Formulations and Algorithms for Network Design Problems with Connectivity Requirements

by

S. Raghavan

B.Tech., Aeronautical Engineering
Indian Institute of Technology, Madras
(1987)

M.S., Operations Research and Statistics
Rensselaer Polytechnic Institute
(1988)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1995

Author .....
Department of Electrical Engineering and Computer Science
December 9, 1994

Certified by ....
Thomas L. Magnanti
George Eastman Professor of Management Science
Thesis Supervisor

Accepted by .....
Robert M. Freund
Acting Co-Director, Operations Research Center

Formulations and Algorithms for Network Design Problems with Connectivity Requirements

by

S. Raghavan

Submitted to the Department of Electrical Engineering and Computer Science
on December 9, 1994, in partial fulfillment of the requirements for the
degree of Doctor of Philosophy in Operations Research

Abstract

Recent rapid technological advances (including high-capacity fiber-optic cables) in the telecommunications industry have introduced numerous opportunities for services other than traditional voice transmission. Disruptions in service for these high capacity systems can be (and have been) disastrous. For these reasons, network reliability is critical. This thesis studies the problem of topological design of survivable networks, that is, networks capable of maintaining communication capabilities in response to equipment failure.

We study structural properties and develop strong formulations for a general *Network Design Problem with Connectivity Requirements (NDC)* and for special cases. Using these strong formulations, we devise *dual-ascent* methods that provide good lower bounds and heuristic solutions with performance guarantees on their degree of suboptimality. In particular:

1. We develop a dual-ascent algorithm for a very general NDC problem. This algorithm optimally solves special cases including the k-edge-disjoint path problem and the k-node-disjoint path problem. The algorithm is the most general heuristic procedure of this kind.

2. We investigate the polyhedral structure of a flow formulation for the unitary NDC problem. By projecting out the flow variables, we show that the flow-based formulation implies three new classes of valid inequalities—partition, odd-hole, and combinatorial design—for a well-known cutset formulation of the problem.

3. Using this strong formulation, we devise and empirically test a dual-ascent solution approach for the network design problem with low connectivity requirements. This algorithm requires (on average) 950 seconds of CPU time on a Sun SPARCstation 10 workstation to solve problems with up to 300 nodes and 3000 edges. The algorithm generates solutions known to be within 4 percent of optimality for typical telecommunication applications, and within 1 percent of optimality for classical Steiner tree problems.

4. We develop a new (directed) formulation for the NDC Problem.

The results in this theses demonstrate the value of good problem formulations and the effectiveness of dual-ascent solution procedures that exploit a problem's special structure.

Thesis Supervisor: Thomas L. Magnanti
Title: George Eastman Professor of Management Science

Acknowledgments

Five and a half years is probably the longest time I have spent in any one place. They have also been the most enjoyable. Now that I am leaving (hard to believe!), I would like to thank many who have contributed to this endeavor.

First, a big "THANK YOU" to my advisor Tom Magnanti, without whose guidance and support none of this research would have been remotely possible. I thank him for his patience with me, and for his enthusiasm and constant encouragement, especially these last few months.

I would also like to thank my other committee members, Anant Balakrishnan and Michel Goemans. Both of them have been very generous with their time, feedback and advice.

The Operations Research Center (ORC) has been a very friendly and comfortable place to work in. I would like to thank Cheryl, Laura and Paulette for the wonderful job they do in making the ORC what it is.

On the more personal side, MIT was not the social dungeon that I envisioned. I would like to thank the many ORCers (past and present) for their friendship. In particular, I would like to mention my classmates Armann, Joe, Kerry (she will be glad to answer any questions concerning the proofs in this thesis), Pieter, Rob (I owe you a dinner) and Sungsu (his contributions are too many to fit on this page) for sharing the trials and tribulations of grad student life these past years.

I would like to thank my parents for stressing upon me the value of education and for their support and encouragement through the years. My brothers and sisters have been a remarkable source of inspiration, and I would like to thank them for everything they have done for me.

Ellen, words cannot express my gratitude. You were always there for me through these ups and downs. I hope that I can do the same.

Finally, I would like to dedicate this thesis to Ellen and Sungsu. Their support, friendship and love have made this dream come true.

Contents

Chapter 1

Introduction

1.1 Motivation

In the past few years the telecommunications industry—its technology and its services—has undergone drastic changes. Rapid technological advances have made it possible to provide services other than the traditional voice transmission. The magazine *Telephony*, as cited in [SP93], has estimated that US telecommunication companies (telcos) invested $23 billion in telecommunication equipment alone in 1992. For the seven Baby Bells and GTE, this investment accounted for approximately 15.8 percent of total revenues. Effective network design and management are critical for assuring that the telcos are poised to meet future demand and are profitable. Standard and Poor's industry surveys [SP93] have attributed, in part, the increased profits of telcos in 1992 (up to $11.1 billion in 1992 from $8.76 billion in 1991 for the seven Baby Bells and GTE) to efficiencies resulting from effective network upgrades and management.

Traditional voice transmission is now only part of the telcos' overall offerings. The market for data traffic has been growing at a rate of 20 percent per year, as opposed to 6 percent per year for voice, and is expected to grow at an even faster rate in the future. Data transmission requires higher capacity lines: in anticipation of this demand, the telcos have been installing high capacity fiber-optic cables[1] in some metropolitan areas. Fiber-optic cables, because of their high capacity, can provide a wide variety of services including video on demand (e.g., movies, music videos etc.), video conferencing, distance learning, remote medical imaging and diagnoses, and telecommuting. Indeed, the potential capabilities of the fiber-optic networks are so staggering (and futuristic) that the Vice President of the United

[1]Fiber-optic cables have the capability to carry orders of magnitude more traffic than the traditional copper cable technology. A three inch thick copper cable contains 1200 pairs of copper wire and is capable of carrying 14,400 calls. A 1/2-inch thick fiber optic cable contains 72 pairs of fiber, each capable of carrying 32,256 calls, and is capable of carrying approximately 4.5 million calls.

States, Albert Gore, advocates building a fiber-optic *information superhighway* to bring the US into the information age and increase its global competitiveness. The Governor of the State of New York, Mario Cuomo, has said, "The American dream is riding on Telecommunications."

Fiber-optic cables are one of the fastest growing segments of the network equipment market. For most telcos, upgrading their networks, including the part connecting the main telecommunications network to residential customers, is a priority (providing fiber-optic based service to residential customers is popularly called "fiber to the home" and "fiber to the curb"). Standard and Poor estimates [SP94] that the major US telcos will spend approximately $125 billion in coming years on the implementation of fiber-optic systems and other related infrastructure. New Jersey Bell expects to spend $1 billion over seven years, starting in 1993, to fully upgrade their network to a fiber-optic system. In November 1993, Pacific Bell announced a $16 billion plan to install fiber-optic systems in their network to bring fiber to the home in the urban areas of California. Similarly, Southern New England Telephone has announced plans to invest $4.5 billion over the next fifteen years in its Connecticut network to bring fiber to the home. The Japanese telephone company Nippon Telephone and Telegraph has made fiber to the home a goal by the year 2015 and expects to spend approximately $400 billion to achieve this goal. Further investments in fiber-optic systems are anticipated when and if the telecommunications industry becomes less regulated.

In the future, since society is likely to rely heavily on the varied services offered by fiber-optic networks and other high bandwidth technologies, their reliability is critical. Disruptions in service, even now, are disastrous and not uncommon. For example, in 1991 a major network outage created a queue of at least 35 planes in the three airports around New York City (La Guardia, Kennedy and Newark), blocked 471,000 international calls, and blocked more than 4.5 million domestic calls [TEM93a]. In 1990, a software malfunction halted 50 percent of AT&T's switched traffic [TEM93b] for nine hours. Banks, brokerage firms and credit card firms rely on continuous error-free transmission of financial data valued at millions of dollars per minute. The banking industry alone estimates a loss of more than $100 million (see [TEM94]) during a snow storm on the East Coast of the US in 1993 that brought a nationwide network of automatic teller machines to a standstill.

Network survivability is defined as the ability of a network to maintain its communication capabilities in the face of equipment failure. In a communication network, failures can be of many types. Failures can occur because of software bugs in the network routing software, or the loss of a facility in the network. For example, cables (links in the network) that carry traffic, or switching equipment (nodes in the network) can fail.

Usually the equipment itself is quite reliable, but it fails because of natural disasters

or human error. For example, the Network Reliability Council estimated [TEM93a] that dig-ups (i.e., accidental cuts of cables) create 58 percent of fiber-optic cable failures. Cable failures arise in several other ways: vehicle collisions with utility poles (7.5 percent), procedural errors (6.9 percent), and electric power lines contacting aerial cables (4.4 percent). Other causes are rodent damage, sabotage, fire, firearms, floods, and excavations, each of which was less than 4 percent. Since a single cable-sheath can carry as many as 4.5 million calls, the consequences of a cut can be very damaging. Switching equipment failure, though rare, does occur, at times because of fires (switches use very high DC voltages) and at times because of floods. The consequences of switching equipment failure are usually more severe than cable cuts, since many cables feed into the same switching equipment which is capable of processing millions of calls. The consequences of failure of an entire Central Office, which usually contains many switches and processes the calls for a geographic region, can be catastrophic.

Network survivability is an important issue for the telcos. In 1992 AT&T launched a $600 million program to improve network survivability [TEM93b]. The telcos advocate building alternate fiber routes to provide alternate routing capabilities in the case of cable cuts or loss of switching equipment. In fact, there is a growing trend by the telcos to build (bidirectional) ring networks. Ring networks provide the capability to detect and reroute traffic in the case of failure of a single fiber-optic link or single intermediate routing equipment. Currently, most telcos believe it is important to protect against single link and single node failures in the network.

The telecommunications industry is also a rich field for network design problems other than network survivability. The industry needs to make decisions ranging from facilities upgrading, traffic routing, and network construction. Developing nations like China, India, and Indonesia have a phone penetration rate of only 2 lines per 100 people (as opposed to 58 lines per 100 people in the US) and are making massive investments[2] to build telecommunication infrastructure to provide telephone service. Efficient network design and management provide fast investment returns for the telcos and less expensive phone service for the public.

1.2 Problem Description and Thesis Objectives

This thesis studies the problem of topological design of survivable networks. Given a set of nodes (offices, switches), possible links, costs for each link, and either the number of permitted link or node failures (in a way we shall make more precise shortly) between each

[2]A case in point is China. China is planning to reach a phone penetration rate of 40 lines per 100 people by the year 2020. Given China's population, this means adding approximately 15–17 million lines per year for the next 25 years. China is spending $6 billion on telecommunications equipment in 1994 and is expected to spend about $90 billion on telecommunications equipment over the next 6 years.

pair of nodes, we want to design a cost effective communication network. We refer to this problem as the *Network Design Problem with Connectivity Requirements (NDC).*

Our model considers only costs associated with the topology of a network. The cost of installing a cable between a pair of nodes typically will include the cost of digging trenches and placing the fiber-optic cable into service. Other considerations, such as what facilities[3] to install on each link are ignored. Our sole focus on topological design (i.e., if a link is in the network or not) might appear to be myopic. However, there are two major justifications for this viewpoint. First, currently network planners (see [CMW89]) use a hierarchical approach, first deciding the topological design and then the facilities to install on these links. Second, if we include demand considerations, the problem becomes very complicated. Developing good methods to solve the NDC problem provides us a solid foundation (e.g., subroutines) to build upon when tackling the problem with demands.

As fiber-optic penetration increases, network planners will need to solve very large NDC problems, that is, problems with up to several hundred nodes and several thousand links. In this thesis our objective is to study the NDC problem and develop solution methods that can solve large-scale versions of the NDC problem.

1.2.1 Formal Problem Description

We will now formulate the NDC problem more precisely, introducing notation and terminology we need along the way.

We represent the network as a *graph* G with *nodes* N that represent switches and *edges* E that represent cables. We consider two types of models: those permitting at most one edge and those permitting parallel edges between a pair of nodes. (In practice, the model used by network planners at Bell Communications Research [CMW89] permits at most one edge between a pair of nodes.)

We say that a set of paths $P_1, \ldots, P_k$ between a pair of nodes s and t are *edge disjoint* if they have no edge in common. If the set of paths $P_1, \ldots, P_k$ have no node in common other than s and t (and so they have no edge in common), then we say the paths are *node disjoint.*

An important result in graph theory, Menger's theorem [Men27], relates edge (resp. node) survivability between any pair of nodes to the number of edge-disjoint (resp. node-disjoint) paths between these nodes.

Theorem 1.2.1 (Menger) *In a graph:*

1. *The maximum number of edge-disjoint paths between node s and node t equals the minimum number of edges whose removal from the graph disconnects them.*

[3]The type of facility (i.e., the number of fibers) to install depends upon the traffic estimates (demands).

2. *If the graph does not contain the edge $\{s,t\}$, the maximum number of node-disjoint paths between node s and node t equals the minimum number of nodes whose removal from the graph disconnects them.*

Thus, ensuring k edge-disjoint paths between nodes s and t guarantees that the communication between them survives the failure of up to $k-1$ edges. Similarly, ensuring k node-disjoint paths between nodes s and t guarantees the communication between nodes s and t survives $k-1$ node failures. In practice (see [CMW89]) planners usually specify survivability requirements as the number of edge- or node-disjoint paths between various nodes in the network.

We now formally state the network design problem with connectivity requirements.

Network Design Problem with Connectivity Requirements (NDC): We are given an *undirected* graph $G=(N,E)$, with node set N and edge set E, and a cost vector $\mathbf{c} \in \mathcal{R}_+^{|E|}$ on the edges E. We are also given a symmetric $|N| \times |N|$ matrix $\mathbf{R} = [r_{ij}]$ that prescribes the number of edge- or node-disjoint paths needed between each pair of nodes: r_{ij} represents the required number of edge- or node-disjoint paths between nodes i and j (if the requirement is for node-disjoint paths, then we use a superscript $[n]$ to denote that the paths must be node disjoint). We wish to select a set of edges that satisfy these requirements at minimum cost, as measured by the sum of costs of edges we choose.

We refer to the requirements r_{ij} as connectivity requirements, using the notation *edge connectivity* if the paths must be edge disjoint and *node connectivity* if the paths must be node disjoint. Note that for all pairs of nodes s and t we either specify edge-connectivity requirements or node-connectivity requirements, but not both. In Chapter 4 we will discuss a modeling extension that permits both edge- and node-connectivity requirements.

The NDC problem is an important *combinatorial optimization problem.* It is very general and many network design problems can be cast as special cases of it. For example, the *Survivable Network Design Problem (SND)* is a special case of the NDC problem. The SND problem arises in a telecommunications problem in the Regional Bell Operating Companies (RBOCs) (see [CMW89]). Each node v in the graph has a connectivity requirement r_v. For each distinct pair $\{s,t\}$ of nodes, the network must contain at least $r_{st} = \min\{r_s, r_t\}$ disjoint s–t paths. The SND problem does not permit both edge- and node-connectivity requirements. Therefore, all nodes have either an edge- or a node-connectivity requirement (once again we denote node-connectivity requirements by the superscript $[n]$). Note that the SND problem is an NDC problem with connectivity requirements $r_{st} = \min\{r_s, r_t\}$.

We now briefly describe other well-known network design problems, and show how to represent them as NDC problems.

Minimum Spanning Tree: Design a minimum cost network that connects (spans) all

the nodes in a network. The minimum spanning tree problem is an SND problem in which each node has connectivity requirement 1.

Steiner Tree Problem: We are given a set T of terminal nodes in a graph that we wish to connect. Find a minimum cost network to connect the nodes in T. Note that the network design can use nodes from N not in T. The minimum cost Steiner tree problem is an SND problem in which the nodes in T have a connectivity requirement of 1 and all the other nodes have a connectivity requirement 0.

Disjoint Path Problems: The k-edge-disjoint path problem requires that we find, at minimum total cost, k edge-disjoint paths between a prespecified pair, say s and t, of nodes. To specify the problem as an SND problem, we set the connectivity requirements of nodes s and t as k and the connectivity requirement of all other nodes as zero. Similarly, for the k-node-disjoint path problem, we wish to find, at minimum total cost, k node-disjoint paths between a prespecified pair of nodes (say s and t). To formulate this problem as an SND problem, we set the requirements of nodes s and t as $k^{[n]}$ and the requirements of all other nodes as zero.

Minimum Cost k-Edge-Connected Spanning Subgraph Problem: A graph is *k-edge connected* if it remains connected after deleting fewer than k edges. Whitney [Whi32] showed that a graph is k-edge-connected if and only if it contains k edge-disjoint paths between every pair of nodes. Consequently, the minimum cost k-edge-connected spanning subgraph problem is an SND problem in which each node has connectivity requirement k.

Minimum Cost k-Node-Connected Spanning Subgraph Problem: A graph is *k-node connected* if it remains connected after deleting fewer than k nodes. Whitney [Whi32] showed that a graph is k-node-connected if and only if it contains k node-disjoint paths between every pair of nodes. As a result, the minimum cost k-node-connected spanning subgraph problem is an SND problem in which each node has requirement $k^{[n]}$. Similarly, we can formulate Steiner versions of these problems (i.e., some of the nodes need not be in the network) as SND problems.

