NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A BRANCH-AND-BOUND ALGORITHM FOR THE NETWORK DIVERSION PROBLEM

by

Ozgur Erken

December 2002

Thesis Advisor:	R. Kevin Wood
Second Reader:	Matthew Carlyle

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

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 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) A Branch-and-bound Algorithm for the Network Diversion Problem			**5. FUNDING NUMBERS**
6. AUTHOR(S) Erken, Ozgur			
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)***

In the network diversion problem (NDP), we must find a minimum-weight set of edges in a directed graph $G = (V, E)$ whose deletion forces all *s*-*t* communication to pass through one or more diversion edges in a diversion set E_D. We develop and test a specialized branch-and-bound algorithm for this NP-complete problem. The algorithm is based on partitioning the solution space with respect to edges in certain *s*-*t* cuts and yields a non-standard, non-binary enumeration tree. The algorithm is coded in Java version 1.4 and run on a 1.5 MHz Pentium IV computer with 384 megabytes of RAM. An instance of NDP on a grid graph with 2502 vertices, 9900 edges and one diversion edge is solved in 5.66 seconds; the same problem with 10 diversion edges is solved in only 0.84 seconds.

14. SUBJECT TERMS Networks, Cuts, Network Diversion, Simple Path, Enumeration, Branch-and-bound			**15. NUMBER OF PAGES** 55
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** 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.

A BRANCH-AND-BOUND ALGORITHM FOR THE NETWORK DIVERSION PROBLEM

Ozgur Erken
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
December 2002

Author: Ozgur Erken

Approved by: R. Kevin Wood
Thesis Advisor

Matthew Carlyle
Second Reader

James N. Eagle
Chairman, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In the network diversion problem (NDP), we must find a minimum-weight set of edges in a directed graph $G = (V, E)$ whose deletion forces all *s*-*t* communication to pass through one or more diversion edges in a diversion set E_D. We develop and test a specialized branch-and-bound algorithm for this NP-complete problem. The algorithm is based on partitioning the solution space with respect to edges in certain *s*-*t* cuts and yields a non-standard, non-binary enumeration tree. The algorithm is coded in Java version 1.4 and run on a 1.5 MHz Pentium IV computer with 384 megabytes of RAM. An instance of NDP on a grid graph with 2502 vertices, 9900 edges and one diversion edge is solved in 5.66 seconds; the same problem with 10 diversion edges is solved in only 0.84 seconds.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to express my appreciation to my advisor Professor Kevin Wood for his advice, guidance and patience throughout my course of study. Also, for his support, I wish to thank Professor Matthew Carlyle.

Special thanks go to my wife, Arzu. Without her love, support and encouragement, I would be lost.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

In this thesis we consider the network diversion problem (NDP) in a directed graph $G = (V, E)$. The objective of NDP is to cut links in *G*, using minimal resources, in order to force all flow from "source vertex" *s* to "sink vertex" *t* over at least one link in specified set of links.

NDP arises in the context of intelligence gathering when trying to force an opponent to communicate over a specified set of links in a communication network. It may also be relevant in warfare, if one wishes to force an opponent to travel over vulnerable links by destroying links of the opponent's transportation network. Finally, the solution to NDP may also be useful for finding a simple *s*-*t* path in a directed graph that contains a specified edge (link).

NDP has been modeled and solved as a binary integer program, but this approach has not proved very successful for problems of even moderate size.

An *s*-*t* cut in G is a set of edges whose removal from G disconnects all directed *s*-*t* paths. NDP is stated as the problem of finding a minimum-weight, minimal *s*-*t* cut that contains a diversion edge. We focus on the problem that has a single diversion edge $e = (u, v)$, and use that procedure as a subroutine to solve the multiple-edge problem.

In our study, we develop a branch-and-bound approach to NDP that relaxes the requirement of minimality in the cut identified. This approach first identifies a minimum-weight *s*-*t* cut that contains the diversion edge by "quasi-including" that edge in all cuts. An edge (u, v) is quasi-included by adding two artificial, infinite-weight edges, one from *s* to *u* and one from *v* to *t*, so that every finite-weight *s*-*t* cut in G contains (u, v). If this initial cut is a minimal cut, NDP is solved. Otherwise, we partition the feasible region of NDP using the quasi-inclusion and exclusion technique to create an appropriate branching scheme for NDP. An edge is excluded from a cut by setting its weight to infinity so that no relevant, finite-weight *s*-*t* cut contains that edge.

We use a non-binary branching scheme where there may be as many branches below a node in the enumeration tree as there are edges in a locally-minimum-weight cut. The algorithm backtracks whenever a minimal cut is identified or when the weight of the locally minimum-weight, non-minimal cut exceeds the weight of the incumbent solution.

The new branch-and-bound algorithm is implemented in the Java 1.4 programming language, and tested using a 1.5 GHz Pentium IV computer with 384 megabytes of RAM operating under Windows XP. In a 50 by 50 grid network with 2502 vertices, and 9900 edges with random weights, an optimal solution to NDP containing a randomly chosen link is found in 5.66 seconds; the same problem with 10 randomly chosen links is solved in 0.84 seconds.

I. INTRODUCTION

The network diversion problem involves cutting links in a network, using the least effort possible, to force the user of the network to communicate or travel over certain "exploitable" links. The problem arises in the context of intelligence gathering but may also have warfighting applications. This thesis develops a new, exact methodology for solving this NP-complete problem.

A. PROBLEM DEFINITION AND ITS APPLICATIONS

We consider a *directed network* (graph) $G = (V, E)$ composed of a set of *vertices* V and a set of *directed edges* $E \subseteq V \times V$. Two vertices are distinguished, a *source vertex* s and a *sink vertex* t. A special *diversion set* $E_D \subset E$ is defined; each $e \in E_D$ is a *diversion edge*. Each edge $e = (u, v) \in E$ has a *weight* (interdiction cost) $w_e \geq 0$ associated with it; this is the cost or amount of effort required to attack and destroy the edge. We assume that $w_e = 0$ for all $e \in E_D$, and $w_e > 0$ otherwise. A (*directed*) *path* in G is a sequence of edges connecting two vertices, with all edges pointing in the same direction. The *network diversion problem* (NDP) is: Find a minimum-weight set of edges C such that all *s*-*t* paths in $G - C \equiv (V, E \setminus C)$ must contain at least one edge of E_D. In other words, if no edges in C can be used, all flow from *s* to *t* must pass through one or more diversion edges.

NDP is stated more conveniently by defining a *minimal s-t cut*. This is a minimal set of edges C whose removal from G disconnects all directed *s*-*t* paths. A minimum-weight, minimal *s*-*t* cut, $C \subseteq E$, that contains at least one edge $e \in E_D$ is an optimum solution for NDP.

The intelligence-gathering applications of NDP are obvious. However, there may also be warfighting applications in which an "interdictor" will interdict (attack and destroy) links of his opponent's logistics network and force that opponent to move materiel or troops over certain links, i.e., diversion edges, that are vulnerable to subsequent attack.

A theoretical application of NDP arises in finding a simple (loopless) directed path from s to t that contains a specified edge e_d; this problem is shown to be NP-complete by Fortune, Hopcroft and Wyllie (1980). A feasible solution to NDP exists, with $E_D = \{e_d\}$ (and with $w_e = 1$ for all $e \in E \setminus E_D$, say), if and only if a feasible solution to the path-finding problem exists.

B. BACKGROUND

NDP has been studied by Curet (2001). He develops an integer-programming model (IP) for the problem but finds that IP difficult to solve. He uses heuristics and Lagrangian relaxation to solve the problem approximately. The Lagrangian bound could be embedded in a branch-and-bound algorithm for solving the problem exactly, but this is not implemented. Clearly, the problem is only partially solved: This thesis develops and implements a general methodology for solving NDP exactly.

C. THESIS OUTLINE

The remainder of this thesis is organized as follows: Chapter II describes our general branch-and-bound approach for solving NDP. Chapter III explains the algorithm in detail. Chapter IV presents computational results and Chapter V summarizes of our findings.

II. GENERAL METHODOLOGY

This chapter first reviews basic definitions and notation that will be used throughout this thesis. Subsequently, we outline a general branch-and-bound approach for solving the Network Diversion Problem.

A. DEFINITIONS AND NOTATION

An *s-t cut* in directed graph $G = (V, E)$ is a set of edges C such that $G - C \equiv (V, E \setminus C)$ contains no *s-t* paths. A *non-minimal cut* is a cut in which one of its proper subsets forms another cut. The weight or capacity of a cut C is $w(C) = \sum_{e \in C} w_e$.

The value of a max flow from *s* to *t* in G equals the capacity of a minimum-weight *s-t* cut in *G* according to the maximum-flow minimum-cut theorem (Ford and Fulkerson 1962, pp. 11-14).

