58	4063	0	101625	16.56	0.22	8.92	25.70
0.30	49	63	19798	0	495000	94.86	0.64	39.40	134.90
0.40	49	68	75893	0	1897375	388.38	3.96	150.61	542.95
0.50	49	73	249270	0	6231800	1356.31	15.90	489.37	1861.58
0.60	49	78	730603	0	18265125	4077.99	42.87	1423.98	5544.84
0.70	49	83	1962849	0	49071275	11188.42	121.20	3816.87	15126.49

Table 5. Computational results on unweighted, acyclic dense (AD) graphs with various threshold levels ε. This is a fully dense graph with 50 vertices and 1225 edges. Although the graph is fully dense, the number of non-minimal cuts enumerated for all values of ε is zero.

2. Experiments on Weighted Graphs

Here we use the GGF-square problems with edge weights pseudo-randomly generated integers in the range [1,10]. Results for minimum and near-minimum cut enumeration are summarized in Tables 6 and Table 7, respectively.

Problem Name	$\|V\|$	$\|E\|$	W_{MIN}	$\|C_0(G)\|$	Non-min. Cuts	Calls to MF	MF TIME (sec.)	CHECK TIME (sec.)	ID TIME (sec.)	TOTAL TIME (sec.)
GGF5×5*w*	27	90	17	1	0	7	0.06	0.00	0.06	0.12
GGF10×10*w*	102	380	42	2	0	92	0.00	0.00	0.05	0.05
GGF15×15*w*	227	870	45	1	0	19	0.00	0.00	0.05	0.05
GGF20×20*w*	402	1560	69	1	0	32	0.06	0.00	0.06	0.12
GGF25×25*w*	627	2450	87	1	0	33	0.00	0.00	0.05	0.05
GGF30×30*w*	902	3540	108	6	0	217	0.05	0.06	0.16	0.27

Table 6. Computational results for minimum-cut enumeration (AMCP-st) on weighted, GGF-square problems. As in the unweighted case, no non-minimal cut are encountered. All minimum cuts are identified in less than one second for these instances.

Problem Name	$\|V\|$	$\|E\|$	W_{MIN}	W_{MAX}	$\|C_\varepsilon(G)\|$	Non-min. Cuts	Calls to MF	MF TIME (sec.)	CHECK TIME (sec.)	ID TIME (sec.)	TOTAL TIME (sec.)
ε = 0.05											
GGF5×5*w*	27	90	17	17	1	0	7	0.05	0.00	0.05	0.10
GGF10×10*w*	102	380	42	44	5	0	46	0.00	0.00	0.05	0.05
GGF15×15*w*	227	870	45	47	3	0	43	0.00	0.05	0.00	0.05
GGF20×20*w*	402	1560	69	72	20	0	308	0.11	0.00	0.06	0.17
GGF25×25*w*	627	2450	87	91	33	0	645	0.16	0.00	0.11	0.27
GGF30×30*w*	902	3540	108	113	416	0	7221	1.59	0.17	2.04	3.80
ε = 0.10											
GGF5×5*w*	27	90	17	18	1	0	7	0.06	0.00	0.06	0.12
GGF10×10*w*	102	380	42	46	11	0	78	0.05	0.00	0.05	0.10
GGF15×15*w*	227	870	45	49	6	0	75	0.00	0.00	0.06	0.06
GGF20×20*w*	402	1560	69	75	116	0	1508	0.22	0.00	0.16	0.38
GGF25×25*w*	627	2450	87	95	309	0	4263	0.51	0.11	0.69	1.31
GGF30×30*w*	902	3540	108	118	8025	0	104836	25.60	1.85	31.66	59.11
ε = 0.15											
GGF5×5*w*	27	90	17	19	1	0	7	0.05	0.00	0.05	0.10
GGF10×10*w*	102	380	42	48	22	0	161	0.00	0.06	0.00	0.06
GGF15×15*w*	227	870	45	51	22	0	223	0.00	0.06	0.00	0.06
GGF20×20*w*	402	1560	69	79	1031	0	10744	1.54	0.06	0.88	2.48
GGF25×25*w*	627	2450	87	100	3345	22	35917	5.11	0.55	5.49	11.15
GGF30×30*w*	902	3540	108	124	144511	0	1503784	348.56	32.41	466.98	847.95

Table 7. Computational results for near-minimum cut enumeration (ANMCP-st) on weighted GGF-square problems. Non-minimal cuts are only encountered GGF25×25*w*.

Finally, we test Algorithm B on the DBLCYC-I problems with ε values ranging from 0.0 to 2.0. These tests are the only problem types where Algorithm B generates large numbers of non-minimal cuts compared to the number of near-minimum cuts. For $\varepsilon \geq 1.25$, the ratio of the number of non-minimal cuts to the number of minimal cuts increases dramatically; see Table 8.