Network Design with Low Connectivity Requirements (NDLC): For this special case of the SND problem, which is of particular interest to the RBOCs, the connectivity requirements r_v are restricted to either $\{0, 1, 2\}$ or $\{0, 1, 2^{[n]}\}$ (corresponding to edge- and node-connectivity versions of the NDLC problem).

Point to Point Connection Problem: Given a set of sources $\{s_1, s_2, \ldots, s_P\}$ and a set of destinations $\{t_1, t_2, \ldots, t_P\}$, we wish to find a minimum cost graph that has a path

from s_i to t_i for $i = 1$ to P. We model the problem as an NDC problem as follows. An element r_{ij} in the requirements matrix $\mathbf{R}$ is 1 if the node corresponding to row i and node corresponding to column j are a source destination pair. All other entries in $\mathbf{R}$ are 0.

Steiner Forest Problem: Given a graph $G = (N, E)$ and node sets $T_1, T_2, \ldots, T_P$ with $T_i \cap T_j = \phi$ for all node set pairs i, j, design a graph at minimum cost that connects the nodes in each node set. To formulate the Steiner forest problem as an NDC problem, set r_{ij} in the requirements matrix $\mathbf{R}$ to value 1 if the node corresponding to row i and the node corresponding to column j are in the same node set (say T_k). All other elements of $\mathbf{R}$ are 0. Notice that the point to point connection problem is a special case of the Steiner forest problem with $T_i = \{s_i, t_i\}$ for $i = 1$ to P.

These problems arise in many diverse settings in telecommunications, transportation, and VLSI circuit design. These special cases show that the NDC problem models a wide variety of connectivity problems on graphs. Therefore, methods for solving NDC problems have many potential applications.

We classify NDC problems in two ways. If the connectivity requirements imply that all nodes with a connectivity requirement must be connected, we say the problem is a *unitary NDC problem.* Otherwise, it is a nonunitary NDC problem. For example, the SND problem is a unitary NDC problem, and the Steiner forest problem is a nonunitary NDC problem.

In the next section we will briefly review different solution methods for combinatorial optimization problems and motivate the solution methodology that we pursue.

1.2.2 Solution Methods

Because special cases of the NDC problem, like the Steiner tree problem [Kar72] and the point to point connection problem [LMSL92], are $\mathcal{NP}$-hard, *the NDC problem is an $\mathcal{NP}$-hard* problem. For a comprehensive treatment of computational complexity theory and recent developments in this problem domain, see the textbook by Papadimitriou [Pap94]. From a lay perspective, the fact that the NDC problem is $\mathcal{NP}$-hard implies it is a very difficult problem to solve: no one knows of a *polynomial time* algorithm that optimally solves the problem. (The research community widely believes that there is no polynomial time algorithm for $\mathcal{NP}$-hard problems). Loosely speaking, by a polynomial time algorithm we mean an algorithm whose worst-case running time is bounded by a polynomial in the size of problem parameters. This means that all known algorithms that optimally solve the problem, require in the worst case, an exponential number of operations.

Researchers approach $\mathcal{NP}$-hard combinatorial optimization problems in many ways. Often, they sacrifice optimality and focus on finding good solutions. These methods are

called *heuristics* and can be quite effective. For example, Johnson [Joh90] reported solutions within 1 percent of optimality for 10,000 city *Traveling Salesman Problems*[4] (TSP) and Bentley [Ben92] reported solutions within 4 percent of optimality for million city TSPs, using local search algorithms. However, a heuristic, by itself, does not usually provide information on the quality of the solution.

In order to assess the quality of a heuristic it is important to provide lower bounds[5] on the value of the optimal solution. One approach to providing lower bounds to combinatorial optimization problems is to formulate them as *Integer Linear Programming* (IP) problems. Solving the continuous relaxation (that is, the *Linear Programming* (LP) relaxation) of the IP yields a lower bound on the value of the optimal solution. If solving the LP relaxation provides an integer solution, then we have also found the optimal solution. Often the solutions provided by the LP relaxation are fractional and the bounds they provide are *weak*. In order to improve the LP bound we can add valid inequalities to the model, inequalities that eliminate the LP fractional optimal solution but do not eliminate any integer feasible solution. Methods that iteratively add such valid inequalities are called *cutting plane methods*. Gomory [Gom63] in the early sixties first developed a general theory for these methods. The best valid inequalities to add to a formulation are those that are in a certain sense the *deepest*, ones that lie on the face of the convex hull[6] of the integer feasible solutions to the problem. Such inequalities are called *facets*.[7] The polyhedral approach attempts to describe by linear inequalities the convex hull of the feasible integer solutions to a combinatorial optimization problem.

For $\mathcal{NP}$-hard problems, describing the convex hull by linear inequalities is itself an $\mathcal{NP}$-hard problem. However, often even a partial description of the convex hull is very useful: in many cases, solving the linear program based on the partial description provides an integer solution. Even if they do not provide integer solutions, LP relaxation bounds frequently are close enough to the optimal objective value that a branch and bound procedure finds the optimal solution reasonably quickly. *Polyhedral combinatorics* is the branch of mathematical programming that studies the linear inequality representations of the convex hull of the integer feasible solutions to combinatorial optimization problems. Many early efforts with the polyhedral method have studied the TSP (see [GP85]). Recently, using the polyhedral

[4]The traveling salesman problem is to find a minimum cost tour (sequence of edges) in a graph that visits every node (city) exactly once.

[5]Without loss of generality, we assume the problem is a minimization problem.

[6]$\mathbf{x}$ is a *convex combination* of a set of points $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^l$ if $\mathbf{x} = \sum_{i=1}^{l} \lambda_i \mathbf{x}^i$ for some vector $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_l)$ with $\sum_{i=1}^{l} \lambda_i = 1$ and $\lambda_i \geq 0$ for $i = 1, \ldots, l$. The *convex hull* of a set S is the set of all points that are convex combinations of points in S.

[7]To be more precise, a valid inequality $\mathcal{F}$, for a polyhedron $\mathcal{P}$, is facet-defining if it is a face of the polyhedron $\mathcal{P}$ whose dimension is one less than the dimension of $\mathcal{P}$. Consequently, a facet belongs to the set of inequalities that are essential (nonredundant) to the linear inequality description of the polyhedron $\mathcal{P}$.

method, researchers have been able to solve to optimality a TSP with as many as 7397 cities [ABCC94]. Following up on the success of the polyhedral method for the TSP, researchers have studied the polyhedral structure of many other combinatorial optimization problems (see Nemhauser and Wolsey [NW88], Hoffman and Padberg [HP85]).

Although the polyhedral approach has met with some success after many years of effort on the TSP, in general the polyhedral approach can be computationally intensive. There are three main ingredients to the polyhedral method. (i) Finding classes of facet-defining inequalities for the combinatorial optimization problem. This can be quite a challenging task by itself. (ii) Solving the so called *separation problem.* Most combinatorial optimization problems have exponentially many facet-defining inequalities. Instead of adding all of these inequalities explicitly to the LP, researchers add the inequalities only if required. Therefore, given a fractional solution to an LP relaxation of the problem, and a class of facet-defining inequalities, we need to either produce a violated inequality or show that the solution satisfies all the inequalities. This is the *separation problem.* For $\mathcal{NP}$-hard combinatorial optimization problems, often the associated separation problem for given classes of facet-defining inequalities is $\mathcal{NP}$-hard. (iii) Implementing cutting plane methods (e.g., determining which facets to use, how to generate them, and how to combine facets with an LP solver) based on the the results of the first two steps. This step is very much an art and is often problem and instance dependent. Therefore, polyhedral methods have not been particularly successful as a generic solution approach for quickly solving large-scale problems. Another disadvantage is that the polyhedral method can take a very long time to solve large-scale problems. If we terminate it before it obtains the optimal solution, the method provides only a fractional solution. Thus, the polyhedral method, although very useful to find provably optimal solutions and useful for several important classes of special applications, is unable to provide a good solution for a given problem when it terminates, perhaps because of running time limitations, before it finds the optimal solution.

Two other methods, *Lagrangian relaxation* and *dual-ascent* have been effective for solving many large-scale combinatorial optimization problems. Lagrangian relaxation methods are typically used in formulations that have a set of relatively easy constraints and a set of complicating constraints. By relaxing the complicating constraints and penalizing violations to them in the objective function, we obtain a simpler problem, called the subproblem, to solve. The objective value to the subproblem is a lower bound on the optimal objective value. The Lagrangian dual problem seeks the choice of Lagrange multipliers (the penalties for violating the constraints) that provide the best lower bound; that is, the choice of Lagrange multipliers that provides the largest objective value when we solve the subproblem. Lagrange multiplier theory shows that the bound provided by the solution of the Lagrangian dual is always as large as the objective value of the linear programming relaxation of the

problem. Lagrange multiplier methods have proven to be quite effective in practice. In a seminal paper, Held and Karp [HK70] used Lagrangian relaxation on the subtour elimination formulation of the TSP to provide a lower bound on an optimal tour. In practice, these bounds are within 2 percent of the optimal solution [Joh90].

Linear programming theory shows that solving a linear program is equivalent to solving its dual. Sometimes instead of solving the LP relaxation of a problem, we might solve the dual, perhaps because it exhibits some special structure that we can exploit. Dual-ascent methods operate on the LP dual. They choose the dual variables iteratively, improving the dual objective (maximization) at each step. Dual-ascent methods also arise in Lagrangian relaxation: in this setting, we iteratively adjust the Lagrange multipliers on the relaxed constraints in a way that increases the objective value of the subproblem (dual objective) at each step. Of course, since we are not solving the dual optimally, the dual-ascent procedure need not give the best possible bound. However, it is often an extremely quick method for finding a lower bound and has proven to be very effective in practice for a number of problems. For example, Erlenkotter [Erl78] used a dual-ascent procedure to develop a solution procedure for the uncapacitated facility location problem.[8] In computational experiments the lower bounds produced by his dual-ascent algorithm were always within 1 percent of the optimal integer solution. Wong [Won84] devised a very successful dual-ascent algorithm for the Steiner tree problem. He reported an average gap of less than 1 percent between the upper and lower bounds that his dual-ascent algorithm produces. Balakrishnan, Magnanti and Wong [BMW89] generalized this approach for uncapacitated fixed charge network design problems,[9] solving problems with up to 45 nodes and 500 edges to within 4 percent of optimality. Balakrishnan, Magnanti and Mirchandani [BMM94b] used a dual-ascent approach to solve large-scale two-level network design problems,[10] solving problems with up to 500 nodes and 5000 edges to within 1 percent of optimality.

Another approach to formulating combinatorial optimization problems is to introduce additional continuous variables. Such formulations are called *extended formulations*. For

[8]The uncapacitated facility location problem can be described as follows. Let $I = \{1, \ldots, n\}$ denote possible locations for a distribution facility, and let c_i denote the cost of opening (locating) a facility at location i. Let $J = \{1, \ldots, m\}$ denote customers, and let d_{ij} denote the cost of satisfying customer j's demand from a facility at location i. The problem is to determine, at minimum cost, which facilities to open, and for each customer, the facility that satisfies its demand.

[9]In the uncapacitated fixed-charge network design problem, we are given a set of nodes, a set of uncapacitated edges (arcs), and, between selected pairs of nodes, a required flow that must be routed over the network. Each edge (arc) has two types of costs: a per unit flow cost and a fixed cost for using the edge (arc). The problem is to select a subset of edges (arcs) that minimizes the sum of the routing costs and the fixed-charge costs.

[10]In the two-level network design problem, we are given a network with two possible facility types for each edge—primary and secondary—and a partition of the nodes into two types—primary nodes and secondary nodes. In this scenario, primary edges are more expensive than secondary edges. We wish to design a minimum cost connected network that connects the primary nodes to each other by primary edges.

example, network design problems can often be formulated with integer variables to represent the edges (the natural[11] variables) and with continuous variables interpreted as flows in the network. Extended formulations can be compact (i.e., they have a polynomial number of variables and constraints) and represent an exponential set of constraints in the space of the natural variables. For example, Wong [Won80] presented a compact multicommodity flow formulation of the TSP that has the same LP relaxation as the well-known (exponential) subtour elimination formulation. In many problem settings, it is easier to solve these extended formulations or use their structure, since they often have network flow variables, to devise Lagrangian or dual-ascent methods. For example, Wong's dual-ascent procedure, which is one of the most successful algorithms for the Steiner tree problem to date, is based on an extended flow-based formulation.

Dual-ascent methods can be attractive for another reason: for certain problems the dual information they provide can be used to devise a heuristic solution to the combinatorial optimization problem. These methods might be attractive because the heuristics are based on the dual variable information and therefore in a certain sense have some basis from optimization theory. Whenever the dual-ascent method operates on the dual of an LP relaxation, the best bound it provides can only be as good as the LP itself. Therefore, it is important to use strong formulations, in a polyhedral sense,[12] when developing dual-ascent methods (and Lagrangian methods).

In this thesis our goal is to study formulations and methods to solve large-scale NDC problems. We study flow-based extended formulations for the NDC problem and examine the strength of different flow-based formulations. For special cases of the NDC problem, we show how to improve the flow-based formulation. In developing solution methodologies, we focus mainly on a dual-ascent approach. We seek dual-ascent methods that provide good lower bounds and heuristic solutions to the NDC problem. We concentrate on two issues:

- How to improve formulations for the NDC problem or special cases of the problem, and
- How to use strong formulations to devise dual-ascent methods that provide good lower bounds and heuristic solutions.

[11] A formulation is a *natural formulation* if it has one variable for each member of its "ground set". For example, a natural formulation for the NDC problem would contain one binary (or integer) decision variable for each edge in the graph.

[12] There might be more than one way to model an integer program. Although these formulations are equivalent as integer programs, when solved as LPs, one formulation might be better—i.e., consistently provide a better solution—than the others because it contains many facet-defining inequalities, or because it most closely approximates the convex hull of the integer feasible points. Such a formulation is called a *strong formulation*.

1.3 Formulations for the NDC Problem

1.3.1 Notation

In this thesis we work with undirected graphs and directed graphs. We refer to an undirected graph as a *graph* and a directed graph as a *digraph*. In order to distinguish between directed and undirected edges, we use *edge* to refer to an undirected edge and *arc* to refer to a directed edge. We also use braces to denote an edge between nodes i and j, i.e., $\{i,j\}$, and parentheses to denote a directed arc from node i to node j, i.e., (i,j). For a directed arc (i,j), we refer to node i as the *tail* of the arc and node j as the *head* of the arc. If one endpoint of an edge is node i, we say that the edge is *incident* to node i. Similarly, if the head or tail of an arc is node i, we say that the arc is *incident* to node i.

We refer to the set of node-connectivity requirements in the matrix $\mathbf{R}$ as $\mathbf{R}^{[n]}$. At times, we will refer to connectivity requirements of a set of nodes. Let

$$\begin{aligned} \text{econ}(T) &:= \max\{r_{ij} | j \in T; i \in N\backslash T\}; \\ \text{ncon}(T) &:= \max\{r_{ij} | r_{ij} \in \mathbf{R}^{[n]} j \in T; i \in N\backslash T\}. \end{aligned}$$

Econ(T) is the maximum *edge-connectivity* requirement between any node in T and its complement. Similarly, ncon(T) is the maximum *node-connectivity* requirement between any node in T and its complement.

In models that permit parallel edges, we let $G = (N, E)$ represent the underlying graph and b_{ij} represent the number of parallel edges allowed between nodes i and j. For example, if a model permits two edges between nodes i and j, then the graph G contains the edge $\{i,j\}$ and $b_{ij} = 2$.

We will also consider graphs obtained by deleting nodes or edges from a graph G. We let $G - Z$ denote the graph obtained by deleting a set Z (of nodes, edges, or combinations of nodes and edges). In deleting any node, we also delete all the edges incident to that node.

In an undirected graph, any set of nodes T defines a cut $\{N\backslash T\}$-$\{T\} = \{\{i,j\} : i \in N\backslash T, j \in T\}$. We let $\delta(T)$ denote the set of edges in this cut and refer to $\delta(T)$ as a cut. We use $\delta_G(T)$, in case of possible ambiguity, to indicate which graph the cut T is defined on. Similarly, any set of nodes T in a directed graph defines a dicut $\{N\backslash T\}$-$\{T\} = \{(i,j) : i \in N\backslash T, j \in T\}$. We let $\delta^-(T)$ denote the set of arcs in this dicut, that is the set of arcs directed into T. Also, let $\delta^+(T)$ denote the set of arcs directed out of T. An *s-t cut* is a cut, say $\delta(T)$, with $s \notin T$ and $t \in T$. Similarly, an *s-t dicut* is a dicut, say $\delta^-(T)$, with $s \notin T$ and $t \in T$.

Sometimes, we will *contract* a set of nodes in a graph or digraph. To contract a set of nodes T in a graph

1. Add a single node, say t, to the graph (digraph).

2. Replace an edge $\{i,j\}$ in $\delta(T)$ with $j \in T$ by the edge $\{i,t\}$. (For a digraph (i,j) in $\delta^-(T)$ [resp. (i,j) in $\delta^+(T)$] by the arc (i,t) [resp. (t,j)].) If this operation creates parallel edges (arcs) between nodes, delete all but the smallest cost parallel edge (arc).