A rooted (enumeration) tree, *T*, is a directed connected graph that contains no cycles, and has a distinguished initial node, called the *root*, which does not have any incoming edges. There is a simple directed path from the root node to all other nodes of a tree. The nodes on the simple path from a given node *i* to the root node are called *ancestors* of node *i*. All the nodes that can be reached from a node *i* by going along any simple directed path are *descendants* of node *i*. The root is said to be on level 0. Any other node *i* in *T* is at some level $l \in \{1, 2, ...\}$ which equals the length (number of edges) of the path from the root node to node *i*.

B. THE NEW BRANCH-AND-BOUND METHODOLOGY

We propose a completely new branch-and-bound algorithm for solving NDP, a procedure that is not based on an IP formulation and its linear-programming (LP) relaxation. Another unique feature of the methodology is that the associated enumeration tree is not a binary tree, but may have as many branches below a node as there are edges in an *s-t* cut. Our description below begins by considering a binary branching scheme, however.

Consider a typical, binary IP:

$$\text{(BIP)} \qquad z^* = \min_{\mathbf{x}} \mathbf{cx} \tag{1}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \tag{2}$$

$$\mathbf{x} \in \{0,1\}^J . \tag{3}$$

This is usually solved through its LP relaxation which replaces the integer value restrictions (3) with $0 \le \mathbf{x} \le 1$:

$$\text{(BIPR)} \qquad \underline{z} = \min_{\mathbf{x}} \mathbf{cx} \tag{4}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \tag{5}$$

$$0 \le \mathbf{x} \le 1 . \tag{6}$$

BIPR is solved for $\hat{\mathbf{x}}$. If the solution is integer, we have solved BIP; if not, we select some fractional $\hat{x}_j$ and "branch" to solve two problems:

$$\text{(BIPRa)} \qquad \underline{z}_a = \min_{\mathbf{x}} \mathbf{cx} \tag{7}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \tag{8}$$

$$0 \le \mathbf{x} \le 1 \tag{9}$$

$$x_j = 0, \tag{10}$$

and

$$\text{(BIPRb)} \qquad \underline{z}_b = \min_{\mathbf{x}} \mathbf{cx} \tag{11}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \tag{12}$$

$$0 \le \mathbf{x} \le 1 \tag{13}$$

$$x_j = 1 . \tag{14}$$

An enumeration tree is formed in which each problem has the same constraints and objective function as BIPR except for some additional bounds on certain components of **x**. This branching procedure is repeated recursively, using upper- and lower-bounding information, to create a convergent algorithm. The leaves of the tree represent subregions of the complete feasible region of BIP. Each subproblem can produce two more problems in the enumeration tree when we branch on one of the noninteger components of their solution. In order to find an optimal solution to BIP, the enumeration tree must be *monotonic*, i.e., the objective value of each subproblem in the tree is no better than that of any of its ancestors. The objective value for the best integer solution found so far, called the *incumbent solution*, is kept as an upper bound on z^*. Whenever a node is reached whose objective value is no better than the incumbent solution, the tree is pruned at that node. The tree can also be pruned at any node if an infeasible solution arises from all the restrictions that have been added. In this approach, we hope that we quickly find a good integer solution to BIP through one of its restricted LP relaxations.

Now, let us consider an analogous solution procedure for NDP. Assume that $E_D = \{e_d\}$; cases with more than one diversion edge will be discussed later. Let $\mathbf{C}'$ denote the set of minimal *s*-*t* cuts in *G*, let $\mathbf{C}''$ denote the set of non-minimal *s*-*t* cuts, and let $\mathbf{C} = \mathbf{C}' \cup \mathbf{C}''$. We state NDP through this formulation:

$$\text{(NDP)} \qquad z^* = \min_{C} \sum_{e \in C} w_e \tag{15}$$

$$\text{s.t. } e_d \in C \tag{16}$$

$$C \in \mathbf{C}' \tag{17}$$

This formulation simply says that we are looking for a minimum-weight, minimal *s*-*t* cut in G that contains the diversion edge e_d. If minimality of the optimal cut C^* is not enforced, it is possible for $C^* - e_d$ to be a cut and thus no *s*-*t* communication at all would be possible in $G - (C^* - e_d)$ (Curet 2001). We would have failed to solve the problem in this case.

We do propose, however, to solve NDP by considering a relaxation of the problem in which non-minimal cuts are allowed:

$$\text{(NDPR)} \qquad \underline{z} = \min_{C} \sum_{e \in C} w_e \tag{18}$$

$$\text{s.t. } e_d \in C \tag{19}$$

$$C \in \mathbf{C}' \cup \mathbf{C}'' \ (= \mathbf{C}) \tag{20}$$

We derive NDPR from NDP by removing the minimality restriction (17), similar to the relaxation of BIP to BIPR. We solve NDP using a branch-and-bound procedure based on NDPR, analogous to solving BIP through branch-and-bound with BIPR. However, the details of branching, bounding, and monotonicity do change.

We will be solving NDPR through its dual, a max-flow problem. Furthermore, in the primal problem, not all cuts of $\mathbf{C}''$ are feasible, and the actual set of feasible, non-minimal cuts will change as our branch-and-bound algorithm proceeds. However, we are always working with valid partitions of subsets of $\mathbf{C}''$, so the following discussion about the correctness of the branch-and-bound algorithm is essentially correct.

The graph G with diversion edge $e_d = (u,v)$ "quasi-included" is denoted $G \oplus e_d$. The edge e_d is *quasi-included* in a cut by adding two artificial, infinite-weight edges, one from *s* to *u* and one from *v* to *t*. Since any minimum-weight cut must intersect at least one of the edges in the path (s,u), (u,v), (v,t), and since any minimum-weight cut of interest must have finite weight, we are assured that e_d is an element of any minimum-weight cut in $G \oplus e_d$.

NDPR can be solved by modifying G into $G \oplus e_d$, and by then finding a min-weight cut C_0 in $G \oplus e_d$ through standard techniques. (That is, find a maximum *s*-*t* flow in $G \oplus e_d$ using weights w_e as edge capacities, and then identify a minimum cut, in $O(|E|)$ time, by finding a cut that is saturated by that maximum flow.) C_0 will also be a cut of G, and if it is a minimal cut in G, NDP is solved.

Unfortunately, the quasi-inclusion method modifies the graph G into $G \oplus e_d$ so that a minimal, min-weight cut in $G \oplus e_d$ may be a non-minimal cut in G (Balcioglu and Wood 2003). Thus, $C_0 = \{e_1, e_2, ..., e_k\}$ may not solve our problem. Now, consider partitioning the space of feasible solutions based on a particular edge e_a. It will be convenient to select e_a from the initial non-minimal cut identified, i.e., $e_a \in C_0 - e_d$. In this case, we create subproblems analogous to BIPRa and BIPRb by alternately including or excluding e_a from the initial non-minimal cut:

(NDPRa)

$$\underline{z}_a = \min_C \sum_{e \in C} w_e \tag{25}$$

$$\text{s.t. } e_d \in C \tag{26}$$

$$C \in \mathbf{C}' \cup \mathbf{C}'' \tag{27}$$

$$e_a \in C \quad , \tag{28}$$

and

(NDPRb)

$$\underline{z}_b = \min_C \sum_{e \in C} w_e \tag{21}$$

$$\text{s.t. } e_d \in C \tag{22}$$

$$C \in \mathbf{C}' \cup \mathbf{C}'' \tag{23}$$

$$e_a \notin C \quad . \tag{24}$$

NDPRa and NDPRb represent subproblems in which e_a is quasi-included or excluded, respectively.

NDPRa can be solved using $(G \oplus e_d) \oplus e_a$. In this case, a new cut $\hat{C}$ in $(G \oplus e_d) \oplus e_a$, which is a solution to NDPRa, is found by applying the max-flow min-cut theorem.

NDPRb can be solved by "excluding" e_a from all cuts of $G \oplus e_d$. The graph $G \oplus e_d$ with edge e_a excluded is denoted $(G \oplus e_d) \odot e_a$. The edge e_a is *excluded* from all cuts by setting its weight to infinity. No relevant minimum-weight *s*-*t* cut contains e_a because the solution to NDP must be a cut with finite weight. In this case, a new cut $\hat{C}$ in $(G \oplus e_d) \odot e_a$, which is a solution to NDPRb, is found by applying the max-flow min-cut theorem.

NDPRa and NDPRb illustrate a general technique, using quasi-inclusion and exclusion, to partition the solution space of NDPR. A convergent branch-and-bound algorithm can be created through repeated applications of the partitioning scheme analogous to a branch-and-bound algorithm to solve BIP. Each subsequent node in the enumeration tree corresponds to a restriction (quasi-inclusion or exclusion of an edge) of the problem above it, and thus we maintain monotonicity of the objective function, which is required for branch-and-bound. Monotonicity also derives from the fact that the maximum flow in G, which equals the capacity of the cut identified at each node, is never decreasing, because quasi-inclusion adds infinite-weight edges and exclusion increases the capacity on an edge.