ε	W_{MIN}	W_{MAX}	$\lvert C_\varepsilon(G)\rvert$	Non-min. Cuts	Calls to MF	MF TIME (sec.)	CHECK TIME (sec.)	ID TIME (sec.)	TOTAL TIME (sec.)
0.00	1000	1000	2	0	10	0.00	0.00	0.06	0.06
0.10	1000	1100	499	0	1976	0.21	0.05	0.28	0.54
1.00	1000	2000	511	8	2032	0.21	0.00	0.35	0.56
1.25	1000	2250	2479	237411	957178	96.73	21.32	89.80	207.85
1.50	1000	2500	2479	237411	957178	93.71	22.19	92.82	208.72
1.75	1000	2750	2479	237411	957178	92.76	23.16	91.48	207.40
2.00	1000	3000	2509	238041	959683	89.67	21.01	82.72	193.40

Table 8. Computational results for near-minimum cut enumeration (ANMCP-st) on weighted DBLCYC-I problems when n = 500 (and m = 1000). The number of the non-minimal cuts increases dramatically with increases in ε. For example, when ε = 2.00, i.e., W_{max} = (1+2.00)× 1000 = 3000, the number of non-minimal cuts is 238,041 whereas the number of near-min cuts is only 2509.

Our results show that Algorithm B performs quite well on various types of graphs, but that non-minimal cuts can slow computations when the threshold parameter ε becomes large. For dense acyclic graphs, Algorithm B does not enumerate any non-minimal cuts, but for the "double-cycle graphs" DBLCYC-I, the number of non-minimal cuts can outnumber the minimal cuts by a huge margin, at least when ε becomes large. This behavior of the algorithm suggests that it does have polynomial complexity for certain graph topologies, but not others. We believe this issue merits further investigation.

For large graphs where the time for flow augmentations dominates the overall running time, use of a state-of-the-art max flow algorithm such as Goldberg and Rao (1998) might reduce *MAXFLOW* time.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMEDATIONS

In this thesis, we have developed an algorithm for enumerating *all near-minimum-weight s-t cuts* in a directed graph $G = (V,E)$. The enumeration algorithm finds a minimum-weight cut in the input graph via a maximum flow algorithm and then recursively partitions the space of possible solutions to the "near-min threshold level" $W_{MAX} = (1+\varepsilon)\times W_{MIN}$, where W_{MIN} is the weight of that minimum weight cut. Given a minimum cut, the set of acceptable cuts is recursively partitioned by forcing inclusion and exclusion of edges from subsequent cuts. An edge (u,v) is *quasi-excluded* by simply setting its weight to the infinity and *quasi-included* by implicitly introducing two infinite-weight edges in G, one extending from s to u and the other from v to t. The algorithm solves a max flow min-cut problem for each modified graph that is obtained in the enumeration tree.

We have implemented our algorithm using the following enhancements to improve the efficiency: (1) The algorithm solves a complete max flow problem at the beginning (at the root node of enumeration tree) but solves only an "incremental" max flow problem at the all other nodes (the max flow at a parent node is feasible for all descendants and can easily be re-optimized), and (2) quasi-inclusion of an edge (u,v) is simulated by treating u as an additional source and v as an additional sink.

Unfortunately, our quasi-inclusion technique sometimes leads to the enumeration of non-minimal cuts together with the minimal cuts. Non-minimal cuts are easily identified (and ignored), but they can increase the computational workload and stop us

from deriving a polynomial-time bound for the worst-case complexity of the algorithm. We do obtain, however, a polynomial bound of $O(f(n,m)+nm|C_\varepsilon(G)|)$ for minimum-cut enumeration (which is the same as near-minimum cut enumeration but with $\varepsilon = 0$).

Computational results for $\varepsilon > 0$ show that the algorithm is empirically efficient for modest-sized problems with modest values for ε. For example, in an unweighted grid graph with 402 vertices and 1,560 edges, (a) all 19 minimum cuts can be enumerated in 0.05 seconds (on a 733 MHz Pentium III personal computer), (b) it takes 0.77 seconds to enumerate all 703 near-minimum cuts when $\varepsilon = 0.05$, and (c) it takes only 20.09 seconds to enumerate all 113,090 near-minimum cuts when $\varepsilon = 0.10$. However, in another unweighted graph with 102 vertices and 380 edges, and $\varepsilon = 0.50$, the algorithm enumerates 134,705 near-min cuts together with 4,474 non-minimal cuts in 601.2 seconds. For $\varepsilon = 0.90$, on the same graph, 7,811,043 near-min cuts and 1,941,792 non-minimal cuts are obtained in 2,303.5 seconds. The running times and the number of non-minimal cuts increase significantly for relatively large values of ε.