3. Delete the nodes in T and all edges (arcs) incident to them.

The *capacity of a cut* $\delta(W)$ is defined as the sum of the capacity of the edges in the cut. The *capacity of a dicut* $\delta^-(W)$ is defined as the sum of the capacity of the arcs in the dicut.

Sometimes we will want to eliminate the variables from an "extended" formulation of a problem. Let $\mathbf{A}$ and $\mathbf{B}$ be two given matrices and $\mathbf{b}$ be a column vector, all with the same number of rows. Consider the polyhedron $P = \{(\mathbf{x},\mathbf{f}) : \mathbf{Ax} + \mathbf{Bf} \leq \mathbf{b}\}$. The polyhedron $Q = \{\mathbf{x} : \mathbf{Ax} + \mathbf{Bf} \leq \mathbf{b}$ for some vector $\mathbf{f}\}$ obtained by eliminating the $\mathbf{f}$ variables is called the *projection* of the the polyhedron P.

If two formulations for a problem provide, for all objective function coefficients, the same optimal objective value when solved as LPs, we say the two formulations are *equivalent.*

We say that adding an inequality $\mathcal{I}$ *strengthens* a formulation of a (mixed) integer programming problem if it is valid and adding it to the formulation improves the objective value of the LP relaxation of the formulation for some choice of the objective function coefficients. We say that a formulation $\mathcal{P}_1$ is stronger than a formulation $\mathcal{P}_2$ if, when solved as LPs, the objective value of $\mathcal{P}_1$ is always better than the objective value of $\mathcal{P}_2$, and sometimes is strictly better than the objective value of $\mathcal{P}_2$.

Throughout this thesis, we use some complexity notions as applied to network design problems. Typically, the network problems we consider have the following associated data parameters: number of nodes $|N|$, number of edges $|E|$, costs, and sometimes, capacities on the edges. We represent the largest cost by C and the largest edge capacity by U. An algorithm is said to be a *polynomial time algorithm* if its worst-case running time is bounded by a polynomial in the size of the problem parameters. For network design problems, this means the worst-case running time is bounded by a polynomial in $|N|$, $|E|$, $\log U$, and $\log C$. A polynomial time algorithm is said to be *strongly polynomial* if its worst-case running time is bounded by a polynomial function in the number of nodes $|N|$ and the number of edges $|E|$. Otherwise, we say the polynomial algorithm is *weakly polynomial.* An algorithm is *psuedopolynomial* if its worst-case running time is bounded by a polynomial in $|N|$, $|E|$, and C, and U.

1.3.2 Formulation

In Section 1.2.1 we stated Menger's theorem in terms of paths and edges (or nodes) whose deletion destroys these paths. Another equivalent statement of Menger's theorem relates

paths and edges across cutsets:

Theorem 1.3.1 *The maximum number of edge-disjoint paths between nodes s and t in a graph equals the minimum number of edges across every s–t cut.*

Menger's theorem permits us to state the following well-known "cutset" formulation for the NDC problem. Let x_{ij} be an integer variable that represents the number of copies of edge $\{i,j\}$ in the network.

Cutset formulation for the NDC problem:

$$\text{Minimize} \quad \sum_{\{i,j\}\in E} c_{ij}x_{ij} \tag{1.1a}$$

$$\text{subject to:} \quad \sum_{\{i,j\}\in\delta(T)} x_{ij} \geq r_{st} \qquad \begin{array}{l}\text{for all pairs } s,t \in N,\ s \neq t \text{ and}\\ \text{for all } T \text{ with } s \in N\backslash T,\ t \in T;\end{array} \tag{1.1b}$$

$$\sum_{\{i,j\}\in\delta_{G-Z}(T)} x_{ij} \geq r_{st} - |Z| \qquad \begin{array}{l}\text{if } r_{st} \geq 2 \text{ with } r_{st} \in \mathbf{R}^{[n]}, \text{ and}\\ \text{for all pairs } s,t \in N,\ s \neq t \text{ and}\\ \text{for all } Z \subset N\backslash\{s,t\} \text{ with } 0 < |Z| < r_{st},\\ \text{and for all } T \subset N\backslash Z; \text{ with } s \notin T,\ t \in T;\end{array} \tag{1.1c}$$

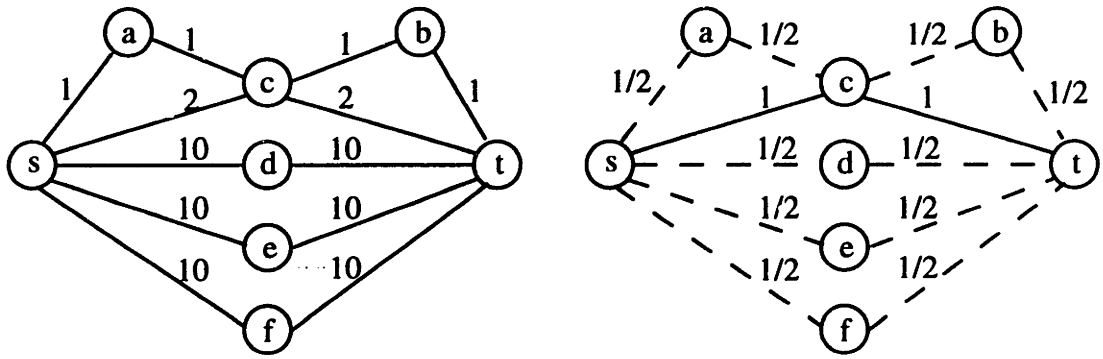
$$x_{ij} \leq b_{ij} \qquad \text{for all } \{i,j\} \in E; \tag{1.1d}$$

$$x_{ij} \geq 0 \qquad \text{and integer, for all } \{i,j\} \in E. \tag{1.1e}$$

Constraint (1.1b) says that if the desired design must contain either r_{st} edge- or node-disjoint paths between nodes s and t, then the network contains at least r_{st} edges across each s–t cut. Moreover, if the paths between nodes s and t are to be node disjoint, then deleting any subset of $k \leq r_{st}$ nodes should leave at least $r_{st} - k$ paths between s and t. Constraint (1.1c) represents this condition. Constraint (1.1d) says that the network can contain at most b_{ij} copies of edge $\{i,j\}$. For models that do not permit parallel edges, $b_{ij} = 1$. For models that permit parallel edges, b_{ij} represents the maximum number of parallel edges between nodes i and j.

Note that we could model the node-connectivity requirements with fewer constraints of type (1.1c) since if a network contains r_{st} node-disjoint paths between nodes s and t, then if we delete any $r_{st} - 1$ nodes, the network will still contain a path between nodes s and t. To model this condition requires the use of constraint (1.1c) for only those sets Z containing $r_{st} - 1$ nodes. Therefore, in the integer programming model the other constraints of type (1.1c) are redundant.

As shown by the example in Figure 1-1, however, the additional constraints can improve (strengthen) the LP relaxation of this model. In this example, we want to find, at minimum cost, 3 node-disjoint paths between nodes s and t. Figure 1-1a shows the edge costs and



(a) (b)

Figure 1-1: Example to show that constraints (1.1c) can strengthen the formulation. We wish to find 3 node-disjoint paths between nodes s and t: (a) Costs of the edges, (b) Optimal LP solution for the LP model that contains constraint (1.1c) only for those sets Z containing 2 nodes. This solution is not feasible for the LP model that contains all constraints of type (1.1c).

Figure 1-1b shows the (fractional) optimal solution (cost 36 units) to the LP relaxation of the model that contains constraint (1.1c) only for those sets Z containing $r_{st} - 1 = 2$ nodes. Note that deleting any two of the nodes a, b, c, d, e and f satisfies constraint (1.1c). On the other hand, this solution is not feasible to the LP relaxation of the model that contains all constraints of type (1.1c). The solution violates constraint (1.1c) when $T = \{t\}$ and $Z = \{c\}$. The lefthand side of this constraint $x_{dt} + x_{et} + x_{ft}$ has value 1.5 and the righthand side $r_{st} - 1$ has value 2.

By deleting redundant constraints in the LP relaxation of the cutset formulation, we obtain the following formulation.

Cutset formulation (II) for the NDC problem:

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij}$$

subject to:

$$\sum_{\{i,j\} \in \delta(T)} x_{ij} \geq \text{econ}(T) \qquad \text{for all } T \text{ with } \phi \neq T \neq N;$$

$$\sum_{\{i,j\} \in \delta_{G-Z}(T)} x_{ij} \geq \text{ncon}_{G-Z}(T) - |Z| \qquad \text{for all } T \subset N \backslash Z \text{ with } \phi \neq T \neq N \backslash Z, \text{ and for all } Z \subset N \text{ with } 0 < |Z| < \text{ncon}_{G-Z}(T);$$

$$x_{ij} \leq b_{ij} \qquad \text{for all } \{i,j\} \in E;$$

$$x_{ij} \geq 0 \qquad \text{and integer, for all } \{i,j\} \in E.$$

Consider an alternative way to formulate the problem. We can enforce the connectivity requirements of the matrix $\mathbf{R}$ using commodity flows. For each pair $\{s,t\}$ of nodes, with $r_{st} \geq 1$, create a commodity, arbitrarily choosing one of the nodes as the origin of the commodity and the other node as the destination. Let K denote the set of commodities and let q_k, for each $k \in K$, denote the edge- or node-connectivity requirement between the origin and destination of commodity k. Let $K^{[n]}$ denote the subset of commodities that correspond to node-connectivity requirements. For example, if $r_{st} = 3$ then $q_k = 3$ for the commodity k corresponding to the node pair $\{s,t\}$. The following Mixed Integer Program, with x_{ij} representing the number of copies of edge $\{i,j\}$ in the network and f_{ij}^k flows, is a valid formulation for the NDC problem.

Undirected flow formulation for the NDC problem:

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{1.2a}$$

$$\text{subject to:} \quad \sum_{j \in N} f_{ji}^k - \sum_{l \in N} f_{il}^k = \begin{cases} -q_k & \text{if } i = O(k); \\ q_k & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } i \in N \text{ and } k \in K \tag{1.2b}$$

$$\left.\begin{matrix} f_{ij}^k \\ f_{ji}^k \end{matrix}\right\} \leq x_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{1.2c}$$

$$\sum_{l \in N} f_{il}^k \leq 1 \qquad \text{for all } k \in K^{[n]} \text{ and all } i \neq O(k), D(k) \tag{1.2d}$$

$$f_{ij}^k, f_{ji}^k \geq 0 \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{1.2e}$$

$$x_{ij} \leq b_{ij} \qquad \text{for all } \{i,j\} \in E; \tag{1.2f}$$

$$x_{ij} \geq 0 \qquad \text{and integer, for all } \{i,j\} \in E. \tag{1.2g}$$

Constraint (1.2c) in this formulation permits us to send flow on an edge only if it is in the design. The number of units of flow of a commodity we can send on an edge equals the number of copies of that edge in the graph. This capacity constraint, together with the flow balance constraints (1.2b), enforce the edge-connectivity requirements. Consider for example the model that does not allow parallel edges (that is, each $b_{ij} = 1$). If we want to send 3 units of flow from node s to node t, then the formulation forces us to send this flow on 3 edge-disjoint paths between these nodes. Suppose the requirements are for node-disjoint paths. Then constraint (1.2d) ensures that at most one unit of a commodity flows into (and out of due to flow balance constraint (1.2b)) any node that is not the source or destination of the commodity. This ensures that the network will contain the required number of node-disjoint paths.

Note that the solution in Figure 1-1b has no feasible flow satisfying constraint (1.2d). The only feasible flow, with the values of the edge variables shown in Figure 1-1b, that sends 3 units of flow from s to t has $f_{sc} = f_{ac} = 1$, and $f_{sa} = f_{ac} = f_{cb} = f_{bt} = f_{sd} = f_{dt} = f_{se} = f_{et} = f_{sf} = f_{ft} = 1/2$. (This solution sends one unit of flow on the path s-c-t, and half a unit of flow on the paths s-a-c-b-t, s-d-t, s-e-t and s-f-t). However, this solution has 1.5 units of flow flowing into node c, violating constraint (1.2d). (This formulation also provides the integer optimal solution for the problem in Figure (1-1).)

We will come back to the cutset and undirected flow formulation in Chapter 4, where we will show that they are equivalent. We will then describe a dual-ascent algorithm for the NDC problem based on the undirected flow formulation.

1.4 Organization of the Thesis

In Chapter 2 we review the NDC problem literature. We discuss heuristics as well as exact algorithms. We also introduce a directed network design problem with connectivity requirements and review its literature.

In Chapter 3 we describe the basic principles underlying the dual-ascent method. We illustrate the method by considering the minimum spanning tree problem and the more general minimum cost branching problem.[13] Our discussion shows how to use the dual-ascent approach to derive Edmonds' algorithm [Edm67] for the minimum cost branching problem. We then provide a Lagrangian duality interpretation of Edmonds' algorithm. Finally, we show how to extend these ideas to derive Wong's algorithm [Won84] for the Steiner branching problem.

In Chapter 4 we consider the network design problem with connectivity requirements. We compare the flow-based formulation to the traditional cutset formulation. We then provide a dual-ascent algorithm that generates both a feasible heuristic solution and a lower bound for the problem. For the k-edge-disjoint path problem and the k-node-disjoint path problem, the dual-ascent algorithm provides the optimal solution. We also compare our method with that of Goemans et al. [GGW93, WGMV93, GGP+94] and describe similarities and differences between the two algorithms. We then consider several modeling extensions to the NDC model. These extensions permit connectivity requirements of the following type: the network design must contain k edge-disjoint paths between nodes s and t, of which at least l must be node disjoint. We also show how to modify the dual-

[13]The minimum cost branching problem can be described as follows. Given a digraph $D = (N, A)$ and a designated root node r, design a minimum cost network that contains a directed path from the root node to all other nodes. In the Steiner version of this problem, the minimum cost Steiner branching problem, we are also given a set T ($r \notin T$) of terminal nodes. We need to design a minimum cost network containing a directed path from the root node r to every node in T.

ascent algorithm for the NDC problem so that it applies to directed network design problems with connectivity requirements.

In Chapter 5 we consider the unitary network design problem with connectivity requirements. Motivated by a result due to Nash-Williams [NW60], we show how to to strengthen the flow-based formulation (1.2) for the unitary NDC problem. We investigate the polyhedral structure of the improved formulation by projecting out the flow variables, and show that the improved flow-based formulation implies three classes of valid inequalities—partition, odd-hole, and combinatorial design—for the cutset formulation in the space of the edge variables (these inequalities are generalizations of known inequalities for the Steiner tree problem). Each of these three classes of valid inequalities is new for the unitary NDC problem.

In Chapter 6 we consider the network design problem with low connectivity requirements. We use the improved formulation described in Chapter 5 to devise a new dual-ascent algorithm for this problem. We then study the computational performance of the dual-ascent algorithm. When applied to problems with up to 300 nodes and 3000 edges, the dual-ascent procedure generates solutions known to be within about 4 percent of optimality.

In Chapter 7 we develop an improved flow-based model for the NDC problem (i.e., both unitary and non-unitary). We first consider the Steiner forest problem and introduce a flow-based formulation on a directed network that is stronger than Formulation (1.2). We then generalize this directing procedure and develop an improved flow model for all NDC problems. To our knowledge the literature does not contain a directed model for the Steiner forest problem.

In Chapter 8 we summarize this thesis.

Chapter 2

Literature Review

In this chapter we review the extensive literature on the network design problem with connectivity requirements (NDC). In later chapters, we elaborate on some of these results. Part of the material in this chapter draws on references cited in [Sto92] and [Fra94].

The NDC problem is a special class of the general network design problem. Network design is a fertile field with many different problem variants that are not restricted to the design of physical networks, and include many transportation planning and production planning problems. For surveys of network design problems, see the papers by Magnanti and Wong [MW84] and by Minoux [Min89]. A good collection of recent research results on network design problems is the book *Network Models* edited by Ball, Magnanti, Monma and Nemhauser [BMMN95].

One special case of the NDC problem—the Steiner tree problem—is well studied. For this reason, we will not review the Steiner tree problem in detail. For a comprehensive survey of this problem, see the recent book by Hwang, Richards and Winter [HRW92].

In this chapter, unless otherwise specified we assume the NDC model does not permit edge replication. Usually (if this is not the case we will explicitly mention it), algorithms for the model that does not permit edge replication also apply to the model that permits edge replication. To apply these algorithms replace each edge by $\min(b_{ij}, r_{\max})$ copies of the edge. On the other hand, algorithms designed for the model that allows (unlimited)[1] edge replication usually do not apply to the model that does not permit edge replication.

The plan for the rest of this chapter is as follows. We first describe polynomially solvable cases of the NDC problem. Next, we describe heuristics for the NDC problem, and then describe exact (not polynomial) algorithms for the NDC problem. Finally, we describe a directed version of the NDC problem and review its literature.