The enumeration tree formed by using this partitioning scheme leads to the problem of repeatedly generating the same cut. NDPRa gives back the same solution as NDPR because the edge e_a is already an edge of C_0. We use a generalized, non-binary partitioning scheme to avoid this.

Every node in the enumeration tree obtained in the generalized partitioning scheme will have as many subproblems as there are edges in the current cut. All the subproblems below the current non-minimal cut $C_0 = \{e_1, e_2, ..., e_k\}$ are derived by establishing a partition of the solution space of cuts **C** through the relationship

$$\begin{aligned} \mathbf{C} = &[\mathbf{C} \cap \bar{e}_1] \cup [\mathbf{C} \cap e_1 \cap \bar{e}_2] \cup [\mathbf{C} \cap e_1 \cap e_2 \cap \bar{e}_3] \\ &\cup ... \cup [\mathbf{C} \cap e_1 \cap e_2 \cap ... \cap e_{k-1} \cap \bar{e}_k] \\ &\cup [\mathbf{C} \cap e_1 \cap ... \cap e_k]. \end{aligned} \tag{25}$$

Here, $[\mathbf{C} \cap e_1 \cap e_2 \cap ... \cap e_{k-1} \cap \overline{e}_k]$ denotes the set of all cuts that include e_1 through e_{k-1}, and exclude e_k, which is implemented through $G \oplus \{e_1,...,e_{k-1}\} \odot e_k$.

An optimal solution to NDP must occur in one of the subsets of the partition, although it cannot occur in $[\mathbf{C} \cap e_1 \cap ... \cap e_k]$, because we have already established that any cut in this set must be non-minimal. (C_0 is non-minimal and any cut in this subset is a superset of C_0 and can thus be ignored.) In other words, including all the edges e_1 through e_k in a new cut leads to repeated generation of the same cut which we know to be non-minimal.

So, in this instance, our branch-and-bound algorithm will create *k* branches by forcing quasi-inclusion and exclusion of the indicated edges from the initial cut. When a minimum-weight, minimal *s*-*t* cut contains e_d is found, we can use its weight as a bound to trim branches of the tree in the usual manner. A branch is also trimmed at any node if an infeasible solution arises from all the edges that have been included in or excluded from a current node, i.e., an infinite maximum flow is detected. Note that in solving an IP by LP-based branch-and-bound, we allow fractional solutions and "hope for" integrality. In our procedure for solving NDP, we allow non-minimality and hope for minimality.

THIS PAGE INTENTIONALLY LEFT BLANK

III. IMPLEMENTATION

This chapter introduces a simple implementation of the new branch-and-bound algorithm for solving NDP, along with a pseudo-code description of the algorithm and its demonstration on a sample graph. After establishing the correctness of this simple implementation, a generalized implementation is stated with a number of refinements. Finally, some pathological cases and approaches to their solutions are discussed.

A. NEW BRANCH-AND-BOUND ALGORITHM

This section describes how the new branch-and-bound methodology will be implemented to solve NDP. We assume that $E_D = \{e_d\}$ and that we are given a directed network. Note that an undirected network can be transformed to a directed network by replacing each undirected edge by two directed anti-parallel edges whose weights are identical to the weight of the undirected edge (Ahuja et al. 1993, pp. 39).

Algorithm C (Figure 1) to solve NDP is based on the near-min-cut enumeration procedure, Algorithm B, in Balcioglu and Wood (2003). It is stated in simple terms for clarity; we discuss a number of practical enhancements later in this chapter.

Algorithm C

INPUT: A directed graph $G = (V, E)$, $s, t \in V$, diversion edge e_d, and edge weights $\mathbf{w} > \mathbf{0}$ except that $w_{e_d} = 0$.

OUTPUT: A min-weight, minimal *s*-*t* cut $\hat{C}$ containing e_d, i.e., a solution to NDP.

GLOBAL VARIABLES: $\bar{z} \leftarrow \infty$; $\hat{C} \leftarrow \varnothing$;

begin

$E^+ \leftarrow \{e_d\}$; /* set of edges to be included */

$E^- \leftarrow \varnothing$; /* set of edges to be excluded */

EnumerateC(G, s, t, $\mathbf{w}$, E^+, E^-);

print $\hat{C}$; /* If $\hat{C} = \varnothing$, the problem is infeasible */

end.

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

begin

$\mathbf{w}' \leftarrow \mathbf{w}$; $G' \leftarrow G$;

for (each edge $e = (u,v) \in E^-$) $w'(u,v) \leftarrow \infty$;

for (each edge $e = (u,v) \in E^+$) **begin**

add artificial edge (s,u) to G' and let $w'(s,u) \leftarrow \infty$;

add artificial edge (v,t) to G' and let $w'(v,t) \leftarrow \infty$;

endfor;

/* G' and $\mathbf{w}'$ are now interpreted to include artificial edges */

/* *MaxFlow*() below finds a min-weight *s*-*t* cut C_0 in G', and its weight z, but subject to edges E^- being excluded from the cut, and edges E^+ being quasi-included. It uses a maximum-flow algorithm to do this. */

$[z, C_0] \leftarrow MaxFlow(G', s, t, \mathbf{w}')$;

if ($z \geq \bar{z}$) **return**;

if (C_0 is a minimal *s*-*t* cut in G) **then**

$\hat{C} \leftarrow C_0$; $\bar{z} \leftarrow z$;

return;

endif;

for (each edge $e \in C_0 \setminus E^+$) **begin**

$E^- \leftarrow E^- \cup \{e\}$;

EnumerateC(G, s, t, $\mathbf{w}$, E^+, E^-);

if ($z \geq \bar{z}$) **return**;

$E^- \leftarrow E^- \setminus \{e\}$; $E^+ \leftarrow E^+ \cup \{e\}$;

endfor;

return;

end.

Figure 1. Algorithm C: A branch-and-bound algorithm to solve the network diversion problem when $E_D = \{e_d\}$.

The algorithm starts by assuming no initial solution, i.e., $\hat{C} = \varnothing$ and $\bar{z} = \infty$. The diversion edge e_d is added to E^+ because any solution must contain e_d. Then, the algorithm calls the procedure *EnumerateC* which attempts to find a min-weight *s*-*t* cut containing e_d.

The procedure *EnumerateC* first modifies G into G' by changing the weight of each edge being excluded to infinity and by adding two artificial, infinite-weight edges for each edge being quasi-included. Then, it finds an initial min-weight *s*-*t* cut C_0 in G'; its weight z is a local lower bound, i.e., a lower bound subject to the restrictions defined through E^+ and E^-. C_0 is also an *s*-*t* cut in *G*. If the local lower bound z is greater than upper bound $\bar{z}$, the algorithm backtracks because no solution better than the incumbent can be found given the current restrictions (or any superset of them which would occur below in the tree). If a minimal *s*-*t* cut whose weight is less than upper bound $\bar{z}$ is found, i.e., a feasible solution to NDP is found, then this cut is kept as an incumbent solution, and $\bar{z}$ is updated. The algorithm also backtracks in this case because the weight of any other *s*-*t* cut below the current cut in the monotonic enumeration tree can be no better than the incumbent solution. In other cases, the algorithm calls itself recursively, creating a branch for each edge in $C_0 \setminus E^+$, i.e., a branch for each edge in the current cut yet fixed (forced in). For each branch, it finds a new min-weight *s*-*t* cut that contains the edges being quasi-included and does not contain the edges being excluded according to the generalized partitioning scheme. The algorithm terminates when all of the branches are pruned through bounding arguments or through infeasibility. The algorithm is clearly finite since the depth of the enumeration tree is at most $|E|$.

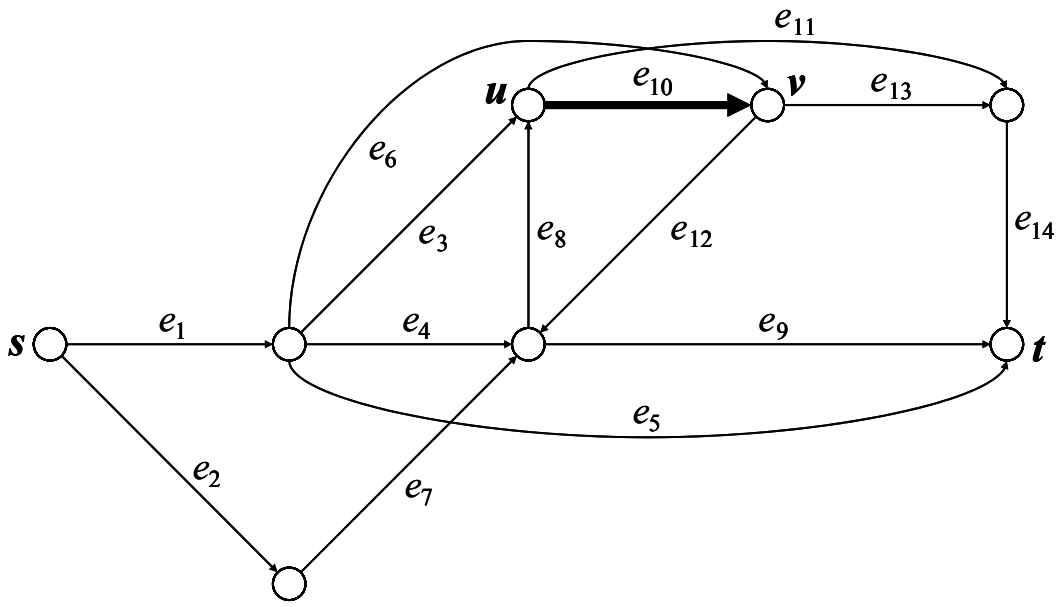

Figure 2. Sample Graph. The diversion edge is $e_{10} = (u, v)$. All edge weights are 1 except that $w_{e_{10}} = 0$.