Although Algorithm B appears to be quite efficient in practice, more work might improve the quasi-inclusion technique or develop another technique for edge inclusion. If "true edge inclusion" (as opposed to quasi-inclusion) can be efficiently implemented, this should yield a provably polynomial-time algorithm for near-minimum cut enumeration.

If the current quasi-inclusion technique is retained, another approach might be used to avoid enumerating non-minimal cuts. In particular, edges that cannot occur in any minimal cut given those that are already included can be identified and marked as

"forbidden for inclusion." This means that they can be given an infinite weight and excluded because they are not allowed to appear in any minimal cut given the current inclusions. An edge (u,v) can be forbidden from inclusion if every $(s\text{-}u)$ or $(v\text{-}t)$ path contains at least one such included edge.

Another approach to avoid non-minimal cuts might be to take the advantage of max flow min-cut duality and corresponding linear programming (LP) techniques. This might be fruitful because our algorithm enumerates some of the near-minimum-cost bases in the dual of the maximum-flow LP. Although a minimal cut might correspond to more than one basis, a non-minimal cut certainly should be indicated by the absence of a basis. Use of linear algebra (linear independence or dependence of basic variables) to check the corresponding bases during the enumeration might lead to a new technique to eliminate non-minimal cuts.

Finally, it would be interesting to see if the enumeration algorithm, as it exists now, will enumerate only minimal cuts for certain types of graph topologies, e.g., undirected s-t planar graphs. Using the dual of a planar graph and shortest-path techniques, it is possible to enumerate near-min cuts in an undirected s-t planar graph in polynomial time per cut. Thus, it is natural to wonder if our algorithm can also enumerate such cuts efficiently.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Ahmad, S. H., 1990, "Enumeration of Minimal Cutsets of an Undirected Graph," *Microelectron. Reliab.*, Vol. 30, pp. 23-26.

Ball, M. O., and Provan, J. S., 1983, "Calculating Bounds on Reachability and Connectedness in Stochastic Networks," *Networks*, Vol. 13, pp. 253-278.

Boyle, M. R., 1998, "Partial-Enumeration For Planar Network Interdiction Problems," Master's Thesis, Operations Research Department, Naval Postgraduate School, Monterey, California, March.

Colbourn, C. J., 1987, *The Combinatorics of Network Reliability*, Oxford University Press.

Curet, N. D., 1999, "An Efficient Network Flow Code for Finding All Minimum Cost *s-t* Cutsets," Working Paper, U.S. Department of Defense, Fort Meade, Maryland.

DIMACS, 1991, *The First DIMACS International Algorithm Implementation Challenge,* Rutgers University, New Brunswick, New Jersey. (Available via anonymous ftp from dimacs.rutger.edu.)

Edmonds, J. and Karp, R. M., 1972, "Theoretical Improvements in Algorithm Efficiency for Network Flow Problems," *Journal of the ACM*, Vol. 19, pp. 248-264.

Fard, N. S., and Lee, T. H., 1999, "Cutset Enumeration of Network Systems with Link and Node Failures," *Reliab. Eng. System Safety*, Vol. 65, pp. 141-146.

Ford, L. R., and Fulkerson, D. R., 1956, "Maximal Flow through a Network," *Canadian Journal of Mathematics*, Vol. 8, pp. 399-404.

Gibbons, M., 2000, "Enumerating Near-Minimum Cuts in a Network," Master's Thesis, Operations Research Department, Naval Postgraduate School, Monterey, California, June.

Goldberg, A. V., and Rao, S., 1998, "Beyond the Flow Decomposition Barrier," *Journal of the ACM*, Vol. 45, pp. 783-797.

Gusfield, D., and Naor, D., 1993, "Extracting Maximal Information About Sets of Minimum Cuts," *Algorithmica*, Vol. 10, pp. 64-89.

Kanevsky, A., 1993, "Finding All Minimum-Size Separating Vertex Sets in a Graph," *Networks*, Vol. 23, pp. 533-541.

Karger, D. R., 1999, "A Randomized Fully Polynomial Time Approximation Scheme for the All-Terminal Network Reliablity Problem," *SIAM J. Computing*, Vol. 29, pp. 492-514.

Karger, D. R., 2000, "Minimum Cuts in Near-Linear Time," *Journal of ACM*, Vol. 47, pp. 46-76.

Karger, D. R., and Clifford S., 1996, "A New Approach to the Minimum Cut Problem," *Journal of ACM*, Vol. 43, pp. 601-640.