[1] In this chapter every model with edge replication permits unlimited edge replication. Therefore, the constant b_{ij} in model (1.2) is equal to $r_{\max}$.

2.1 Polynomially Solvable Cases

The NDC problem is polynomially solvable for certain restricted choices of edge costs c_{ij}, connectivity requirements r_{ij}, or structures of the underlying graph G.

2.1.1 Graph Augmentation Problems

In these problems the underlying graph is complete and the edge costs are either 0 or 1. These problems have the following interpretation. Starting from the graph defined by the zero-cost edges, add the fewest possible edges (augment the graph) to satisfy the connectivity requirements. If all edge costs are 1, these problems are also referred to as uniform cost problems. Observe that uniform cost problems are graph augmentation problems with an empty starting graph.

Uniform Cost Problems: Harary [Har62] first considered problems of this type in the context of the k-node-connected spanning subgraph problem. He constructively showed that the minimum number of edges in a k-node-connected graph on $|N|$ nodes is $\left\lceil \frac{k|N|}{2} \right\rceil$.

Chou and Frank [CF70] considered the edge-connectivity version of the NDC model with edge replication. They constructively showed how to find the optimal solution to this problem. The running time of their algorithm grows linearly with $r_{\max}$, the highest connectivity requirement; therefore, the algorithm is psuedopolynomial. Sridhar and Chandrasekaran [SC92] described a strongly polynomial algorithm that solves the same problem. They modify the Gomory-Hu [GH61] network synthesis algorithm[2] so that it provides integer capacities for the edges. Frank and Chou [FC70, FC69] described an algorithm to solve the edge-connectivity version of the NDC problem without edge replication.

Edge-Connectivity Augmentation: Eswaran and Tarjan [ET76] introduced graph augmentation problems. They showed how to solve the 2-edge-connected augmentation problem. Cai and Sun [CS89] and Watanabe and Nakamura [WN87] solved the general k-edge-connected augmentation problem with edge replication. Frank [Fra92] generalized these results by showing how to solve the edge-connectivity version of the NDC augmentation problem with edge replication. It is an important open problem to find algorithms for the model without edge replication. Taoka, Takafuji and Watanabe [TTW93] have recently taken a step in this direction. They showed how to solve the k-edge-connected augmenta-

[2] In the network synthesis problem we are given a symmetric requirement matrix $\mathbf{R}$ whose entries can be fractional. We need to decide on capacities b_{ij} for the edges so that for every pair of nodes s and t, the network can send at least r_{st} units of flow. This can be considered as the fractional version of the uniform cost problem. Gomory and Hu [GH61] described an algorithm for this problem. Even if all entries in $\mathbf{R}$ are integer, the Gomory-Hu algorithm can give fractional solutions.

tion problem for the model without edge replication when $k \leq 5$ (this algorithm does not apply to the model that permits edge replication).

Node-Connectivity Augmentation: The node-connectivity augmentation problem is quite difficult. In fact, at present, it is not known if the problem is $\mathcal{NP}$-hard or whether it is polynomially solvable (even for the model that permits edge replication). As a consequence, results for the node-connectivity version are limited. Eswaran and Tarjan [ET76] showed how to determine the exact number of edges to be added to make a graph 2-node connected. Using Eswaran and Tarjan's result Rosenthal and Goldner [RG77] provided an $\mathcal{O}(|N|+|E|)$ algorithm (here E is the set of zero-cost edges) for the problem of constructing 2-node-connected graphs. Watanabe and Nakamura [WN88] described an algorithm for constructing 3-node-connected graphs. Recently, Jordán [Jor94] showed how to augment a $(k-1)$-node-connected graph to a k-node-connected graph using at most $\max(0, k-3)$ more edges than the optimal augmentation. Other than these results, not much is known about node-connectivity augmentation problems.

2.1.2 Restricted Connectivity Requirements

The following special cases of the NDC are polynomially solvable.

- The minimum spanning tree problem. In this case, $r_i = 1$ for all nodes in the SND model. Kruksal's [Kru56] and Prim's [Pri57] algorithms are well-known algorithms for solving this problem.
- The k-edge-disjoint or k-node-disjoint path problem. This problem can be formulated as a minimum cost flow problem, which is polynomially solvable. Suurbaale [Suu74] has devised a specialized algorithm for this problem.
- The Steiner tree problem when either the number of terminal nodes or the number of Steiner nodes is bounded. Dreyfus and Wagner [DW71], Lawler [Law76], and Levin [Lev71] have given dynamic programming algorithms for this problem.

2.1.3 Restricted Graphs: Series-Parallel Graphs

Researchers have studied versions of the NDC problem on special graphs called series-parallel graphs (or 2-trees). These graphs can be constructed by a recursive construction process. As a result, many problems that are $\mathcal{NP}$-hard on general graphs are polynomially solvable, using a dynamic programming algorithm, on series-parallel graphs. This is formalized in various ways by Arnborg and Proskurowski [AP89], and Bern, Lawler and Wong [BLW87]. Although the basic algorithm design philosophy for problems on series-parallel graphs is quite easy, deriving a linear time algorithm is not so straightforward. For

example, to prove the correctness of a linear time algorithm for the edge-connectivity version of the NDLC on series-parallel graphs, Raghavan [Rag92] has considered 110 different cases. We now summarize the polynomial algorithms for various special cases of the NDC problem on series-parallel graphs. Wald and Colbourn [WC83] described a linear time algorithm for the Steiner tree problem on series-parallel graphs. Winter [Win87] described linear time algorithms for the Steiner 2-node-connected and Steiner 2-edge-connected spanning subgraph problems on series-parallel graphs. Raghavan [Rag92] described linear time algorithms for both the edge- and node-connectivity versions of the NDLC problem on series-parallel graphs.

Other researchers have focused on providing complete polyhedral descriptions for special cases of the NDC problem on series-parallel graphs. This characterization is important because valid inequalities for a problem on a series-parallel graph are usually valid for general graphs. Consequently, a formulation that provides a complete polyhedral description of the problem on series-parallel graphs is often a strong formulation for the problem on general graphs (for example, the Steiner branching problem).

Goemans [Goe94b] and Margot, Prodon and Liebling [MPL94] independently provided polyhedral descriptions for the node weighted Steiner tree problem on series-parallel graphs -- that is, a Steiner tree problem with a fixed cost (in addition to the cost of its incident edges) for selecting a Steiner node in the network design. These formulations are extended formulations for the Steiner tree problem on series-parallel graphs.

Coullard, Rais, Rardin and Wagner [CRRW91a] provided a complete polyhedral description of the Steiner 2-node-connected spanning subgraph polytope on series-parallel graphs. Baïou and Mahjoub [BM93] described the Steiner 2-edge-connected spanning subgraph polytope on series-parallel graphs.

Chopra [Cho89] studied the minimum cost k-edge-connected spanning subgraph problem. He provided a complete linear inequality description of the polytope associated with the k-edge-connected spanning subgraph problem on an outerplanar graph[3] when k is odd.

Researchers have studied two other classes of decomposable graphs (i.e., graphs that can be constructed recursively): the so called W_4-free graphs and Halin graphs. Coullard, Rais, Rardin and Wagner [CRRW91b] studied the Steiner 2-node-connected subgraph problem on W_4-free graphs. They described a linear time algorithm for this problem and provide a complete polyhedral description of this problem. Winter [Win85] described linear time algorithms for the Steiner 2-node-connected and Steiner 2-edge-connected spanning subgraph problems on Halin graphs.

Recently, Margot and Schaffers [MS91, MS94] reported on a general theory to derive complete polyhedral descriptions for problems on graphs that can be constructed recursively.

[3] An outerplanar graph is a special type of series-parallel graph.

With this unifying theory, they provide alternate proofs for many of the results cited in this section.

It is interesting to note that although there is a linear time algorithm to solve the Steiner tree problem on series-parallel graphs, no one has been able to provide a complete polyhedral description of the Steiner tree problem on series-parallel graph in the space of the natural edge variables. Perhaps, the recent results by Margot and Schaffers will provide some clues as to how to solve this open problem.

2.2 Heuristics for the NDC Problem

Steiglitz, Weiner, and Kleitman [SWK69] seem to have been the first to consider the NDC problem. They considered problems with node-connectivity requirements, describing a heuristic algorithm that works in two steps. It first incrementally adds edges to the graph and creates a feasible solution. Then, it makes local improvements, using pairwise exchanges of edges until no more local improvements are possible. They applied their heuristics to two real-world problems. Their heuristic required approximately 12 minutes of running time on a UNIVAC 1108 for a 6-node-connected spanning subgraph problem on a 58 node graph (the larger of the two problems).

Monma and Shallcross [MS89] described heuristics for the network design problem with low connectivity requirements (NDLC). Their heuristic method also has two parts. It first constructs a feasible solution and then applies a number of local interchange heuristics. Their heuristics were designed to work well on sparse underlying graphs. For some sparse telephone network design problems with 116 nodes, their heuristics required up to 156 seconds of running time on an IBM-PC/AT.

Using similar ideas to those used by Monma and Shallcross, Ko and Monma [KM89] have proposed heuristics for designing k-edge-connected and k-node-connected graphs. They tested their method on random problems. Running times, on a VAX 8650, for their heuristic varied from 13 seconds for problems with 20 nodes and $k = 3$, to 120 seconds for problems with 40 nodes and $k = 5$.

Goemans and Talluri [GT91] studied the k-edge-connected spanning subgraph problem. They showed that if k is even or 1 and G is a k-edge-connected graph, then replacing any two nonadjacent edges (a, c) and (b, d) in G by either the edges (a, d) and (c, b), or the edges (a, b) and (c, d) produces a k-edge-connected graph. As a result, 2-opt[4] is a feasible heuristic for the Steiner k-edge-connected spanning subgraph problem when k is even or 1.

[4] A 2-opt heuristic starts with a feasible network. It then replaces two edges contained in the solution by two edges that are not, if the solution remains feasible and its cost decreases. It continues in this fashion until no further improvement is possible.

2.2.1 Approximation Algorithms for the NDC Problem

An approximation algorithm is an algorithm that provides a heuristic solution with a provable worst-case ratio between the value of the heuristic solution and the value of an optimal integer solution. An α-approximation algorithm, or an algorithm with a worst-case ratio of α, is an algorithm that provides a heuristic solution that is always guaranteed to have a solution value that is at most α times the optimal (integer) solution value. For the general NDC problem no approximation algorithms are known. However for special cases researchers have been able to provide approximation algorithms.

One of the earliest approximation results was for the Steiner tree problem. Several authors (see El-Arbi [EA78], Kou et al. [KMB81], Plesnik [Ple81], and Takahashi and Matsuyama [TM80]) independently proposed a minimum spanning tree heuristic that has a worst-case ratio of $2 - (2/|T|)$ (recall that T denotes the set of required nodes). This algorithm can be described as follows:

Minimum spanning tree heuristic (Takahashi and Matsuyama)

1. Select an arbitrary node $i \in T$ as the initial tree B. Let $T' = \{i\}$.
2. While $T \backslash T' \neq \phi$
   - determine a node $j \in T$ closest to B (break ties arbitrarily),
   - update B by adding the shortest path joining node j to B,
   - $T' = T \cup \{j\}$.

Zelikovsky [Zel93] described a $\frac{11}{6}$-approximation algorithm for the Steiner tree problem that runs in $\mathcal{O}(|T|(|E| + |N||T| + |N| \log |N|))$ time. Berman and Ramaiyer [BR92] build upon Zelikovsky's ideas and proposed a $\frac{16}{9}$-approximation algorithm for the Steiner tree problem that runs in $\mathcal{O}(|N|^{7/2})$ time. Currently, this is the best worst-case bound of any approximation algorithm for the Steiner tree problem.

Frederickson and JáJá [FJ81] described a 3-approximation algorithm for the 2-edge-connected spanning subgraph problem. Khuller and Vishkin [KV92] improved upon this worst-case ratio by giving a 2-approximation algorithm for the k-edge-connected spanning subgraph problem. Their algorithm replaces each undirected edge $\{i, j\}$ by two directed arcs, (i, j) and (j, i), each with the same cost as $\{i, j\}$. It then arbitrarily selects one of the nodes as a root node and solves the minimum cost k-branching problem. That is, it designs a digraph that contains k arc-disjoint branchings rooted at the root node. Since the k-branching problem is a matroid intersection problem (see Section 2.4.1), it is solvable optimally in polynomial time [Edm79]. Next, they showed that the network designed by selecting an edge $\{i, j\}$ if at least one of the arcs (i, j) or (j, i) is in the solution to the

k-branching problem is a k-edge-connected subgraph whose cost is no more than twice that of the optimal solution.

Balakrishnan, Magnanti and Mirchandani [BMM94a] described a tree completion heuristic that is a 2-approximation algorithm for the k-path Steiner tree problem. In the k-path Steiner tree problem we are given a set T of terminal nodes we wish to connect. In addition we require that two prespecified nodes in T, say s and t, are connected by k edge-disjoint paths. The tree completion heuristic can be described as follows:

Tree completion heuristic for k-path Steiner tree problem

1. Solve the k-edge-disjoint path problem between nodes s and t. Set B to be the edges in these k edge-disjoint paths. Set T' to be the set of terminal nodes that are incident to edges in B.
2. While $T \backslash T' \neq \phi$ (continue with Step 2 of the minimum spanning tree heuristic)
   - determine a node $j \in T$ closest to B (break ties arbitrarily),
   - update B by adding the shortest path joining node j to B,
   - $T' = T \cup \{j\}$.

Goemans and Bertsimas [GB93] were the first researchers to devise an approximation algorithm for the SND problem. They described a tree heuristic for the edge-connectivity version of the SND problem that permits edge replication. Its simplicity makes it worth describing.

Tree heuristic

1. Compute, for all pairs of nodes i, j, the shortest path lengths c'_{ij} with respect to the cost vector $\mathbf{c}$.
2. Prepare a sorted list $L = \{0 = \rho_0 < \rho_1 < \rho_2 < \ldots < \rho_p\}$ consisting of all distinct connectivity types.
3. Set $\mathbf{x} = \mathbf{0}$.
4. For $k = 1$ to p do
   - Let $N_k = \{i \in N : r_i \geq \rho_k\}$.
   - Compute $T_k = (N_k, E_k)$, the minimum spanning tree with respect to $\mathbf{c}'$ of the complete graph induced by N_k.
   - Let $x_{ij} = x_{ij} + (\rho_k - \rho_{k-1})$ for all edges $\{i, j\} \in E_k$.

5. If $x_{ij} > 0$, replace each edge $\{i, j\}$ in the complete graph on N by x_{ij} copies of the shortest path from node i to node j.

Goemans and Bertsimas showed that the worst-case ratio θ_1 of the tree heuristic is

$$\left(2 - \frac{2}{|T|}\right)\left(\sum_{k=1}^{p} \frac{\rho_k - \rho_{k-1}}{\rho_k}\right).$$

In this expression, T denotes the nodes with connectivity requirement greater than 0. When applied to the Steiner tree problem the tree heuristic reduces to the well-known minimum spanning tree heuristic. In proving the worst-case ratio of the tree heuristic, Goemans and Bertsimas actually showed that θ_1 is the ratio between the tree heuristic's solution and the optimal linear programming value of a cutset model for the SND problem (see Formulation (1.1) on page 24). As a result, they bounded the ratio between the optimal IP solution and optimal LP solution for the cutset model. Goemans and Bertsimas also showed that if $\rho_1 = 1$ and $p \geq 2$, the worst-case ratio of the tree heuristic improves by a unit to $\theta_1 - 1$.

Goemans and Bertsimas described another algorithm called the improved tree heuristic for the edge-connectivity version of the SND problem that allows edge replication. This heuristic improves upon the tree heuristic when there is a gap in the connectivity requirements—that is, when two consecutive values of ρ differ by more than 1. Suppose the two consecutive values of ρ differ by 2. The improved tree heuristic adds to the current solution (i) one copy of the minimum spanning tree T_k computed by the tree heuristic, and (ii) a minimum weight matching M_k, with respect to the edge cost vector $\mathbf{c}'$, on the odd degree vertices of T_k. Since the union of T_k and M_k is Eulerian,[5] it is 2-edge connected. Therefore, the resulting network has at least 2 more edge-disjoint paths between the nodes in $T_k \cup M_k$ (when applied to the 2-edge-connected spanning subgraph problem, the improved tree heuristic is Christofides' heuristic [Chr76] for the TSP without its short-cutting step). In general, if the gap in the connectivity requirement is even, the improved tree heuristic adds $(\rho_k - \rho_{k-1})/2$ copies of the tree T_k and the matching M_k to the solution. If the gap in the connectivity requirement is odd, it adds $(\rho_k - \rho_{k-1} + 1)/2$ copies of the tree T_k and $(\rho_k - \rho_{k-1} - 1)/2$ copies of the matching M_k to the solution. Goemans and Bertsimas showed that the worst-case ratio θ_2 of the improved tree heuristic is

$$\sum_{k=1}^{p} \frac{f(\rho_k - \rho_{k-1})}{\rho_k}.$$

[5] An Eulerian graph is a connected graph whose nodes all have even degrees. In 1736, in a result that many researchers believe is the origin of graph theory, Euler [Eul36] showed that it is possible to start from any any node in an Eulerian graph, traverse all edges in the graph and return to the start node without traversing any edge twice. As a result, an Eulerian graph is 2-edge connected.