To illustrate Algorithm C, consider the enumeration tree in Figure 3, which corresponds to solving NDP on the graph of Figure 2. The algorithm first finds an initial min-weight *s*-*t* cut $\{e_1, e_2, e_{10}, e_{11}\}$ (node 1) in $G \oplus e_d = G \oplus e_{10}$ at the root node at level 0. This is a non-minimal cut because it contains the minimal cut $\{e_1, e_2\}$. Thus, the algorithm must partition the solution space of cuts based on the edges $\{e_1, e_2, e_{11}\}$.

After partitioning, the algorithm finds a minimal cut at node 2 $\{e_5, e_6, e_9, e_{10}, e_{11}\}$ in its first branch marked by $\overline{e}_1$. It keeps this cut as an incumbent solution, sets $\overline{z} = 4$ and backtracks (as it always does when it finds a feasible solution). The second branch, $(e_1, \overline{e}_2)$, yields a non-minimal cut $\{e_1, e_7, e_{10}, e_{11}\}$ (node 3). The local lower bound is $z = 3$ so the algorithm cannot backtrack. At this node then, it branches to $\overline{e}_7$ and finds a minimal cut (node 4) better than the incumbent. It keeps this cut as the incumbent solution and backtracks while setting $\overline{z} = 3$. Nodes 5 and 6 are pruned by bounding, without exploration, because the corresponding local lower bounds cannot be better than $\overline{z}$, i.e., no recursive call is made for nodes 5 and 6 because of the third return statement. Finally, the algorithm terminates with the optimal solution $C^* = \{e_1, e_9, e_{10}, e_{11}\}$.

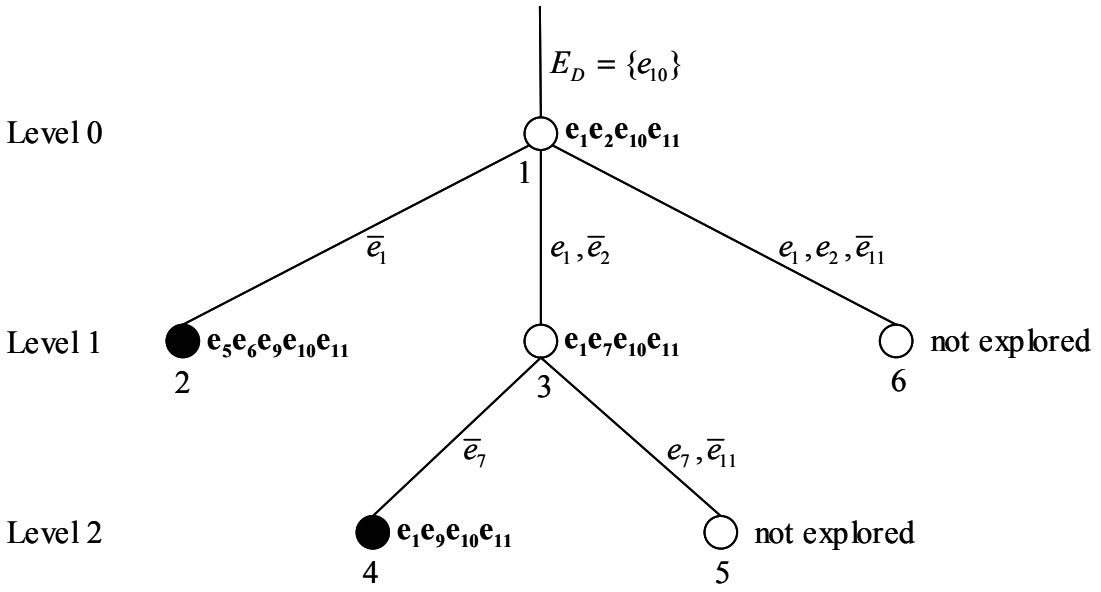

Figure 3. Enumeration tree for the graph of Figure 2.

B. CORRECTNESS OF THE ALGORITHM

To state that Algorithm C correctly solves the Network Diversion Problem, it suffices to show that (a) the algorithm would enumerate all feasible cuts containing the specified diversion edge if backtracking occurs only by infeasibility, and (b) the enumeration tree maintains monotonicity of the objective function.

Balcioglu and Wood (2003) give a proof that all *s*-*t* cuts in a graph can be found by using this partitioning scheme with an initial min-weight *s*-*t* cut. In our methodology, we begin to partition by quasi-including the diversion at the root node of the enumeration tree; but this is the same technique that Balcioglu and Wood (2003) use in their enumeration algorithm, to force inclusion of edges. Hence, Algorithm C with no "value-based backtracking" must enumerate all minimal cuts containing e_d.

When the solution space is partitioned recursively, the weight of an edge in G has been increased to infinity or two infinite-weight edges have been added to G at each node of the enumeration tree. The weight of a cut below a current node never decreases because this process never reduces the maximum flow in *G*. Therefore, we conclude that the enumeration tree is monotonic.

C. REFINEMENTS TO ALGORITHM C

Algorithm C can be refined in a number of ways for improved efficiency and greater generality:

1. Algorithm C, as stated above, repeats much work for the sake of clarity, but the actual implementation avoids this for efficiency. For instance, the implementation does not modify the original graph from scratch at each node of the enumeration tree (by using E^+ and E^-). Rather, once an edge is quasi-included or excluded from all cuts at some node in the enumeration tree, it is automatically quasi-included or excluded from all descendant-node problems, with no additional work required.

2. A max flow in a parent-node graph is always feasible for all its child-node graphs. This makes identification of a min-cut in the child nodes significantly faster, through a small number of flow augmentations, because we do not have to identify a max flow and thus a min cut from scratch.

3. Most branch-and-bound algorithms can be improved by adding a heuristic that finds a feasible initial solution so that the algorithm begins with $\bar{z} < \infty$. Nodes of the enumeration tree can often be trimmed quickly this way, rather than having to wait for the algorithm to stumble upon a feasible solution and corresponding finite upper bound. The problem of finding a minimal cut containing a specific edge is an NP-complete problem (Caryle and Wood 2002). Thus, the following heuristic is not guaranteed to work (Curet 2001); in practice it usually does, however. Let $e_d = (u, v)$:

   a. Using breadth-first search (BFS), find a shortest path (minimum number of edges) from *s* to *u* that traverses neither *v* nor *t*. Let E_1 be the set of edges in that path.
   b. Using BFS, attempt to find a shortest path from *v* to *t* that does not use any vertex on the first shortest path. If such a path is found, let its edges be denoted E_2.
   c. If two paths can be found, a min-weight cut in $G \odot (E_1 \cup E_2)$ will be a minimal cut containing e_d (Curet 2001).

We implement the heuristic as a subroutine *TwoPathsHeuristic*(*G, s, u, v, t,* **w**) which returns a finite upper bound $\bar{z}$ if successful, and returns ∞ otherwise. Figure 4 gives an example.