Levine, M. S., 1997, "Experimental Study of Minimum Cut Algorithms," Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, May. (Available at http://theory.lcs.mit.edu/~mslevine.)

Nagamochi, H., Ono, T., and Ibaraki, T., 1994, "Implementing an Efficient Minimum Capacity Cut Algorithm," *Mathematical Programming*, Vol 67, pp. 297-324.

Nagamochi, H., Nishimura, K., and Ibaraki, T., 1997, "Computing All Small Cuts in an Undirected Network," *SAIM J. Discrete Math.* Vol. 10, pp. 469-481.

Nahman, J. M., 1997, "Enumeration of Minimal Cuts of Modified Networks," *Microelectron. Reliab.*, Vol. 37, pp. 483-485.

Patvardhan, C., Prasad, V. C. and Pyara, V. P., 1995, "Vertex Cutsets of Undirected Graphs," *IEEE Trans. Reliab.*, Vol. 44, pp. 347-353.

Picard, J. C., and Queyranne, M., 1980, "On The Structure of All Minimum Cuts in a Network and Applications," *Math. Programming Study*, Vol. 13, pp. 8-16.

Picard, J. C., and Queyranne, M., 1982, "Selected Applications of Minimum Cuts in Networks," *INFOR*, Vol. 20, pp. 394-422.

Prasad, V. C., Sankar, V., and Rao, P., 1992, "Generation of Vertex and Edge Cutsets," *Microelectron. Reliab.*, Vol. 32, pp. 1291-1310.

Provan, J. S., and Ball, M. O., 1983, "The Complexity of Counting Cuts and of Computing The Probability That A Graph Is Connected," *SIAM J. Computing*, Vol. 12, pp. 777-788.

Provan, J. S., and Shier, D. R., 1996, "A Paradigm for Listing (s,t)-Cuts in Graphs," *Algorithmica*, Vol. 15, pp. 351-372.

Ramanathan, A., and Colbourn, C. J., 1987, "Counting Almost Minimum Cutsets with Reliability Applications," *Mathematical Programming*, Vol. 39, pp. 253-261.

Shier, D. R., and Whited, D. E., 1986, "Iterative Algorithms for Generating Minimal Cutsets in Directed Graphs," *Networks*, Vol. 16, pp. 133-147.

Sung, C. S., and Yoo, B. K., 1992, "Simple Enumeration of Minimal Cutsets Separating 2 Vertices in a Class of Undirected Planar Graphs," *IEEE Trans. Reliab.*, Vol. 41, pp. 63-71.

Sun Microsystems Inc., 1998, Java Platform Version 1.2.2.

Tsukiyama, S., Shirakawa, I., Ozaki, H., and Ariyoshi, H., 1980, "An Algorithm to Enumerate All Cutsets of a Graph in Linear Time per Cutset," *Journal of ACM*, Vol. 27, pp. 619-632.

Ubel, A., and Bicker, R., 1982, "Determination of All Minimal Cut-Sets between a Vertex Pair in an Undirected Graph," *IEEE Trans. Reliab.*, Vol. R-31, pp. 167-171.

Vazirani, V. V., and Yannakakis, M., 1992, "Suboptimal Cuts: Their Enumeration, Weight and Number," *Automata, Languages and Programming. 19th International Colloquium Proceedings*, Vol. 623 of Lecture Notes in Computer Science, Springer-Verlag, pp. 366-377.

Wood, R.K., 1993, "Deterministic Network Interdiction," *Mathematical and Computer Modeling*, Vol. 17, pp. 1-18.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Professor R. K. Wood 2
   Department of Operations Research
   Naval Postgraduate School, Code OR/Wd
   Monterey, CA 93943

4. Professor C. W. Rasmussen 1
   Department of Mathematics
   Naval Postgraduate School, Code MA/Ra
   Monterey, CA 93943

5. Professor G. G. Brown 1
   Department of Operations Research
   Naval Postgraduate School, Code OR/Bw
   Monterey, CA 93943

6. Av. Huseyin Celik (Ahmet Balcioglu) 3
   Hukumet Cad. No. 39
   Boyabat-Sinop 57200 TURKEY

7. Kara Kuvvetleri Komutanligi 1
   Kutuphane
   Bakanliklar, Ankara, TURKEY

8. Kara Harp Okulu Komutanligi 1
   Kutuphane
   Bakanliklar, Ankara, TURKEY

9. Bilkent Universitesi Kutuphanesi 1
   06533 Bilkent, Ankara, TURKEY

10. Orta Dogu Teknik Universitesi Kutuphanesi 1
    Balgat, Ankara, TURKEY