In this expression, $f(l) = \frac{3}{2}l$ if l is even and $f(l) = \frac{3}{2}l + \frac{1}{2}$ if l is odd. For the Steiner k-edge-connected subgraph problem, their algorithm gives a worst-case ratio of $3/2$ when k is even and $3/2 + 1/(2k)$ when k is odd. Again, in proving the worst-case ratio of the improved tree heuristic, Goemans and Bertsimas actually showed that θ_2 is the ratio between the value of the improved tree heuristic's solution and the optimal LP value of the cutset model for the SND problem, thereby bounding the ratio between the optimal IP and optimal LP values for the cutset model. Goemans and Bertsimas also showed that if $\rho_1 = 1$ and $p \geq 2$, the worst-case ratio of the improved tree heuristic improves by a unit to $\theta_2 - 1$.

Building upon the work of Goemans and Bertsimas, Goemans and Williamson [GW92] described an algorithm for network design problems that can be formulated as integer programs with a certain type of cut constraint. This class of problems include the edge-connectivity version of the NDC problem with edge replication. Suppose that r_{st} assumes at most p different nonzero values, $0 = \rho_0 < \rho_1 < \rho_2 < \ldots < \rho_p$. Then their algorithm has a worst-case ratio of θ_1.[6] That is,

$$\theta_1 = \left(2 - \frac{2}{|T|}\right)\left(\sum_{k=1}^{p} \frac{\rho_k - \rho_{k-1}}{\rho_k}\right).$$

In this expression, T is the set of nodes that need to be connected. In proving the worst-case ratio of this heuristic, Goemans and Williamson showed that θ_1 is the ratio between the value of the heuristic's solution and the optimal LP solution to the cutset model for the NDC problem, and therefore a bound on the ratio between the optimal IP and optimal LP values for the cutset model. Although Goemans and Williamson do not show this, arguments similar to those used by Goemans and Bertsimas will reduce the worst-case ratio of the heuristic by 1 to $\theta_1 - 1$ when $\rho_1 = 1$ and $p \geq 2$. A reduced ratio does not apply to the optimal IP to optimal LP ratio of the cutset model.

Williamson, Goemans, Mihail and Vazirani [WGMV93] and Gabow, Goemans and Williamson [GGW93] described an approximation algorithm for a class of problems that include the edge-connectivity version of the NDC problem without edge replication (their algorithm is the *first* approximation algorithm for the edge-connectivity version of the NDC problem without edge replication). They described an approximate version of the primal-dual[7] method for combinatorial optimization problems that can be formulated as

[6] Goemans and Williamson stated a worst-case ratio of $2\left(\sum_{k=1}^{p} \frac{\rho_k - \rho_{k-1}}{\rho_k}\right)$. However, their paper implies this ratio.

[7] A primal-dual method for solving a linear program can be viewed as a special case of the dual-ascent method. It starts with a dual feasible solution and an infeasible solution to the primal that together satisfy the linear programming complementary slackness conditions. At each subsequent iteration, the method alternately improves the feasibility of the primal and the value of the dual solution (thereby providing ascent). The method maintains complementary slackness and terminates when the primal solution becomes

IPs. Using this approximate version of the primal-dual method, they derived a worst-case guarantee for their algorithm. If $r_{\max}$ is the maximum value of r_{st}, their algorithm has a worst-case bound of $2r_{\max}$, and a running time of $\mathcal{O}(|N|r_{\max}(|N|r_{\max} + \sqrt{|E|\log\log|N|}))$. Goemans, Goldberg, Plotkin, Shmoys, Tardos and Williamson [GGP+94] made a simple modification to [WGMV93]'s algorithm and improved the worst-case ratio to $2\mathcal{H}(r_{\max})$, where $\mathcal{H}(r_{\max}) = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{r_{\max}}$. Again, this ratio also bounds the ratio between the optimal IP and optimal LP values to the cutset model for the NDC problem. When edge replication is not permitted, the approach that Goemans and Bertsimas used to improve the worst-case ratio for problems with $\rho_1 = 1$ and $p \geq 2$ (same notation as Goemans and Williamson) does not work. In fact, the algorithm achieves its worst-case ratio for the k-edge-disjoint path problem (a polynomially solvable problem, see Section 2.1.2). The underlying methodology for all these algorithms ([GW92], [WGMV93], [GGW93], and [GGP+94]) applies to many other problems. For example, Ravi and Williamson [RW93] used the approximate version of the primal-dual method to derive the *first* approximation algorithm for the minimum cost k-node-connected spanning subgraph problem.

Balakrishnan, Magnanti and Mirchandani [BMM94a] introduced a new extended formulation for the edge-connectivity version of the NDC problem, that we refer to as the enhanced cutset model. They strengthened the results of Goemans and Bertsimas and Goemans and Williamson (for the edge-connectivity version of the NDC problem) by showing that the improved worst-case ratios of $\theta_1 - 1$, $\theta_2 - 1$ and $\theta_3 - 1$ also apply to the ratio between the values of the heuristics and the optimal LP solution of the enhanced cutset model.

So far, we have described approximation results for edge-connectivity versions of the NDC problem. We now describe some approximation results for the node-connectivity version of the NDC problem, and an approximation result for a uniform cost problem where the underlying graph is not complete.

Node Connectivity: There are very few approximation results for node-connectivity problems. Frederickson and JáJá [FJ81] described a 3-approximation algorithm for the 2-node-connected spanning subgraph problem that runs in $\mathcal{O}(|N|^2)$ time. Khuller and Thurimella [KT93] showed how to improve this algorithm so that it runs in $\mathcal{O}(|E| + |N|\log|N|)$ time. Ravi and Williamson [RW93] described a $2\mathcal{H}(k)$-approximation algorithm for the minimum cost k-node-connected spanning subgraph problem.

Uniform Cost Problem when the Underlying Graph is not Complete: Khuller and Vishkin [KV92] considered the 2-edge- and 2-node-connected spanning subgraph prob-

---

feasible.

lem when the underlying graph is arbitrary, edge replication is prohibited, and all edge costs are 1. This problem is NP-hard [KV92] (recall that when the graph is complete, this problem is polynomially solvable). They described a 3/2-approximation algorithm for the 2-edge-connected spanning subgraph problem and a 5/3-approximation algorithm for the 2-node-connected spanning subgraph problem. They also showed that it is $\mathcal{NP}$-hard to approximate the optimal solution to within an additive constant (i.e., find a solution whose cost is no more than C units more than the optimal solution, where C is a constant).

2.2.2 Cost Function Satisfies Triangle Inequality

Some researchers have concentrated on the version of the NDC problem applied to complete graphs whose edge costs satisfy the triangle inequality (i.e., $c_{ik} \leq c_{ij} + c_{jk}$). In this section, unless specified otherwise, we assume that the edge costs satisfy the triangle inequality and the underlying graph is complete.

Frederickson and JáJá [FJ82] showed that when the cost function satisfies the triangle inequality, a 2-edge-connected graph can be made 2-node connected without increasing (and possibly decreasing) its cost. As a result, any algorithm that applies to the Steiner 2-node-connected spanning subgraph problem also applies to the Steiner 2-edge-connected spanning subgraph problem and vice versa. Consequently, we will not distinguish between the Steiner 2-node-connected and Steiner 2-edge-connected subgraph problems.

By viewing the solution to the NDLC problem as a 2-node-connected network with trees attached to it, Frederickson and JáJá's result extends to the NDLC problem. Thus, for the NDLC problem, the optimal objective value of the edge-connectivity model is equal to the optimal objective value of the node-connectivity model. In other words, the edge-connectivity version of the NDLC problem is equivalent to the node-connectivity version of the NDLC problem. As a result, we do not make distinctions between the edge- and node-connectivity versions of the NDLC problem.

Frederickson and JáJá [FJ82] showed that Christofides' heuristic [Chr76] for the TSP is a 3/2-approximation algorithm for the 2-node-connected spanning subgraph problem. They also described a slight modification of Christofides' heuristic, and showed that it is a 3/2-approximation algorithm whose solution is guaranteed to be no worse (and in some cases better) than Christofides' heuristic.

A similar argument to the one used by Frederickson and JáJá [FJ82], regarding the equivalence of 2-edge-connected and 2-node-connected graphs, establishes the following result.

Proposition 2.2.1 *When costs satisfy the triangle inequality, it is possible to transform, with no increase in cost, any 2-edge-connected graph that contains 3 or more nodes and contains replicated edges, into a 2-node-connected graph without replicated edges. Furthermore,*

this transformation can be done in polynomial time.

As a consequence, Proposition 2.2.1 and Goemans and Bertsimas' result regarding the worst-case ratio of the improved tree heuristic (which for the Steiner 2-edge-connected subgraph problem is Christofides' heuristic without the short-cutting step [assuming there are more than 3 required nodes]) shows that Christofides' heuristic is also a 3/2-approximation algorithm for the Steiner 2-node-connected problem in the model without edge replication. Note that if the problem has exactly two nodes with connectivity requirement 2, Christofides' heuristic will give a solution that contains two parallel edges between the two required nodes. But, when only two nodes have connectivity requirement 2, the problem is the 2-edge-disjoint path problem which is optimally solvable in polynomial time.

Monma, Munson and Pulleyblank [MMP90] studied structural properties of 2-node-connected graphs. They showed that when the cost function satisfies the triangle inequality, in some optimal solution, the degree of each node is either 2 or 3. They also showed that the optimal TSP tour is a 4/3-approximation algorithm for the 2-node-connected spanning subgraph problem. For the Steiner 2-node-connected subgraph problem, they showed that the worst-case ratio of the minimum cost 2-node-connected network over the required nodes (that is a 2-node-connected network on the graph induced by the required nodes) to the minimum cost Steiner 2-node-connected network is 4/3.

Balakrishnan, Magnanti and Mirchandani [BMM94a] used Frederickson and JáJá's result to develop a tree completion heuristic that has a worst-case ratio of 5/2 for the NDLC model without edge replication. Their algorithm first applies Frederickson and JáJá's algorithm on the graph defined by the nodes with connectivity requirement 2. It then sets B to be the set of edges in this solution, and sets T' to be the set of terminal nodes that are incident to edges in B. It continues with Step 2 of the minimum spanning tree heuristic (tree completion stage) to obtain a feasible solution for the problem. Their algorithm assumes at least three nodes have a connectivity requirement of 2. If only two nodes have a connectivity requirement of two, we can apply the tree completion heuristic for the k-path Steiner tree problem to obtain a heuristic with a worst-case ratio of 2. If no node has a connectivity requirement of 2, we obtain the Steiner tree problem for which the minimum spanning tree heuristic gives a worst-case ratio of 2.

We can improve upon Balakrishnan, Magnanti and Mirchandani's result by using Proposition 2.2.1 to provide a 2-approximation algorithm for the NDLC problem. In the previous paragraph we showed how to find a solution whose cost is no more than twice the cost of the optimal solution when two or fewer nodes have a connectivity requirement of 2. If three or more nodes have a connectivity requirement of 2, we apply Goemans and Bertsimas' tree heuristic. The solution of this heuristic contains a 2-edge-connected component with trees attached to it. Using Proposition 2.2.1 we can convert, with no increase in cost, the

2-edge-connected component that has replicated edges to a 2-node-connected component without replicated edges. Since Goemans and Bertsimas' tree heuristic has a worst-case ratio of 2, this heuristic also has a worst-case ratio of 2.

Using a simple doubling argument, Balakrishnan, Magnanti and Mirchandani [BMM94a] showed that optimally solving the NDLC problem (without edge replication) on a graph that contains only the required nodes is no more than twice as expensive as an optimal solution. Their result can be viewed as a generalization of a similar result for the Steiner tree problem. When the costs satisfy the triangle inequality, the solution of Goemans and Bertsimas' tree heuristic contains only edges between required nodes. Therefore, if the number of nodes with connectivity requirement 2 is not equal to 2, Proposition 2.2.1 together with the worst-case bound of Goemans and Bertsimas' tree heuristic constructively proves Balakrishnan, Magnanti and Mirchandani's result.[8]

Bienstock, Brickell and Monma [BBM90] showed that a generalization of Monma, Munson and Pulleyblank's result is not true. They showed that the optimal cost of the k-edge-connected spanning subgraph problem does not equal the optimal cost of the k-node-connected spanning subgraph problem. They also showed that in the optimal solution to the k-edge-connected spanning subgraph each node has degree k or $k+1$. This statement regarding the degree of a node in the optimal solution is also true for the k-node-connected spanning subgraph problem if the number of nodes in the graph is greater than $2k$.

2.3 Exact Algorithms for the NDC Problem

The first exact algorithm for the SND problem is due to Christofides and Whitlock [CW81]. They described a cutting plane algorithm combined with branch and bound. The cutting plane algorithm is based on the cutset formulation. If the solution obtained by the cutting plane algorithm is not integer, they apply a branch and bound procedure. They reported successfully solving problems with more than 100 nodes.

Grötschel, Monma and Stoer [GMS92a, GMS92b, GMS92c, GMS95, Sto92] considered a polyhedral approach for the SND problem. They described valid inequalities and facets for the problem, incorporated them in a cutting plane routine, and reported on successful computations for some real-world problems. Their cutting plane algorithms perform well for low-connectivity problems. The largest NDLC problem they solve (after preprocessing to reduce the size of the graph) has 39 nodes and 86 edges. The cutting plane algorithm required 21 seconds on a SUN 4/50 IPX (a 28.5 MIPS machine) for this problem. The largest SND problem they solve has 461 nodes, 1096 edges and $r_{max} = 3^{[n]}$. The cutting

[8] Actually, it establishes a stronger result. It shows that a heuristic solution on the terminal nodes is no more than twice as expensive as the optimal solution.

plane algorithm required 10122 minutes, on a SUN 4/50 IPX workstation, to solve this problem (by applying some heuristics to reduce the size of this graph they were able to bring down the running time to 426 minutes). Grötschel, Monma and Stoer [Sto92] stated that in their experience the cutset formulation used by Christofides and Whitlock is weak, except for the Steiner k-edge-connected and Steiner k-node-connected spanning subgraph problems when k is large. Conceivably,[9] the test problems of Christofides and Whitlock are of this type.

Chopra [Cho92b] considered a version of the NDLC problem that permits edge replication. Based on a result due to Robbins [Rob39], he directed the network and introduced a directed cut formulation. He then developed a cutting plane algorithm for this formulation. In his computational studies, he solved four types of problems—sparse graphs with Euclidean edge costs, sparse graphs with random edge costs, complete graphs with Euclidean edge costs, and complete graphs with random edge costs. The largest sparse graph problem he solved contains 100 nodes and 200 edges. The cutting plane algorithm required 25 minutes to solve this problem on a VAX 8700 computer. The largest complete graph problem he solved had 50 nodes and 1225 edges, and required a running time of 5.6 minutes (336 seconds) on a VAX 8700 computer. Chopra reported that the lower bounds provided by solving the LP relaxation of the directed cut formulation were consistently within 1 percent of the optimal objective value.

Boyd and Hao [BH93] studied the polytope associated with the minimum cost 2-edge-connected spanning subgraph problem. They introduced a class of valid inequalities for this polytope and described necessary and sufficient conditions for these valid inequalities to define facets. Although they do not provide computational results, they indicate that these inequalities will be useful in a cutting plane method.

2.4 Directed Network Design Problems with Connectivity Requirements

For the most part, the problems we consider in this thesis are defined on undirected graphs. In some telecommunications applications, the medium transporting data cannot carry traffic in both directions, and so the connectivity requirements are specified on directed graphs (digraphs). In his Ph.D. thesis, Dahl [Dah92] described several real-world applications of the directed network design problem with connectivity requirements (DNDC). Formally, we can describe the problem as follows.

[9]The original paper by Christofides and Whitlock, as cited in [CW81], does not appear to be published.

Directed Network Design Problem with Connectivity Requirements: Given a digraph $D = (N, A)$ with node set N, arc set A, cost vector $\mathbf{c} \in \mathcal{R}_+^{|A|}$ on the arcs A, and a $|N| \times |N|$ matrix $\mathbf{R} = [r_{ij}]$ that prescribes the number of arc- or node-disjoint paths[10] needed between each pair of nodes, select a set of arcs that satisfy these requirements at minimum cost.

We now briefly describe other well-known network design problems, and show how to represent them as DNDC problems.

Minimum Cost Branching Problem: Given a root node r, we wish to design a minimum cost network that has a directed path from the root node to every other node. The minimum cost branching problem is a DNDC problem in which $r_{ri} = 1$ if $i \neq r$ and is 0 otherwise.

Minimum Cost k-Branching Problem: Given a root node r, we wish to design a minimum cost network that has k arc-disjoint branchings rooted at node r. Edmonds [Edm73] showed that a graph contains k arc-disjoint branchings rooted at a given node r if and only if it contains k arc-disjoint directed paths from node r to every other node. Consequently, the minimum cost k-branching problem is a DNDC problem in which $r_{ri} = k$ if $i \neq r$ and is 0 otherwise. Similarly we could formulate Steiner versions of the problem.