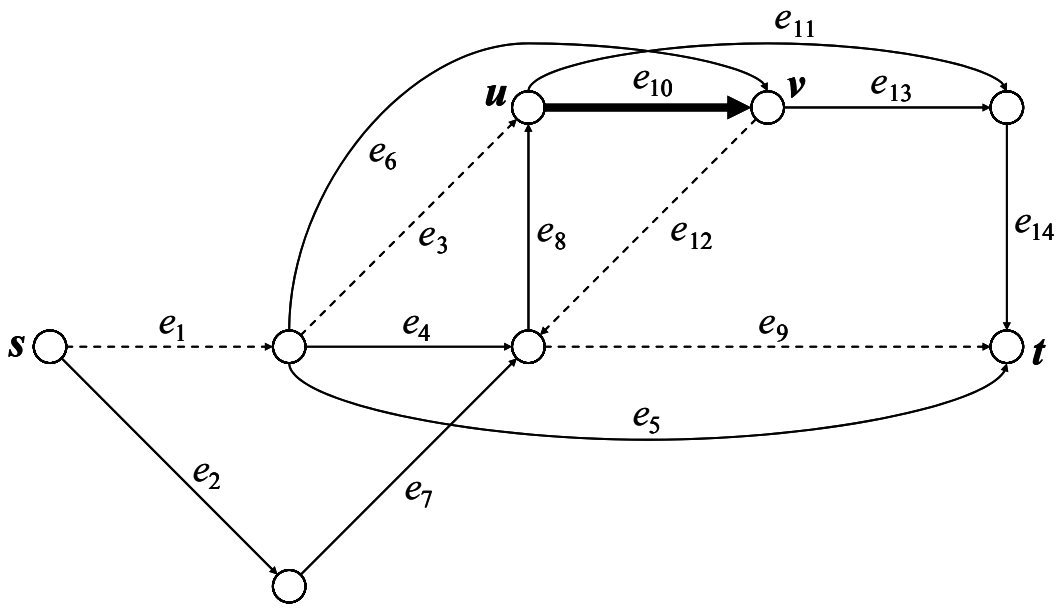

Figure 4. Illustrating the *TwoPathsHeuristic*() to find a feasible solution for NDP. A shortest *s*-*u* path is first identified as $E_1 = \{e_1, e_3\}$. Then, a shortest *v*-*t* path, which does not use any vertex on the first shortest path, is identified as $E_2 = \{e_{12}, e_9\}$. Finally, a minimal cut $C' = \{e_2, e_4, e_5, e_6, e_{10}, e_{11}\}$ is computed in $G \odot (E_1 \cup E_2)$, and $w(C') = 5$ is kept as an initial upper bound for NDP when $E_D = \{e_{10}\}$.

4. We may wish to solve NDP when $E_D = \{e_1, ..., e_p\}$, i.e., when $|E_D| > 1$. This is easily accomplished by running Algorithm C with $e_d = e_1$, then with $e_d = e_2$, etc., and taking the best solution from among the p solutions. However, it is clear that we can improve that procedure by using bounding information and partial solutions from one subproblem to the next. Furthermore, if $|E_D|$ is large, it may be worthwhile to first find a min-weight cut C' in G and simply check to see if some diversion edge is contained in that cut. (Recall that $w_e = 0$ for all $e \in E_D$, and all other edges have positive weights. Thus, if $|E_D|$ is large, it seems intuitively likely that one of the edges in E_D will turn up

in a min-weight cut.) If it does, NDP has been solved by just a single min-cut (max-flow) calculation. If not, the flow just found is still a useful point from which to begin the algorithm.

The improved and generalized implementation of Algorithm C to solve NDP when $|E_D| > 1$ is shown in Figure 5.

Algorithm C
INPUT: A directed graph $G = (V, E)$, $s, t \in V$, diversion edges E_D, and edge weights $\mathbf{w} > \mathbf{0}$ except that $w = 0$ for all $e \in E_D$.
OUTPUT: A min-weight, minimal *s*-*t* cut containing at least one $e_d \in E_D$, i.e., a solution to NDP.
GLOBAL VARIABLES: $\overline{z} \leftarrow \infty$; $\hat{C} \leftarrow \varnothing$;
begin
$\left[w(C_0), C_0\right] \leftarrow MaxFlow(G, s, t, \mathbf{w})$
if ($C_0 \cap E_D \neq \varnothing$) **then** print C_0 and halt;
for (each edge $e \in E_D$**) begin**
$E^+ \leftarrow \{e_d\}$;
$u \leftarrow$ tail vertex of e_d; $v \leftarrow$ head vertex of e_d;
$[z, C_0] \leftarrow TwoPathsHeuristic(G, s, u, v, t, \mathbf{w})$;
if ($z < \overline{z}$) **then**
$\overline{z} \leftarrow z$; $\hat{C} \leftarrow C_0$;
endif;
endfor;
for (each edge $e \in E_D$**) begin**
$E^+ \leftarrow \{e_d\}$; /* set of edges to be included */
$E^- \leftarrow \varnothing$; /* set of edges to be excluded */
EnumerateC(G, s, t, $\mathbf{w}$, E^+, E^-);
endfor ;
print $\hat{C}$**;** /* If $\hat{C} = \varnothing$, the problem is infeasible */
end.

Procedure *EnumerateC*(G, s, t, $\mathbf{w}$, E^+, E^-)
begin
Same as in Figure 1 ;
end.

Figure 5. An improved and generalized version of Algorithm C for $|E_D| > 1$.

D. PATHOLOGICAL CASES

We call an edge that is not contained in any simple *s*-*t* path in G an *extraneous edge*. A pathological case arises when all diversion edges are extraneous edges for NDP: In this case, a solution to NDP does not exist but Algorithm C "solves" the problem correctly and returns $\hat{C} = \varnothing$. However, the algorithm may perform a huge number of iterations in such cases. Pathological cases cannot occur, however, if the extraneous edges in a network are identified and marked as "ineligible" to be diversion edges.

Before examining pathological cases, we need to define some concepts associated with undirected graphs. Any vertex in a connected undirected graph whose removal results in a disconnected graph is called a *separation vertex*. A graph containing a separation vertex is called *separable*, and a graph that does not contain such a vertex is called *nonseparable*. A subgraph G' of G that contains no separation vertices is a *nonseparable component* of the original graph.

Even (1979, pp. 57-62) gives an algorithm, based on depth-first search (DFS), for finding all the nonseparable components and the separation vertices of a given graph in $O(|E|)$ time. Consider the separable graph G in Figure 6 to find its nonseparable components.

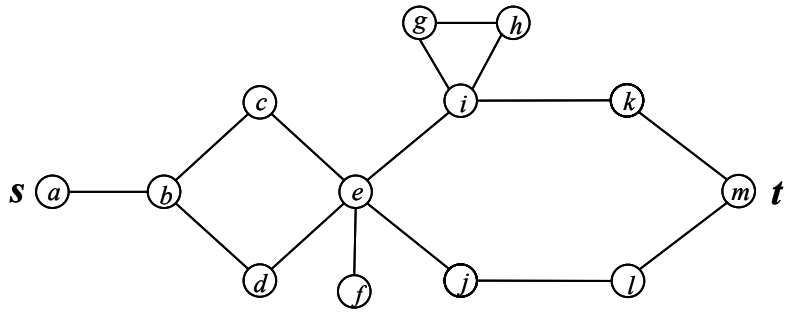

Figure 6. A separable graph G with separation vertices *b*, *e*, and *i*.

First, we modify G into $G + (s,t)$ by adding an artificial edge from *s* to *t* in order to keep *s* and *t* in the same nonseparable component. See Figure 7. The nonseparable components (Figure 8) are then found with Even's algorithm.

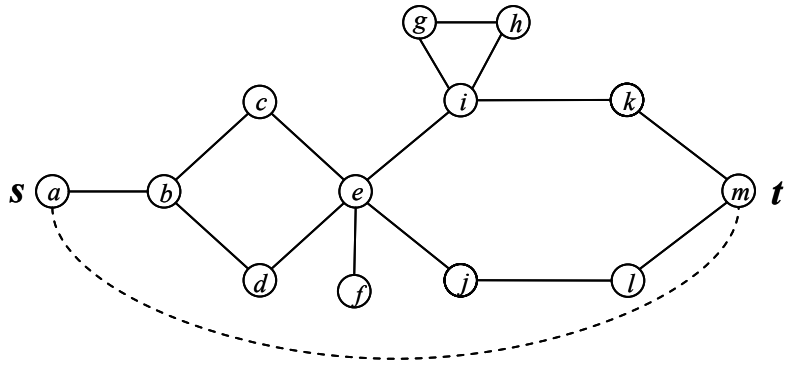

Figure 7. The graph of Figure 6 modified to $G + (s,t)$.

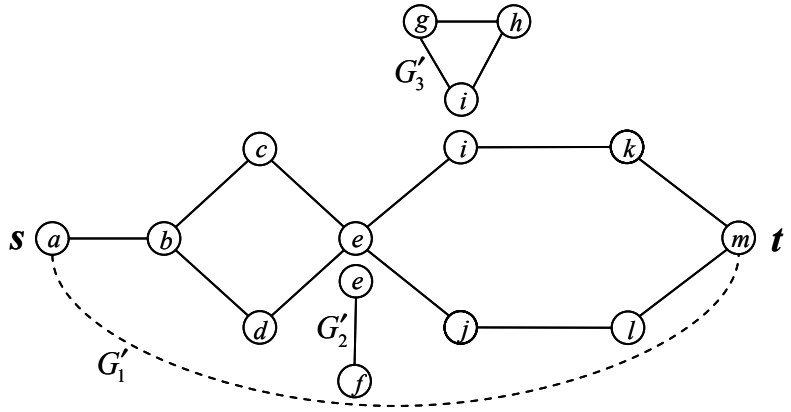

Figure 8. The nonseparable components of the graph $G + (s,t)$ from Figure 7.