Disjoint Path Problems: The k-arc-disjoint path problem requires that we find at minimum total cost k arc-disjoint paths between a prespecified pair, say s and t, of nodes. To specify the problem as a DNDC problem, we set $r_{st} = k$ and set all other entries in the connectivity matrix $\mathbf{R}$ to zero. Similarly, for the k-node-disjoint path problem, we wish to find at minimum total cost k node-disjoint paths between a prespecified pair of nodes (say s and t). To formulate this problem as a DNDC problem, we set $r_{st} = k^{[n]}$ and set all other entries in the connectivity matrix $\mathbf{R}$ to zero.

Minimum Cost k-Arc-Connected Spanning Digraph Problem: A digraph is *k-arc connected* if between every pair of nodes it contains at least k arc-disjoint paths.[11] The minimum cost k-arc-connected spanning digraph problem is a DNDC problem in which $r_{st} = k$ for every pair of nodes in N.

---

[10] This is a generalization of the problem described in Dahl [Dah92]. He permits either arc-connectivity requirements or node-connectivity requirements in the matrix $\mathbf{R}$ but not both.

[11] A digraph is *strongly connected* if it contains a directed path from each node to every other node. A digraph is k-arc connected if deleting fewer than k arcs creates a strongly connected digraph. Again, Whitney [Whi32] showed that a digraph is k-arc connected if and only if it contains at least k arc-disjoint paths between each pair of nodes. Similarly, a digraph is k-node connected if deleting fewer than k nodes creates a strongly connected digraph. Whitney's result implies that a digraph is k-node connected if and only if it contains at least k node-disjoint paths between each pair of nodes.

Minimum Cost k-Node-Connected Spanning Digraph Problem: A digraph is *k-node connected* if between every pair of nodes it contains at least k node-disjoint paths. The minimum cost k-node-connected spanning digraph problem is a DNDC problem in which $r_{st} = k^{[n]}$ for every pair of nodes in N. Similarly, we can formulate Steiner versions of these problems as DNDC problems.

2.4.1 Polynomially Solvable Cases

Like the NDC problem, various special versions of the DNDC problem with specialized costs, connectivity requirements, or graphical structure are polynomially solvable.

Digraph Augmentation Problems

In these problems, the arc costs are either 0 or 1. In addition, unless otherwise specified the underlying digraph is complete and the model permits arc replication.

Uniform Cost Problems: Fulkerson and Shapley [FS71] solved the problem of constructing a k-arc-connected spanning digraph on $|N|$ nodes with the minimum number of arcs. It is interesting to note that the Steiner version of this problem (i.e. when some nodes are not required to be in the k-arc-connected network) is $\mathcal{NP}$-hard [Fra92].

Arc-Connectivity Augmentation: Eswaran and Tarjan [ET76] solved the problem of constructing 1-arc-connected spanning digraphs (strongly connected digraphs). Kajitani and Ueno [KU86] solved the problem of constructing k-arc-connected spanning digraphs when the starting digraph is a branching. Frank [Fra92] generalized these results by solving the augmentation problem for the k-arc-connected spanning digraph problem.

Node-Connectivity Augmentation: Eswaran and Tarjan [ET76] showed how to solve the 1-node-connected spanning digraph problem. (Note that a 1-arc-connected digraph is also a 1-node-connected digraph and vice versa. Consequently, the 1-node-connected spanning digraph problem is identical to the 1-arc-connected spanning digraph problem.) As cited in [Fra94], several other problems are polynomially solvable. Masuzawa, Hagihara and Tokura showed how to solve the k-node-connected spanning digraph problem when the starting digraph is a branching. Frank and Jordán described an algorithm for the 3-node-connected spanning digraph problem when the starting digraph is a 2-node-connected spanning digraph. Frank and Jordán described a necessary and sufficient condition on the number of arcs to add to an arbitrary digraph to make it k-node connected. Unfortunately, this result is of a theoretical nature and does not provide a combinatorial algorithm for the k-node-connected spanning digraph augmentation problem. The result can be used

to provide an algorithm for the problem by using the ellipsoid algorithm as a subroutine (which is not very practical).

Restricted Connectivity Requirements

The following restricted connected requirements are optimally solvable.

- The minimum cost branching problem. Edmonds [Edm67] described an algorithm for this problem, which we will see in Chapter 3.
- The k-arc-disjoint or k-node-disjoint path problem on a digraph. This problem can be formulated as a minimum cost flow problem, which is polynomially solvable. Suurbaale [Suu74] has devised a specialized algorithm for this problem.
- The minimum cost k-branching problem. This is a matroid intersection problem.[12] Consequently, it can be solved optimally [Edm79]. The most efficient implementation of an algorithm for the minimum cost k-branching algorithm is due to Gabow [Gab91].
- The node-disjoint version of the k-branching problem. In this problem we wish to find k node-disjoint directed paths between the root node and every other node in the graph. Using a tricky reduction to submodular flows, Frank and Tardos [FT89] described an algorithm for this problem.

Restricted Graphs: Series-Parallel Graphs

Prodon, Liebling and Gröflin [PLG85] described a dicut model for the Steiner branching problem and showed that it provides a complete polyhedral description (of the dominant) on series-parallel graphs (a digraph is called series-parallel if the graph obtained by replacing the directed arcs by undirected edges is series-parallel).

Chopra [Cho92a] described a dicut model for the minimum cost 1-arc-connected spanning digraph problem and showed that it provides a complete polyhedral description (of the dominant) of the problem on series-parallel graphs.

[12] Edmonds [Edm70] showed how to formulate the problem as the intersection of two matroids: (i) a partition matroid in which a set is independent if it contains at most k arcs directed into any node $i \neq r$; and (ii) the k-fold sum of a graphic matroid. A set is independent in a graphic matroid if it contains no cycle. Matroid theory tells us that the matroid sum of k matroids defined on the same set S is a matroid in which the independent sets are all subsets of S that can be partitioned into k subsets, the ith subset being independent in the ith matroid. If the k matroids are identical, their matroid sum is called a k-fold sum. Consequently, a set is independent for the k-fold sum of a graphic matroid if it can be partitioned into k subsets, none of which contain a cycle.

2.4.2 Heuristics

The only heuristic that we are aware of for a DNDC problem is Wong's algorithm for the Steiner branching problem (which we shall see in Chapter 3).

Heuristics for the DNDC problem appear not to have generated much research and so this topic seems to be an open area for research.

2.4.3 Exact Algorithms

Chopra [Cho92b] considered the minimum cost 1-arc-connected spanning digraph problem. He studied the problem from a polyhedral point of view and described several classes of valid inequalities and facets for the dicut model.

Dahl [Dah92] studied a more general version of the DNDC problem. He presented two models for the DNDC problem: one, a natural directed cut (dicut) model and the other a directed flow formulation. For the dicut model, he presented several classes of strong valid inequalities and described a cutting plane solution algorithm. He investigated the performance of the cutting plane algorithm on two types of problems: the Steiner k-branching problem and the k-node-connected spanning digraph problem.

In computational experiments, reported in his thesis, Dahl solved several Steiner k-branching problems on sparse graphs. The largest Steiner k-branching problem contained 178 nodes and 880 arcs with $k = 1$. His cutting plane algorithm required 126 seconds to solve this problem on a SUN SPARC SLC (a 12 MIPS machine) workstation.

Dahl [Dah94] also solved several Steiner 2-arc-connected digraph problems with the cutting plane algorithm. The largest problem he solved had 70 nodes and 2386 arcs, and the cutting plane algorithm required 1194 seconds on a Solbourne workstation.[13]

Dahl's results indicate that the dicut model and the directed flow model are strong formulations for both problems.

2.5 Concluding Remarks

In this chapter we have reviewed previous work on the NDC and the DNDC problem. The NDC problem has attracted considerable attention in recent years (of the 86 papers cited in this chapter, 40 are from the past five years, i.e., 1990–94).

We categorized research efforts on the NDC problem into three main areas: polynomially solvable cases, heuristics, and exact algorithms. Early research efforts on the NDC problem focused on characterizing polynomially solvable cases. More recently, researchers

[13]Dahl does not indicate the speed of the workstation. A comparable workstation (same chip) is the Sun SPARCstation 10, which is a 90 MIPS machine.

have focused on devising approximation algorithms for the NDC problem (or its special cases) and have used a polyhedral approach to optimally solve this problem.

There are several potential avenues for future research. The research community has shown considerable interest in finding better approximation algorithms for the NDC problem, and a significant number of researchers in the computer science community are working on this problem. DNDC problems have attracted little attention from the research community. Designing and analyzing heuristics for them seems a promising avenue of research. (It is interesting to note that no approximation algorithms are known for the directed connectivity problems.)

As is evident from the discussion in this chapter, other than for the Steiner tree and Steiner branching problem, to our knowledge, research on dual-ascent algorithms for the NDC and DNDC problem appears to be quite limited. The success of the dual-ascent method for the Steiner tree problem suggests that this approach might be a promising avenue for future research.

Much of the research on the NDC problem has focused on special cases of the problem, and on the edge-connectivity version of the NDC problem. For example, Goemans et al.'s [GGP+94] approximation algorithm does not apply to the node-connectivity version of the problem. Further research is needed to develop models and algorithms for the node-connectivity version of the NDC problem. In the polyhedral approach, again much of the work focuses on special cases of the NDC problem. Typically, strong formulations for optimization problems lead to better solution algorithms. One important research area is to develop better (stronger) models for the NDC problem.

In this thesis we address some of the these potential research directions. We design dual-ascent algorithms for the NDC and DNDC problem (and their special cases). These algorithms apply to the most general case of the NDC and DNDC problem. Throughout the course of this thesis, we address modeling issues. We discuss alternative modeling approaches and show how to strengthen models for the NDC problem.

Chapter 3

Dual-Ascent for Network Design

In the first part of this chapter, we introduce some of the basic ideas of dual-ascent as applied to network design, using the minimum spanning tree (MST) problem as an example. Since the MST problem is a NDC problem, we can model it using either of the two formulations presented in Chapter 1. We first present slightly different versions of the cutset and flow formulation. Next we show how to strengthen the flow formulation by directing the model. As we will see in several points in this thesis, directing an undirected model often provides better models for network design problems. Finally, we use the improved formulation to develop a dual-ascent algorithm that also provides a feasible tree solution to the problem. The arguments in this chapter show that this algorithm also solves the more general minimum cost branching problem. This dual-ascent algorithm, although derived in a different manner, is identical to a combinatorial algorithm developed by Edmonds [Edm67].

As an aid to our later development, we next present a Lagrangian duality interpretation of dual-ascent. This interpretation will will prove invaluable in deriving a dual-ascent algorithm for the NDC problem (see Chapter 4).

Finally, we show how to extend the ideas underlying the branching problem to the Steiner branching problem. As a result, we derive Wong's dual-ascent algorithm [Won84]. Our derivation provides a different interpretation of the *root component rule* in Wong's algorithm. The ideas presented in this section will be useful later as we develop a dual-ascent algorithm for the network design problem with low-connectivity requirements (see Chapter 6).

Much of the material presented in this chapter is fairly well-known (for example, see Magnanti and Wolsey [MW95] for a survey on formulations and algorithms for tree optimization problems). This chapter contributes to the thesis (and literature) in several ways. First, it synthesizes results for the (spanning tree and) branching problem in the context of dual-ascent. Second, it provides a different way to interpret Wong's algorithm. This interpretation will prove to be useful when we consider the network design problem with

low connectivity requirements. Third, the chapter presents several efficient algorithms for implementing Wong's algorithm.

3.1 Formulations for the Minimum Spanning Tree Problem

When specialized, the cutset formulation (1.1) for the NDC problem becomes a well-known integer programming model for the minimum spanning tree problem. This formulation has the decision variables

$$x_{ij} = \begin{cases} 1 & \text{if edge } \{i,j\} \text{ is in the minimum spanning tree;} \\ 0 & \text{otherwise.} \end{cases}$$

Cutset formulation for the MST problem:

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{3.1a}$$

$$\text{subject to:} \quad \sum_{\{i,j\} \in \delta(T)} x_{ij} \geq 1 \quad \text{for all } T,\ \phi \subset T \subset N; \tag{3.1b}$$

$$x_{ij} \geq 0 \quad \text{and integer.} \tag{3.1c}$$

This formulation is based upon the fact that any feasible tree must contain at least one edge joining every set T and its complement. Notice, the model imposes no capacity constraint on x_{ij}. Since all costs are nonnegative, any feasible integer solution with $x_{ij} > 1$ can be converted to one with $x_{ij} = 1$ without increasing its cost and yet retaining feasibility. Consequently, model (3.1) always has an optimal solution with unreplicated edges, and thus requires no capacity constraint on the x_{ij} variables.

Alternatively, we can formulate the problem with flow variables to impose the connectivity requirements. As a notational convenience, we assume the nodes of the graph are numbered $0, 1, \ldots, |N| - 1$. Arbitrarily select one of the nodes in the graph, which we call the root node and, without loss of generality, assume that it is node 0. For each node i, other than the root node, create a commodity i with the origin as the root node and node i as the destination node. Since a minimum spanning tree contains a path between every pair of nodes, it has a path from the root node to every other node. Thus, if we can send flow on an edge only if it belongs to the minimum spanning tree, then in a minimum spanning tree it is possible to send one unit of commodity i from the root node to node i for every node i other than the root node in the graph. Consequently, we can formulate the minimum spanning tree problem in the following equivalent way.

Undirected flow formulation for the MST problem:

$$\text{Minimize} \quad \sum_{\{i,j\}\in E} c_{ij}x_{ij} \tag{3.2a}$$

$$\text{subject to:} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k = \begin{cases} -1 & \text{if } i = 0; \\ 1 & \text{if } i = k; \quad \text{for all } i \in N \text{ and } k \in K \\ 0 & \text{otherwise;} \end{cases} \tag{3.2b}$$

$$\left.\begin{matrix} f_{ij}^k \\ f_{ji}^k \end{matrix}\right\} \leq x_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{3.2c}$$

$$f_{ij}^k, f_{ji}^k \geq 0 \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{3.2d}$$

$$x_{ij} \geq 0 \qquad \text{and integer; for all } \{i,j\} \in E. \tag{3.2e}$$

Notice that this formulation is similar to Formulation (1.2), except that it has fewer commodities and does not impose a capacity constraint on the edge choice variables x_{ij}.

The celebrated max-flow min-cut theorem [FF56] is an invaluable tool for comparing formulations of network design models.

Theorem 3.1.1 (Max-Flow Min-Cut Theorem) *The maximum value of the flow from an origin node s to a destination node t in a capacitated network equals the minimum capacity among all s–t cuts.*

The max-flow min-cut theorem implies that the previous two formulations are equivalent. That is, if we consider x_{ij} as edge capacities for the flow formulation, then x_{ij} is feasible for this formulation if and only if it is feasible for the cutset formulation. Moreover, since the flow costs are zero, the objective value is the same in both models.

Our discussion shows that the projection of the feasible region of the LP relaxation of the undirected flow formulation for the MST problem onto the space of the $\mathbf{x}$ variables (that is, $\{\mathbf{x} : (\mathbf{x}, \mathbf{f})$ is feasible for the LP relaxation of model (3.2)$\}$) is the feasible space of the LP relaxation of the cutset formulation.

Notice that the cutset formulation has an exponential number of constraints while the undirected flow formulation has a polynomial number of constraints (as a function of the number of edges and number of nodes). Therefore, the undirected flow formulation is a compact formulation that is equivalent to the exponentially sized cutset formulation.

However, solving the LP relaxation of either the undirected flow or cutset formulation need not provide an optimal solution. For example, in Figure 3-1 the LP relaxation of the undirected flow model provides a solution value of 2.5 whereas the optimal solution has cost 4.

Why does the solution in Figure 3-1 give us fractional solutions? Notice that the optimal solution sends flow on edge $\{2,3\}$ in both directions. However, in any feasible minimum

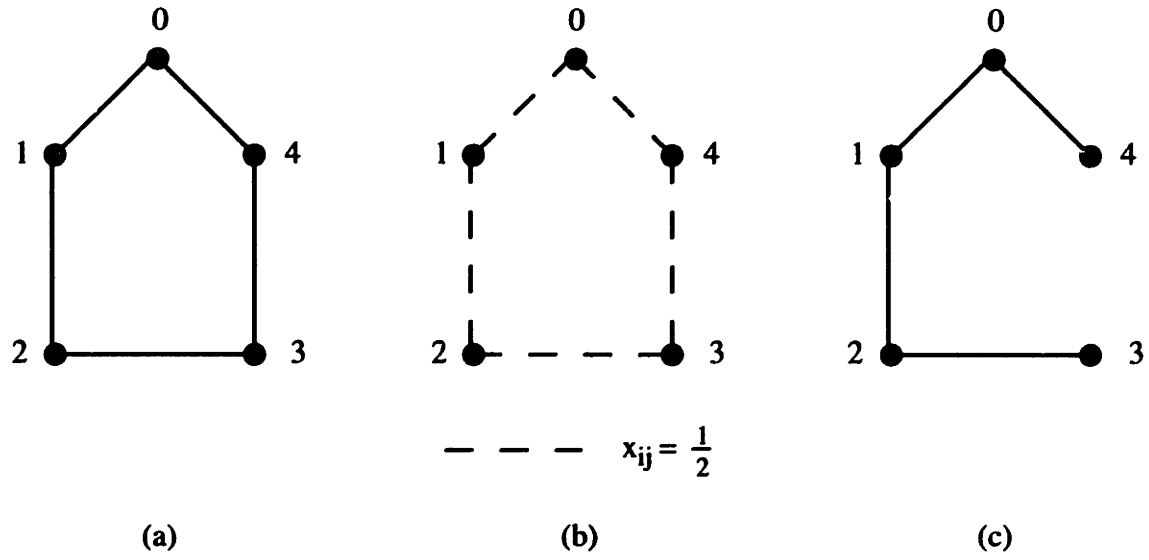


Figure 3-1: The LP relaxation of the minimum spanning tree has a fractional solution. (a) Graph G with all edge costs equal to 1. (b) Fractional solution. Send 1 unit of flow from node 0 to any other node by sending by 1/2 a unit of flow clockwise and 1/2 a unit of flow anticlockwise around the cycle. (c) An optimal minimum spanning tree.

spanning tree, we can send flow from the root node to every other node without sending flow on any edge in both directions. This observation shows that we can replace inequality (3.2c) in the undirected flow formulation for the MST problem by the stronger valid inequality $f_{ij}^k + f_{ji}^h \leq x_{ij}$ for all commodity pairs k and h (as shown by Balakrishnan, Magnanti and Wong [BMW89] and Martin [Mar86]). Doing so, we obtain the following formulation:

Improved undirected flow formulation for the MST problem:

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{3.3a}$$

$$\text{subject to:} \quad \sum_{j \in N} f_{ji}^k - \sum_{l \in N} f_{il}^k = \begin{cases} -1 & \text{if } i = 0; \\ 1 & \text{if } i = k; \quad \text{for all } i \in N \text{ and } k \in K \\ 0 & \text{otherwise;} \end{cases} \tag{3.3b}$$

$$f_{ij}^k + f_{ji}^h \leq x_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and } k, h \in K \tag{3.3c}$$

$$f_{ij}^k, f_{ji}^k \geq 0 \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{3.3d}$$

$$x_{ij} \geq 0 \qquad \text{and integer; for all } \{i,j\} \in E. \tag{3.3e}$$

We can also formulate the MST problem on a digraph. To convert the undirected graph to a digraph $D = (N, A)$, we replace each undirected edge $\{i,j\}$ by two directed arcs (i,j)

and (j,i), each with the same cost as the undirected edge $\{i,j\}$. In the directed flow formulation, let y_{ij} be 1 if arc (i,j) is in the design and be 0 otherwise. The directed flow formulation is

Directed flow formulation for the MST problem:

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}y_{ij} \tag{3.4a}$$

$$\text{subject to:} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k = \begin{cases} -1 & \text{if } i = 0; \\ 1 & \text{if } i = k; \quad \text{for all } i \in N \text{ and } k \in K \\ 0 & \text{otherwise;} \end{cases} \tag{3.4b}$$

$$f_{ij}^k \le y_{ij} \qquad \text{for all } (i,j) \in A \text{ and } k \in K \tag{3.4c}$$

$$f_{ij}^k \ge 0 \qquad \text{for all } (i,j) \in A \text{ and } k \in K \tag{3.4d}$$

$$y_{ij} \ge 0 \qquad \text{and integer for all } (i,j) \in A. \tag{3.4e}$$

For the minimum spanning tree problem, the improved undirected flow formulation (3.3) and the directed flow formulation (3.4) are equivalent. That is, they provide the same optimal objective value when solved as LPs. To see this result, note that we can convert any optimal solution $(\mathbf{y},\mathbf{f})$ to the LP relaxation of the directed flow formulation into a feasible solution, with the same cost, to the LP relaxation of the improved undirected flow formulation by setting $x_{ij} = y_{ij} + y_{ji}$, and $f_{ij}^k = f_{ij}^k$, for all i, j, and k. Similarly, to convert an optimal solution to the LP relaxation of the improved undirected flow formulation to a feasible solution, with the same cost, to the LP relaxation of the directed flow formulation set $y_{ij} = \max_{k\in K}\{f_{ij}^k\}$, and $y_{ji} = x_{ij} - y_{ij}$. The constraint $f_{ij}^k + f_{ji}^h \le x_{ij}$ implies that $\max_{k\in K}\{f_{ji}^k\} \le y_{ji}$. Thus, we satisfy all the constraints of the directed flow formulation without any change in the objective function value.

The directed flow formulation actually models the more general minimum cost branching problem. Recall that in a minimum cost branching problem, we are given a digraph $D = (N,A)$ and a root node. We need to find a minimum cost network that has a directed path from the root node to every other node. Note that in the minimum cost branching problem, the arc costs need not be symmetric. Moreover, no feasible branching will contain arcs directed into the root. Therefore, without loss of generality, we assume the digraph D contains no arcs directed into the root node.

3.2 Dual-Ascent for the Minimum Spanning Tree Problem

We will describe an algorithm that solves the dual to the LP relaxation of the directed flow formulation for the branching problem. From this algorithm, we will also obtain a feasi-

ble solution to the branching problem that has the same objective value, thereby proving (i) the algorithm solves the minimum cost branching problem, and (ii) for all nonnegative cost vectors, the LP relaxation of the directed flow formulation for the branching problem provides integer optimal solutions (later we show that the algorithm also solves the branching problem for arbitrary cost vectors).

Dual of directed flow formulation for the MST problem:

$$\text{Maximize} \quad \sum_{k \in K} v_k^k \tag{3.5a}$$

$$\text{subject to:} \quad v_j^k - v_i^k \leq w_{ij}^k \quad \text{for all } (i,j) \in A \text{ and } k \in K \tag{3.5b}$$

$$\sum_{k \in K} w_{ij}^k \leq c_{ij} \quad \text{for all } (i,j) \in A \tag{3.5c}$$

$$w_{ij}^k \geq 0 \quad \text{for all } (i,j) \in A \text{ and } k \in K. \tag{3.5d}$$

In this dual linear program, v_j^k is the dual variable for the flow balance constraint (3.4b) at node j for commodity k (we refer to the v_j^k variables as *node potentials*), and w_{ij}^k is the dual variable for constraint (3.4c). Since for each commodity k, one flow balance constraint in the primal problem (3.4) is redundant, we can choose an arbitrary value for one dual variable v_j^k. We set $v_0^k = 0$ for each commodity $k \in K$.

One way to provide a lower bound on the cost of the optimal branching is to find a dual feasible solution. Notice that for given values of the **w** variables, the dual problem separates by commodity. The subproblem corresponding to commodity k is the dual to the (directed) shortest path problem from node 0 to node k with arc lengths w_{ij}^k. If we prescribe feasible values for the **w** variables, it is easy to calculate the shortest path lengths for each subproblem and thereby obtain a dual solution for that choice of **w**.

To maintain dual feasibility, we need to ensure the solution satisfies constraints (3.5c) and 3.5d. Let s_{ij} be the slack in constraint (3.5c) (i.e., $s_{ij} = c_{ij} - \sum_{k \in K} w_{ij}^k$). If $s_{ij} = 0$, constraint (3.5c) is tight and we say the arc (i,j) is *tight*. We refer to the network defined by the tight arcs as the *auxiliary network*.

How do we select values for the **w** variables? If the auxiliary network contains no directed path from the root node to any node k, then by Menger's theorem,[1] some directed

[1]The following directed version of Menger's theorem is similar to the undirected version as described in Theorem 1.2.1:

Theorem 3.2.1 (Menger) *In a directed graph,*

1. *The maximum number of arc-disjoint paths from node s to node t equals the minimum number of arcs whose removal from the digraph eliminates all the paths from node s to node t.*
2. *If the digraph does not contain the arc (s,t), the maximum number of node-disjoint paths from node s to node t equals the minimum number of nodes whose removal from the digraph eliminates all the*

cut (dicut) between the root and node k contains no arcs. In this case, we can improve the dual objective by simultaneously increasing w_{ij}^k for all the arcs (i,j) in this dicut in the original network and thus the length of the shortest path from the root to node k. The maximum increase is constrained by the minimum slack among the arcs in the dicut, since as we increase w_{ij}^k for all arcs in the dicut, the slack s_{ij} decreases until one of the arcs becomes tight. At this point, we cannot increase the w_{ij}^k values for all arcs in the dicut while maintaining dual feasibility. We refer to a dicut that contains no tight arc as a *loose dicut* and the procedure of increasing the w_{ij}^k values for all arcs in a loose dicut, until the slack on one of the arcs becomes zero, as a *basic dual-ascent step.*

Consider the example shown in Figure 3-2a. The digraph provides dual variable values for a commodity whose origin is node a and destination is node d. (For simplicity we have dropped the superscript k.) The bold arcs are in the auxiliary network. The figure shows the values of v_i^k next to the nodes and the values of w_{ij}^k and s_{ij} next to the arcs. The a–d dicut, with the arcs (b,c) and (g,f), contains no bold arcs, and is therefore a loose dicut. The maximum increase in w_{ij}^k for this a–d dicut is the minimum slack, of value 2 on arc (g,f). After we have incremented the w_{bc}^k and w_{gf}^k values by 2, the new shortest path length to node d, i.e. v_d^k, increases by 2 units from 5 to 7. The new auxiliary network contains the additional arc (g,f). As shown in Figure 3-2b, we have identified a new loose a–d dicut.

The dual-ascent algorithm we develop identifies a commodity k satisfying the property that the auxiliary network contains no path from the commodity's origin (node 0) to its destination (node k). The algorithm then finds a loose 0–k dicut and performs a basic dual-ascent step. Thus at each step, we examine the auxiliary network and see if it contains a branching, i.e., a feasible solution to the primal problem. If it does not, we find a violated constraint and improve the dual solution. When the algorithm terminates, it has generated both primal and dual-feasible solutions. These basic steps constitute the *essence* of most of the algorithms we describe in this thesis.

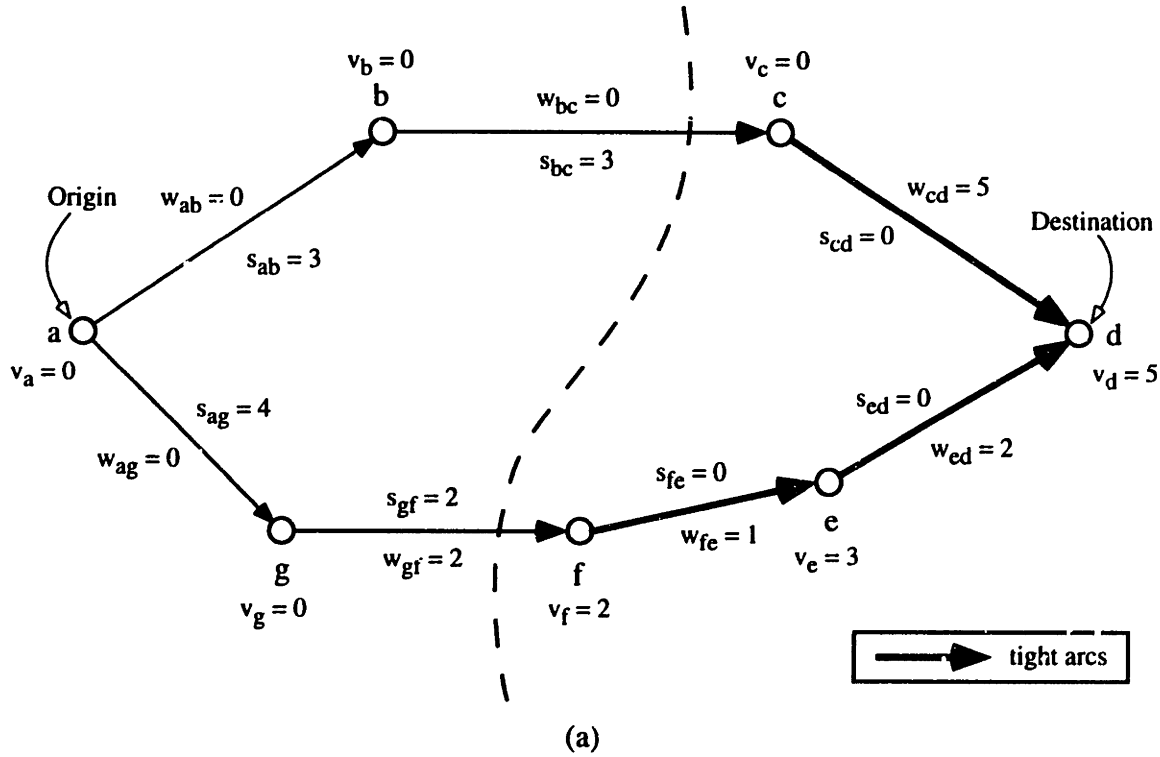
At the outset of our discussion we assume all arc costs are strictly positive (later we relax this assumption). As a result, the auxiliary network is initially empty. For each commodity 1 to $|N| - 1$, set $N(k) = \{k\}$. Then for each commodity k, $\delta^-(N(k))$ is a dicut. Since we have assumed all costs are positive, for each commodity k the dicut $\delta^-(N(k))$ is a loose dicut. Therefore, we can perform a basic dual-ascent step for each commodity k. After

---

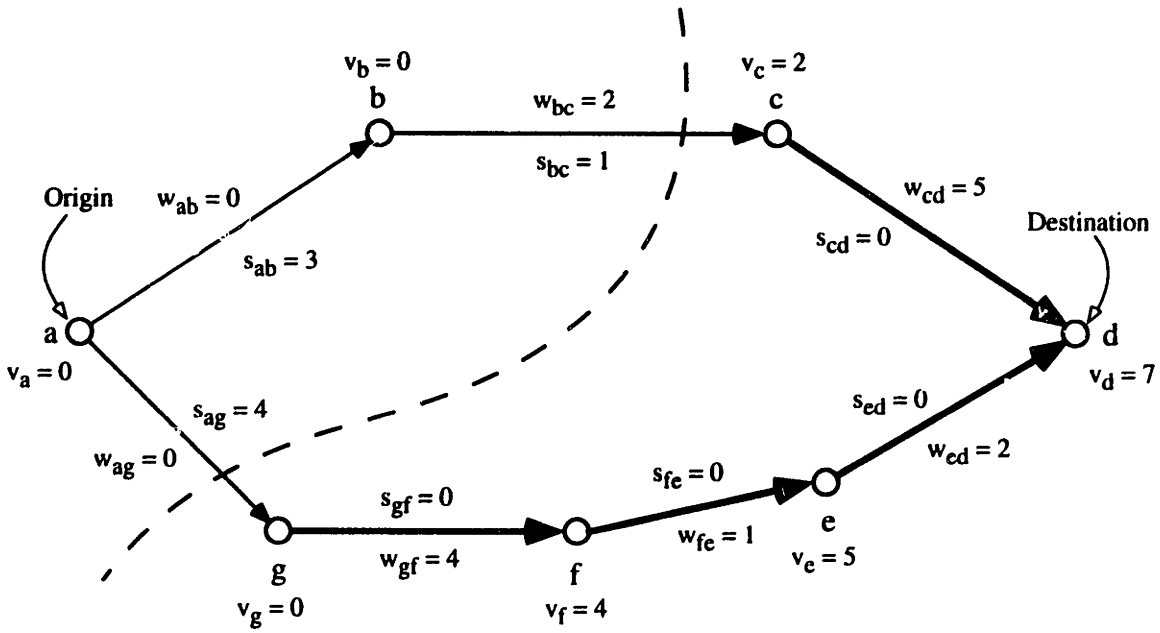
paths from node s to node t.

Similarly, there is a directed version of the max-flow min-cut theorem:

Theorem 3.2.2 (Max-Flow Min-Cut) *The maximum value of the flow from an origin node s to a destination node t in a capacitated directed network equals the minimum capacity among all s-t dicuts.*



(a)



(b)

Figure 3-2: An example of a basic dual-ascent step.

doing so, we examine the auxiliary network.

Recall that a set of nodes S in a digraph is strongly connected if the digraph contains a directed path between every (ordered) pair of nodes in S. A strongly connected component is a maximal strongly connected set of nodes. That is, if we add any node to this set, the nodes in the set will no longer be strongly connected.

Claim 3.2.3 *If a digraph contains no branching, then some strongly connected component has no incoming arc.*

Proof: Suppose not. Note that each node in the digraph must have an incoming arc; otherwise, this node defines a strongly connected component with no incoming arc. Contract each strongly connected component in the digraph into a single node. By assumption, every node, other than the root, in the reduced digraph has an incoming arc. Any such digraph must contain either a branching or a directed cycle C. This is true because, if the digraph does not contain a branching, it has a dicut $\delta^-(T)$, with the root $\notin T$, containing no arcs. Since every member of T has an incoming arc, it contains a cycle.