The extraneous edges in the graph can now be determined easily: All edges of any separable component of $G + (s,t)$, except for the component containing both *s* and *t*, are extraneous. For instance, the nonseparable components G_2' and G_3' of $G + (s,t)$ do not contain both *s* and *t*. Therefore, the edges (e,f), (i,g), (g,h), and (h,i) from G_2' and G_3' are extraneous, and are not allowed to be diversion edges. Thus, we are effectively operating on the reduced graph G'' of Figure 9.

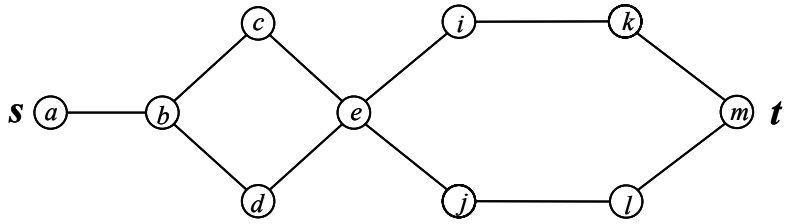

Figure 9. The reduced graph G'' after nonseparable decomposition. The artificial edge (s,t) is removed because it is not useful any more.

The nonseparable decomposition process applied to undirected graphs is guaranteed to identify all extraneous edges, and this process is also valid for finding extraneous edges of a directed graph if it is treated as being undirected. But the nonseparable decomposition process may not identify all extraneous edges in a directed graph. In fact, there is no known polynomial-time algorithm to detect all extraneous edges in a directed graph (Fortune, Hopcroft and Wyllie 1980). This is the same NP-complete problem as finding a simple directed path from *s* to *t* that contains a specified edge. Therefore, extraneous edges in large directed graphs may cause Algorithm C some difficulties.

Our algorithm does not include a nonseparable decomposition procedure because the networks that we test do not contain any nonseparable components. But it may be worthwhile to implement this process because many real-world networks may contain extraneous edges.

If an undirected network is being analyzed, we do not know in which direction an interdiction edge $e_d = (u,v)$ should appear in the optimal solution. That is, we do not know if all *s*-*t* paths in $G - C^* + (u,v)$ traverse from *u* to *v* or from *v* to *u*. We can solve the problem, in theory, by quasi-including the directed edge $e_d = (u,v)$ and running the algorithm, then quasi-including the directed edge $e_d = (v,u)$ and running the algorithm again, and then selecting the best of the two solutions as the optimal solution to NDP. However, it is possible that the "optimal solution" for one of the orientations is $C^* = \varnothing$. In this case, the algorithm may fail to find an optimal solution in a reasonable amount of time for large networks. There is a simple way to deal with this issue.

Recall that an undirected edge is represented by two anti-parallel directed edges in Algorithm C. Figure 10 shows the representation of G'' from Figure 9, which would be used in Algorithm C.

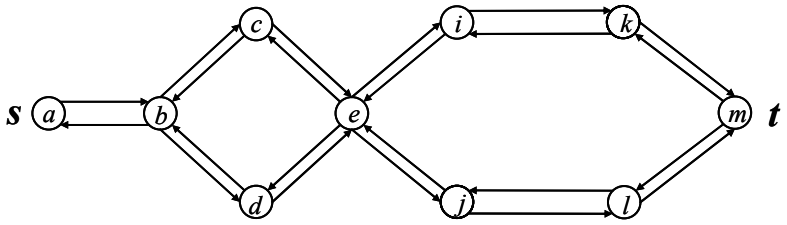

Figure 10. The directed representation of G'' (Figure 9) used in Algorithm C.

Let us refer to the undirected edge (i,k) in G' as $\{i,k\}$. The flow from *s* to *t* in the directed representation of G'' travels over either (i,k) or (k,i). The removal of either (i,k) or (k,i) represents that the edge $\{i,k\}$ in G'' is in a cut. If the edge (i,k) is chosen to represent the diversion edge, then an optimum solution exists for NDP. On the other hand, if the edge (k,i) is chosen, then no feasible solution is obtained to NDP: Algorithm C might fail to return the answer $C^* = \varnothing$ in a reasonable amount of time for larger networks with similar structure.

To avoid this, we replace the edge $\{i,k\}$ with a directed Wheatstone bridge as shown in Figure 11. The edge (v_1,v_2) that takes place of the edge $\{i,k\}$ with identical weight is assigned as the diversion edge, i.e., two artificial, infinite-weight edges (s,v_1) and (v_2,t) are added to the directed representation of G''. In this case, all finite-weight cuts in the new graph must contain the edge (v_1,v_2). Because of that, we conclude the edge $\{i,k\}$ always occurs in a cut oriented in whichever direction is appropriate.

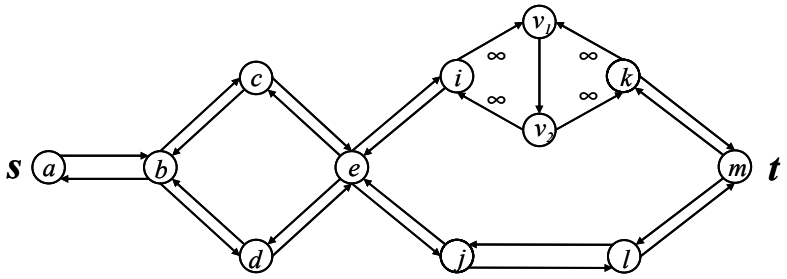

Figure 11. Representation of the diversion edge $\{i,k\}$ in G'' using a Wheatstone bridge.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. COMPUTATIONAL RESULTS

This chapter provides computational results for Algorithm C on a set of structured test problems. The algorithm is written using the Java 1.4 programming language. All tests are performed on a personal computer with a 1.5 GHz Pentium IV processor, 384 MB of RAM and running under the Windows XP operating system.

A. TEST NETWORKS

Topology is the physical configuration of connections comprising a network, i.e., it is the shape of the network. We use two common topologies in communications networks to test Algorithm C. These topologies ensure high reliability because of their high.

1. Grid Networks

A grid topology consists of a network where the vertices may be viewed as being points on the $\mathrm{L}\times\mathrm{H}$ integer lattice in the non-negative quadrant. Each vertex is connected to other vertices, which are positioned in the lattice directly to its north, south, east, and west. There are two additional vertices, a source and a sink vertex, which are positioned at the west and east side of the network, respectively. All westernmost vertices are connected to the source vertex, and all easternmost vertices are connected to the sink vertex. The number of vertices in a grid network is $|V|=\mathrm{H}\times\mathrm{L}+2$, and the number of edges is $|E|=4\times\mathrm{H}\times\mathrm{L}-2\times\mathrm{L}$. Edges incident to *s* and *t* have infinite weights and cannot be interdicted and cannot be diversion edges. Figure 12 shows an example of a small grid network.

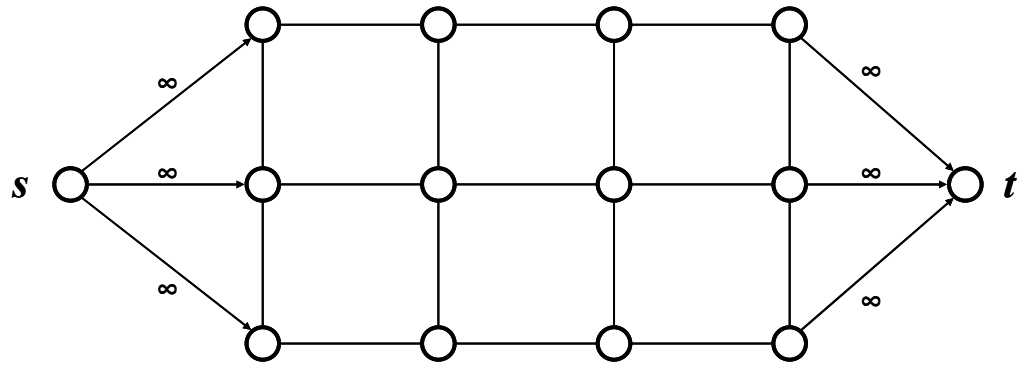

Figure 12. An unweighted, undirected grid network with $\mathrm{H}=3$ and $\mathrm{L}=4$. The undirected edges between two vertices represent two anti-parallel directed edges, and their weights are 1.

A grid topology, or something close to it, is often used in Wide Area Networks (WANs) where reliability is important (Harris 1999).

1. Star-Mesh Networks

A star-mesh network (Miller 2001) is created if H "rays" of L vertices, each connected by an additional center vertex, are cross-connected as well as radially connected. The source vertex *s* is the center vertex. The sink vertex *t* is chosen from the vertices which are positioned at the outer most ring of the network to increase the distance between *s* and *t* as much as possible. The number of vertices in a star-mesh network is $|V|=\mathrm{H}\times\mathrm{L}+1$, and the number of edges is $|E|=4\times\mathrm{H}\times\mathrm{L}$. Figure 13 shows an example of a star-mesh network. Star-mesh networks are often used in local area networks (LANs) (Sharma 1990, pp. 9-12) where the central vertex controls all the vertices in the network.

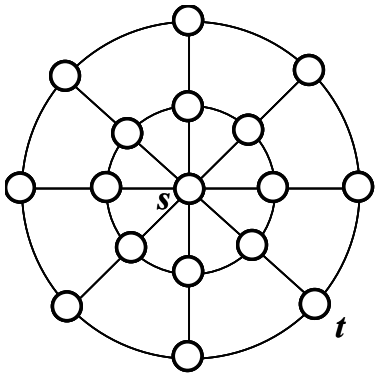

Figure 13. An undirected star-mesh network with $\mathrm{H}=8$ and $\mathrm{L}=2$.

B. RESULTS

We use a grid network generator (GGFGEN) written in Java by Balcioglu and Wood (2003) to generate grid networks. We have coded a star-mesh network generator (STARGEN) in Java to generate star-mesh networks. The input parameters of the generators, H and L, determine the size of the generated network.

We test Algorithm C on unweighted and weighted networks separately. A network is "unweighted" if all interdictable edges e have $w_e = 1$. In our weighted networks, w_e for each interdictable edge is randomly drawn from the discrete uniform distribution on $[1,5]$. For both weighted and unweighted grid networks, the edges incident to *s* and *t* have infinite weights.

The sizes of networks generated are systematically increased to observe the performance of the algorithm. Each network is also tested as $|E_D|$ increases; the set E_D is chosen randomly from E.

1. Unweighted Networks

Table 1 presents problem and solution statistics for Algorithm C applied to solving NDP in unweighted, undirected grid networks. For all networks in the table, the algorithm gives an optimal solution in less than one second: The *TwoPathHeuristic*() always finds the optimal solution, and this solution is proven optimal at the root node of the enumeration tree. Indeed, with some thought about the structure of these problems, we can see that they can be solved by inspection.

Table 2 presents processing times on unweighted, undirected star-mesh networks. In this network type, processing times, upper bounds, and optimal solutions change suddenly depending on the diversion edges and sink vertex which are chosen randomly. Increased numbers of non-minimal cuts in the enumeration tree imply longer processing times. In fact, some of these problems, and the weighted problems for star-mesh networks, cannot be solved to optimality in a reasonable amount of time. In these cases, we have identified ε-optimal solutions, for rather large values of $\varepsilon > 0$, by replacing both instances of "if $(z \geq \bar{z})$ then" with "if $(z\ (1 + \varepsilon) \geq \bar{z})$ then."

Network	$\|V\|$	$\|E\|$	$\|E_D\|$	zUB	zLB	$w(C^*)$	Number Non-minimal Cuts	Number Minimal Cuts	Total Time (sec.)
GRIDNET 10×10	102	380	1	13	13	13	0	0	0.00
			2	9	9	9	0	0	0.02
			3	9	9	9	0	0	0.02
			4	10	10	10	0	0	0.02
			5	10	10	10	0	0	0.02
			10	9	9	9	0	0	0.02
GRIDNET 20×20	402	1560	1	19	19	19	0	0	0.02
			2	19	19	19	0	0	0.03
			3	19	19	19	0	0	0.03
			4	19	19	19	0	0	0.05
			5	19	19	19	0	0	0.05
			10	19	19	19	0	0	0.06
GRIDNET 30×30	902	3540	1	30	30	30	0	0	0.03
			2	30	30	30	0	0	0.06
			3	29	29	29	0	0	0.06
			4	29	29	29	0	0	0.06
			5	29	29	29	0	0	0.06
			10	29	29	29	0	0	0.08
GRIDNET 40×40	1602	6320	1	39	39	39	0	0	0.08
			2	39	39	39	0	0	0.09
			3	40	40	40	0	0	0.09
			4	39	39	39	0	0	0.11
			5	39	39	39	0	0	0.11
			10	39	39	39	0	0	0.16
GRIDNET 50×50	2502	9900	1	49	49	49	0	0	0.13
			2	50	50	50	0	0	0.13
			3	49	49	49	0	0	0.14
			4	49	49	49	0	0	0.16
			5	49	49	49	0	0	0.17
			10	49	49	49	0	0	0.19

Table 1. Problem statistics and computational results for Algorithm C on unweighted, undirected grid networks. GRIDNET H×L denotes a grid network with H rows of L vertices. zUB is the upper bound obtained from the *TwoPathsHeuristic*() procedure. zLB shows the weight of the cut at the root node of the enumeration tree. The columns "Number Non-minimal Cuts" and "Number Minimal Cuts" give the number of non-minimal cuts and minimal cuts in the enumeration tree, respectively. Here, the algorithm does not need to perform any enumeration, and therefore these entries are all zero. The total number of nodes in the enumeration tree is the sum of these numbers plus one.

Network	$\|V\|$	$\|E\|$	$\|E_D\|$	zUB	zLB	$w(C^*)$	Number Non-minimal Cuts	Number Minimal Cuts	Total Time (sec.)
STARNET 10×10	101	400	1	6	6	6	0	0	0.00
			2	8	6	8	6	0	0.02
			3	11	6	11	234	0	0.08
			4	7	6	7	4	0	0.02
			5	10	6	10	132	0	0.05
			10	4	4	4	0	0	0.02
STARNET 20×20	401	1600	1	2	2	2	0	0	0.02
			2	10	6	10	74	0	0.17
			3	14	6	14	3963	0	6.45
			4	17	6	17	110608	0	197.30
			5	13	6	13	3456	0	6.16
			10	9	6	9	107	0	0.25
STARNET 30×30	901	3600	1	29	6	29 (0.75)*	27449	0	170.39
			2	11	6	11	163	0	0.81
			3	27	6	27 (0.50)*	210612	0	1068.39
			4	29	6	16 (0.75)*	83083	1	455.78
			5	16	6	16	56305	0	298.16
			10	23	6	23 (0.25)*	102270	0	551.97
STARNET 40×40	1601	6400	1	39	6	39 (1.25)*	70205	0	710.28
			2	10	6	10	74	0	0.87
			3	21	6	21 (0.25)*	113284	0	1173.43
			4	30	6	30 (1.00)*	15569	0	177.23
			5	21	5	21 (0.50)*	195531	0	2050.08
			10	23	6	23 (0.75)*	14551	0	154.69
STARNET 50×50	2501	10000	1	26	6	26 (0.50)*	3960	0	74.22
			2	39	6	39 (1.50)*	20212	0	344.90
			3	21	6	21 (0.50)*	4960	0	84.83
			4	28	6	28 (1.00)*	5330	0	91.27
			5	49	6	49 (2.50)*	7190	0	127.28
			10	3	3	3	0	0	0.84

***** This problem could not be solved to optimality in a reasonable amount of time, so an ε-optimal solution is given. The value of ε is given in parentheses.

Table 2. Problem statistics and computational results for Algorithm C on unweighted, undirected star-mesh networks. STARNET H×L denotes a star-mesh network with H rays of L vertices each. The difference between zUB and zLB is much larger for the more difficult problems.

2. Weighted Networks

Tables 3 and 4 present problem and solution statistics on weighted, undirected grid and star-mesh networks, respectively. We use the same diversion sets on unweighted and weighted networks. Integer edge weights are generated randomly on $[1,5]$.

Network	$\|V\|$	$\|E\|$	$\|E_D\|$	zUB	zLB	$w(C^*)$	Number Non-minimal Cuts	Number Minimal Cuts	Total Time (sec.)
GRIDNET 10×10	102	380	1	38	28	35	146	1	0.06
			2	22	22	22	0	0	0.02
			3	20	20	20	0	0	0.02
			4	24	24	24	0	0	0.02
			5	26	26	26	0	0	0.03
			10	19	19	19	0	0	0.03
GRIDNET 20×20	402	1560	1	48	42	48	1913	0	4.38
			2	48	48	48	0	0	0.03
			3	48	43	48	635	0	1.48
			4	37	37	37	0	0	0.05
			5	37	37	37	0	0	0.06
			10	39	39	39	0	0	0.06
GRIDNET 30×30	902	3540	1	82	80	82	62	0	1.02
			2	79	77	79	48	0	0.83
			3	73	73	73	0	0	0.08
			4	75	75	75	0	0	0.09
			5	76	76	76	0	0	0.09
			10	73	73	73	0	0	0.11
GRIDNET 40×40	1602	6320	1	84	84	84	0	0	0.13
			2	89	89	89	0	0	0.14
			3	88	93	88	0	0	0.16
			4	90	89	90	8	0	0.81
			5	87	87	87	0	0	0.17
			10	85	85	85	0	0	0.20
GRIDNET 50×50	2502	9900	1	123	121	123	50	0	5.66
			2	122	121	122	28	0	3.05
			3	112	112	112	0	0	0.25
			4	112	112	112	0	0	0.25
			5	111	111	111	0	0	0.28
			10	119	117	118	3	1	0.84

Table 3. Problem statistics and computational results for Algorithm C on weighted, undirected grid networks. The problems that have some non-minimal cuts and no minimal cut show that the feasible solution (zUB) found in the *TwoPathsHeuristic*() is optimal. The problems with some non-minimal and minimal cuts show that the optimal solution is found in the enumeration tree.

Network	$\|V\|$	$\|E\|$	$\|E_D\|$	zUB	zLB	$w(C^*)$	Number Non-minimal Cuts	Number Minimal Cuts	Total Time (sec.)
STARNET 10×10	101	400	1	23	19	23	4	0	0.02
			2	35	21	32	131	1	0.06
			3	35	15	21	29	3	0.03
			4	12	12	12	0	0	0.02
			5	25	16	19	21	2	0.03
			10	9	9	9	0	0	0.02
STARNET 20×20	401	1600	1	13	13	13	0	0	0.02
			2	35	20	35	926	0	2.19
			3	29	14	29	405	0	0.91
			4	41	15	36	21139	3	47.09
			5	24	17	24	49	0	0.16
			10	28	14	25	675	2	1.42
STARNET 30×30	901	3600	1	69	21	69 (0.50)*	19068	0	137.75
			2	21	18	21	6	0	0.17
			3	66	24	66 (0.50)*	19001	0	153.12
			4	61	18	40 (0.50)*	27971	1	203.89
			5	38	16	38	8771	0	40.34
			10	56	16	56 (0.50)*	18711	0	135.36
STARNET 40×40	1601	6400	1	97	16	97 (1.25)*	19978	0	315.90
			2	21	16	21	9	0	0.33
			3	49	13	49 (0.25)*	62458	0	925.06
			4	87	11	87 (1.25)*	23710	0	376.97
			5	62	15	62 (0.50)*	53018	0	829.11
			10	50	10	50 (0.50)*	20310	0	315.41
STARNET 50×50	2501	10000	1	115	21	115 (1.50)*	8774	0	203.16
			2	118	17	118 (2.00)*	5090	0	127.12
			3	54	21	54 (0.25)*	26486	0	673.65
			4	84	16	84 (1.00)*	21363	0	513.34
			5	118	18	118 (2.00)*	13662	0	365.40
			10	57	15	57 (0.50)*	11930	0	322.52

* This problem could not be solved to optimality in a reasonable amount of time, so an ε-optimal solution is given. The value of ε is given in parentheses.

Table 4. Problem statistics and computational results for Algorithm C on weighted, undirected star-mesh networks.

C. INTERPRETATION

In the grid networks, the difference between the weight of the feasible solution found by *TwoPathHeuristic*() (upper bound) and the weight of the solution found at the root node of the enumeration tree (the minimum-weight cut in $G \oplus e_d$) is small so that the algorithm does not need to enumerate many cuts. In fact, $G \oplus e_d$ is likely to yield an optimal solution in the grid networks because *s* and *t* are highly connected.

In the star-mesh networks, the typical initial minimum-weight cut from $G \oplus e_d$ is the non-minimal cut that cuts edges incident to *t*, plus the diversion edge. But the optimal solution will usually have many more edges in it. There is a large gap between the upper bound and the initial cut. Because of this, the algorithm must enumerate a large number of cuts.

V. CONCLUSIONS AND RECOMMENDATIONS

The network diversion problem (NDP) arises when a set of links in a network must be removed using the least effort in order to force the user of the network to communicate or travel over a specified set of links between two designated vertices. This NP-complete problem may have intelligence-gathering and warfighting applications. In this thesis, we have developed and implemented a general branch-and-bound methodology to solve NDP.

NDP can be modeled as a binary integer problem (IP), but the literature indicates that these IPs are difficult to solve. Therefore, we introduce a new methodology, a general branch-and-bound approach, to solve NDP. The methodology is not based on solving linear-programming relaxations. Rather, the minimality of the cuts identified is relaxed. In this methodology, the algorithm first finds a minimum-weight *s*-*t* cut containing the diversion edge which is quasi-included. If this initial cut is non-minimal, then a monotonic enumeration tree is formed by forcing quasi-inclusion and exclusion of edges from the current cut. An edge (u,v) is quasi-included in a cut by adding two artificial, infinite-weight edges, one from *s* to *u* and one from *v* to *t*. An edge is excluded from a cut by setting its weight to infinity. The associated enumeration tree is non-binary since it may have as many branches below a node as there are edges in an *s*-*t* cut.

The algorithm has been improved by incorporating certain internal efficiencies, and by adding a heuristic to find a good initial feasible solution. When there is more than one diversion edge, the solution to the problem can be found by solving NDP for each diversion edge separately, and taking the best of those solutions. However, by using bounding information and partial solutions from one subproblem to the next, faster solutions are obtained. In fact, it is often easier to find a solution when the number of diversion edges is large.

We have tested our algorithm on two common communication-network topologies, grid and star-mesh topologies; with diversion edges chosen randomly. We increased the size of networks and the size of the diversion-edge set systematically to observe the performance of the algorithm. Grid networks were tested with up to 2502 vertices and 9900 edges (a 50 by 50 grid network). An optimal solution to NDP for all

sizes of grid networks is obtained in seconds. For example, in an unweighted grid network with 2502 vertices and 9900 edges (a) the algorithm returns an optimal solution in 0.125 seconds when $|E_D| = 1$, and (b) it takes only 0.172 seconds to get an optimal solution when $|E_D| = 5$.

We have also tested star-mesh networks with up to 2501 vertices and 10000 edges (a 50 by 50 star-mesh network). An optimal solution to NDP is obtained in a reasonable amount of time for networks as large as 20 by 20. Only some of the larger networks can be solved optimally.

Although the Algorithm C works efficiently for certain test networks, its overall run time and efficiency can be improved in different ways. For example, if the number of non-minimal cuts in an enumeration tree of a NDP can be decreased, the algorithm will be more efficient, and run faster. To accomplish this, the quasi-inclusion technique that creates non-minimal cuts should be investigated for improvement or replacement.

Another practical improvement for computation times on some large networks can no doubt be obtained by solving the initial maximum-flow problem using a more efficient algorithm, e.g., binary blocking flow algorithm (Goldberg and Rao 1998).

For problems like the star-mesh networks, we may wish to partition based on some set of edges other than those in the current minimum-weight cut. For instance, branching on edges in a path from the diversion edge to *s* or *t* may prove more effective.

LIST OF REFERENCES

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., (1993), *Network Flows*, Prentice Hall, New Jersey.

Balcioglu, A., and Wood, K., (2003), "Enumerating Near-Min s-t Cuts," in *Network Interdiction and Stochastic Integer Programming*, D. Woodruff, editor, Kluwer Academic Publishers, Boston, pp. 21-49.

Carlyle, M., and Wood, K., (2002), Interview between M. Carlyle and K. Wood Operations Research Department Naval Postgraduate School, Monterey, California, and the author, 10 August.

Curet, N. D., (2001), "The Network Diversion Problem," *Military Operations Research*, Vol. 6, pp. 35-44.

Even, S., (1979), *Graph Algorithms*, Computer Science Press, Potomac, Maryland.

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

Fortune S., Hopcroft J., and Wyllie J., (1980), "The Directed Subgraph Homeomorphism Problem," *Theoretical Computer Science*, Vol. 10, pp. 111-121.

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

Harris, M. P., (1999), "Network Topologies."
[http://www.delmar.edu/Courses/ITNW2313/ network.htm].

Miller, L. E., (2001), "Catalog of Network Connectivity Models."
[http://w3.antd.nist.gov/wctg/netanal/netanal_netmodels.html].

Sharma, R. L., (1990), *Network Topology Optimization*, Van Nostrand Reinhold, New York.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia 22060

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California 93943

3. Professor R. Kevin Wood
   Department of Operations Research, Code OR/Wd
   Naval Postgraduate School
   Monterey, California 93943

4. Professor Matthew Carlyle
   Department of Operations Research, Code OR/Cm
   Naval Postgraduate School
   Monterey, California 93943

5. Deniz Kuvvetleri Komutanligi
   Kutuphane
   Bakanliklar, Ankara, TURKEY

6. Deniz Harp Okulu Komutanligi
   Kutuphane
   Tuzla, Istanbul, TURKEY

7. Bilkent Universitesi Kutuphanesi
   Bilkent, Ankara, TURKEY

8. Orta Dogu Teknik Universitesi Kutuphanesi
   Balgat, Ankara, TURKEY

9. Bogazici Universitesi Kutuphanesi
   Bebek, Istanbul, TURKEY

10. Dz. Utgm. Ozgur Erken
    Deniz Kuvvetleri Komutanligi
    Bakanliklar, Ankara, TURKEY