A branching on the reduced digraph can be extended to a branching in the digraph. Therefore, the reduced digraph must contain a cycle C. But, the nodes on a directed cycle belong to the same strongly connected component and so the reduced digraph has a strongly connected component. This result, however, implies that the nodes from the original network that we have contracted into nodes in C are all strongly connected, contradicting our assumption that each node in C corresponds to a strongly connected component in the original network. ■

We apply this result to the auxiliary network. For each strongly connected component with no incoming arc, we select any node i that it contains, and set $N(i)$ to be the set of all nodes in this strongly connected component. Corresponding to every strongly connected component with no incoming right arc, we have identified a commodity (commodity i) and its associated loose dicut $\delta^-(N(i))$. We perform the basic dual-ascent step for each loose dicut. We then repeat the procedure, terminating when every strongly connected component has an incoming arc.

At termination, the auxiliary network contains a branching. However, it could have more edges than required in a branching. Therefore, we can delete some arcs and still obtain a feasible solution. We sequentially consider the arcs in the opposite order that they became tight (i.e., opposite order that the algorithm adds them to the auxiliary network). We refer to this deletion scheme as the LIFO dropping rule.

The dual-ascent algorithm for the branching problem can be described as follows.

Dual-ascent algorithm for the branching problem

1. Initialization. Set
   (a) $w_{ij}^k = 0$ for all $(i,j) \in A$ and $k \in K$,
   (b) $s_{ij} = c_{ij}$ for all $(i,j) \in A$,
   (c) $Z = \{(i,j) : (i,j) \in A; s_{ij} = 0\}$ (the set of zero-slack arcs),
   (d) $v_i^k = 0$ for all $i \in N$ and $k \in K$,
   (e) $L = \sum_{k=1}^{|N|-1} v_k^k$ (the lower bound).
2. While the digraph $D_Z = (N, Z)$ induced by the arcs in Z contains a strongly connected component with no incoming arc
   (a) Select a node k that is in this strongly connected component, and set $N(k)$ to be the set of nodes in this strongly connected component.
   (b) Find minimum slack.
   - $\epsilon = \min_{\{(i,j):(i,j)\in\delta^-(N(k))\}} s_{ij}$.

   (c) Update relevant dual variables, slacks, the auxiliary network D_Z, and the lower bound L.
   - $w_{ij}^k = w_{ij}^k + \epsilon \quad \forall\ (i,j) \in \delta^-(N(k))$,
   - $s_{ij} = s_{ij} - \epsilon \quad \forall\ (i,j) \in \delta^-(N(k))$,
   - $v_i^k = v_i^k + \epsilon \quad \forall\ i \in N(k)$,
   - $L = L + \epsilon$,
   - $Z = Z \cup \{(i,j) : (i,j) \in \delta^-(N(k)) \text{ and } s_{ij} = 0\}$.
3. The network defined by the arcs in Z (i.e., D_Z) contains a minimum cost branching. Construct a branching by applying the LIFO dropping rule on D_Z.

When it has terminated, the dual-ascent algorithm has in hand feasible values $\mathbf{v}, \mathbf{w}$ for the dual linear program (3.5). The algorithm has also found a feasible solution $(\mathbf{y}, \mathbf{f})$ to the primal problem (3.4), defined as follows. Let $y_{ij} = 1$ if the final auxiliary network contains arc (i,j) and set $y_{ij} = 0$ otherwise. Set $f_{ij}^k = 1$ if arc (i,j) lies in the unique path in the auxiliary network connecting the root node and node k. Set $f_{ij}^k = 0$ otherwise.

We now show the dual-ascent algorithm solves the branching problem.

Theorem 3.2.4 *The dual-ascent algorithm solves the minimum cost branching problem.*

Proof: To prove this result, we will first establish a simple property concerning the structure of the solution the algorithm generates.

Claim 3.2.5 *The solution determined by the dual-ascent algorithm satisfies the following property: every loose dicut used (in a basic dual-ascent step) contains exactly one directed arc.*

Proof: Since we perform the LIFO dropping rule, we will consider dropping other arcs before we consider dropping the tight arcs in any strongly connected component. Feasibility implies that the strongly connected component has at least one incoming arc. There is at most one because our dropping scheme will eliminate all but one arc directed into the strongly connected component. ■

Consider the complementary slackness conditions for the linear programming relaxation of the directed flow formulation (3.4):

$$y_{ij} > 0 \Longrightarrow \sum_{k \in K} w_{ij}^k = c_{ij};$$

$$f_{ij}^k > 0 \Longrightarrow v_j^k - v_i^k = w_{ij}^k;$$

and

$$w_{ij}^k > 0 \Longrightarrow f_{ij}^k = y_{ij}.$$

By design, the algorithm satisfies the first two (so called primal) complementary slackness conditions. The solution also satisfies the dual complementary slackness condition $w_{ij}^k(y_{ij} - f_{ij}^k) = 0$. Consider any loose dicut that the algorithm uses in a dual-ascent step. If the algorithm chooses commodity k while considering this dicut, then by Claim 3.2.5 the final solution has exactly one arc, say (s,t), in the dicut and the flow from the root to node k must flow across this arc. Therefore, for all arcs in the dicut $f_{ij}^k = y_{ij}$ (with $f_{st}^k = y_{st} = 1$ and $f_{ij}^k = y_{ij} = 0$ otherwise). Since the algorithm increases w_{ij}^k from value 0 only for loose dicuts, this observation implies that the final solution satisfies the dual complementary slackness conditions as well.

We have exhibited a feasible primal solution and a feasible dual solution that satisfy complementary slackness. Therefore, our primal and dual solutions are optimal. ■

Now that we know the dual-ascent algorithm finds an optimal solution, we can interpret the steps of the algorithm combinatorially. Recall that we assumed all arc costs c_{ij} are greater than zero.

Combinatorial algorithm for branching problem

Step 1. Set $Z = \{\phi\}$.

Step 2. For each node, other than the root, decrease the cost of all arcs into the node by the cost of the smallest cost arc directed into that node. If the cost of any arc becomes zero, add it to Z.

Step 3. Identify all strongly connected components in the network defined by zero-cost arcs. Contract each strongly connected component in the digraph. If the contraction creates parallel arcs between nodes, delete all but the smallest cost parallel arc.

Step 4. If some node, other than the root, in the digraph has no incoming tight arc go to Step 2.

Step 5. Expand the digraph. The network defined by the arcs in Z contains a minimum cost branching. Construct a minimum cost branching by applying the LIFO dropping rule to Z.

Even if the digraph contains some zero-cost arcs, this combinatorial algorithm (and the dual-ascent algorithm) still solves the minimum cost branching problem. We will see shortly how to apply this combinatorial algorithm to a problem that has some negative-cost arcs.

Example

Figure 3-3 illustrates the algorithm. Initially, $Z = \{\phi\}$. In Step 2 for each node other than root node 0, subtract the cost of the minimum arc into the node from every arc directed into it. For example, the cost of the minimum arc into node 4 is 1. Therefore, we subtract 1 from the cost of arcs $(7,4)$, $(5,4)$ and $(2,4)$. Figure 3-3b shows the zero-cost arcs in bold. We add the arcs $(0,1)$, $(0,2)$, $(5,8)$, $(8,5)$, $(7,4)$, $(6,7)$, $(3,6)$ and $(4,3)$ to the list Z. We now apply Step 3. Nodes 3, 4, 6 and 7 form one strongly connected component and nodes 5 and 8 form another strongly connected component. Contracting these strongly connected components gives the digraph shown in Figure 3-3c. Since none of the nodes in Figure 3-3c have an incoming tight arc, we proceed to Step 2 of the algorithm. In Step 2 we subtract 2 from the cost of arcs into node $\{3,4,6,7\}$ and 4 from the cost of arcs into node $\{5,8\}$, resulting in the digraph shown in Figure 3-3d. Add arcs $(5,4)$ and $(7,8)$ to the list Z. Nodes $\{3,4,6,7\}$ and $\{5,8\}$ form a strongly connected component. By contracting the strongly connected component, we create the digraph shown in Figure 3-3e. Since node $\{3,4,5,6,7,8\}$ has no incoming tight arc, we subtract 1 from the cost of the arcs into node $\{3,4,5,6,7,8\}$, obtaining the digraph shown in Figure 3-3f. Add arc $(2,3)$ to the list Z. The resulting network has no strongly connected component and every node has an incoming tight arc. Therefore, we apply Step 5 of the algorithm. The arcs of Z in LIFO order are $(2,3)$, $(5,4)$, $(7,8)$, $(0,1)$, $(0,2)$, $(5,8)$, $(8,5)$, $(7,4)$, $(6,7)$, $(3,6)$ and $(4,3)$. If we delete arc $(2,3)$, the network defined by the arcs in Z contains no path from the root node to nodes 3, 4, 5, 6,

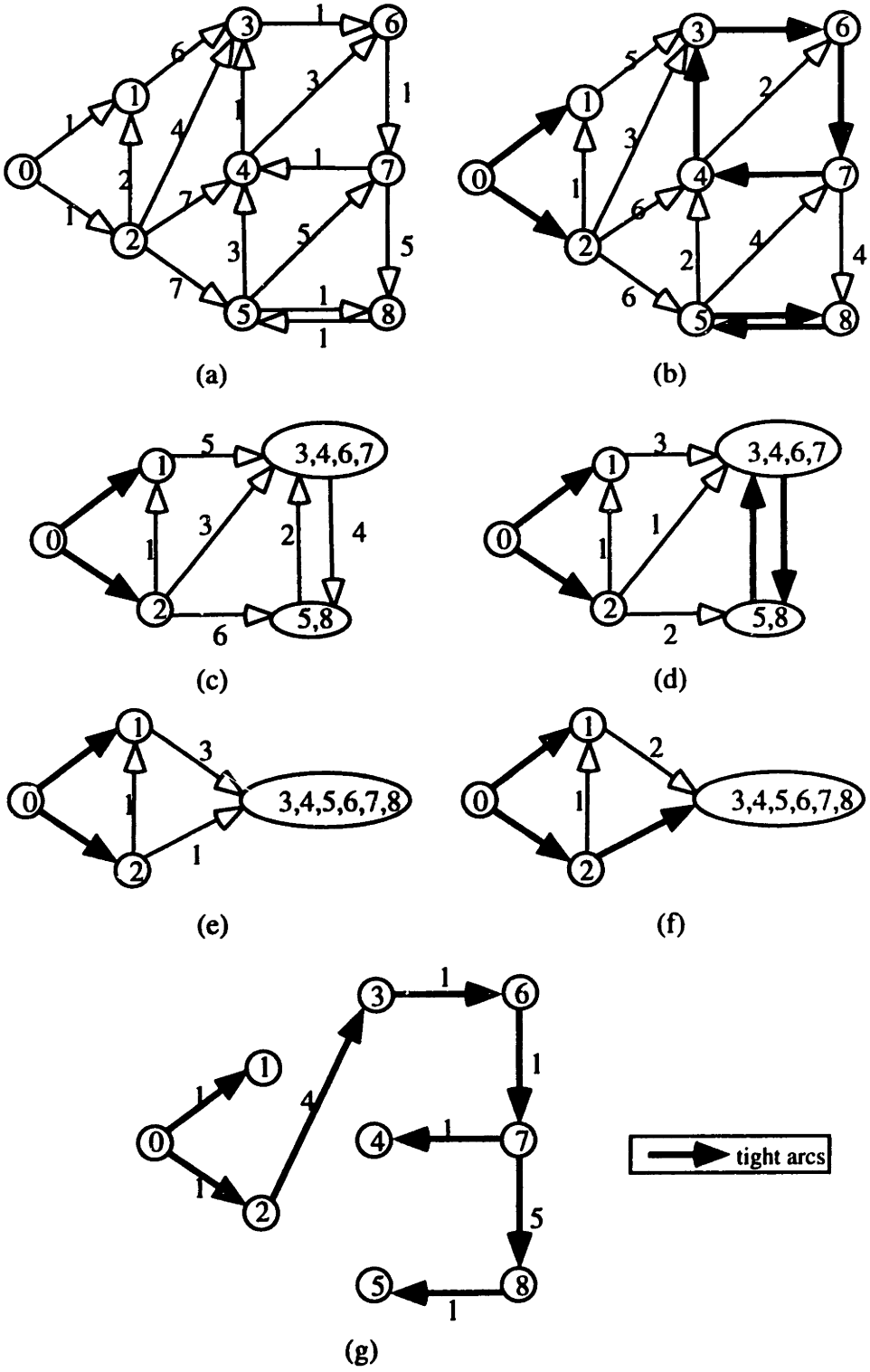


Figure 3-3: Minimum cost branching algorithm.

7 and 8. Therefore, we retain arc $(2,3)$. Proceeding in this fashion, we find the optimal solution shown in Figure 3-3g.

Extension to Negative-Cost Arcs

The earlier definition of a branching was valid for networks with no negative-cost arcs. The precise definition of a branching is as follows. Given a digraph $D = (N, A)$ and a root node r, a branching is a digraph that contains $|N| - 1$ arcs (i.e., the network obtained by replacing the directed arcs by undirected edges is a tree) and has a directed path from the root node to every other node. When costs are nonnegative, it is easy to convert any solution that contains more than $|N| - 1$ arcs and has a path from the root node to every other node to a solution that contains $|N| - 1$ arcs. Therefore, in our prior definition of the branching problem we omitted this condition. (Similarly, in our definition of the minimum spanning tree problem, we omitted the condition that a spanning tree contains $|N| - 1$ edges because we assumed edge costs are nonnegative.)

The combinatorial algorithm will solve the minimum cost branching problem if the digraph has some negative-cost arcs. Suppose the network has negative-cost arcs. We can convert this problem to an equivalent problem with nonnegative arc costs. Consider any node i, other than the root node, that has negative-cost arcs directed into it. Suppose we subtract any constant t_i from all arcs into node i. Since, each node (except the root) in a branching has exactly one incoming arc,[2] any feasible solution to the transformed problem costs t_i less than that of the same solution in the original problem. Thus, the transformed problem and the original problem have the same optimal solutions. By subtracting the cost of the smallest incoming arc into each node from all arcs directed into that node, we obtain a problem with nonnegative arc costs. Now we can apply the combinatorial algorithm.

The combinatorial algorithm we have presented for the minimum cost branching is identical to Edmonds' algorithm [Edm67], as developed and analyzed in another way by Magnanti and Wolsey [MW95]. Our discussion shows how linear programming duality can provide a way to derive combinatorial algorithms for solving certain combinatorial optimization problems.

3.3 Lagrangian Duality Interpretation of Dual-Ascent

In this section we provide a Lagrangian duality interpretation of the dual-ascent algorithm presented in Section 3.2. We consider this interpretation not only for its pedagogical value but, more importantly, because it provides deeper insight into the dual-ascent algorithm.

[2]This result follows from the fact that a branching contains $|N| - 1$ arcs and every node, except the root, in a branching has at least one arc directed into it.

The Lagrangian dual problem to the directed flow formulation can be obtained by a standard procedure in duality theory. Relax all the constraints, except nonnegativity, and consider the Lagrangian dual of the directed flow formulation for the MST problem. For each commodity k, view v_i^k as the Lagrange multiplier associated with the flow conservation constraint for node i, and $w_{ij}^k \geq 0$ as the Lagrange multiplier associated with the constraint $f_{ij}^k \leq y_{ij}$. Let d_i^k denote the demand of commodity k at node i. For the model (3.2), we are considering, for each commodity k, $d_k^k = 1$, $d_0^k = -1$, and $d_j^k = 0$ for $j \neq k$.

The corresponding Lagrangian function is

$$L(\mathbf{y},\mathbf{f},\mathbf{v},\mathbf{w}) = \sum_{(i,j)\in A} c_{ij}y_{ij} + \sum_{i\in N}\sum_{k\in K}(d_i^k - \sum_{j\in N} f_{ji}^k + \sum_{l\in N} f_{il}^k)v_i^k + \sum_{(i,j)\in A}\sum_{k\in K}(f_{ij}^k - y_{ij})w_{ij}^k.$$

Rearranging the terms we obtain

$$L(\mathbf{y},\mathbf{f},\mathbf{v},\mathbf{w}) = \sum_{k\in K}\sum_{i\in N} d_i^k v_i^k + \sum_{(i,j)\in A}(c_{ij} - \sum_{k\in K} w_{ij}^k)y_{ij} + \sum_{(i,j)\in A}\sum_{k\in K}(v_i^k - v_j^k + w_{ij}^k)f_{ij}^k.$$

The value of the Lagrangian dual function $Z(\mathbf{v},\mathbf{w})$ for any vector $(\mathbf{v},\mathbf{w})$ is

$$Z(\mathbf{v},\mathbf{w}) = \min_{(\mathbf{y},\mathbf{f})}\left\{ L(\mathbf{y},\mathbf{f},\mathbf{v},\mathbf{w}) \middle| \begin{array}{l} y_{ij} \geq 0 \text{ and integer for all } (i,j)\in A, \\ f_{ij}^k \geq 0 \text{ for all } (i,j)\in A \text{ and } k\in K. \end{array} \right\}.$$

Because the Lagrangian function $L(\mathbf{y},\mathbf{f},\mathbf{v},\mathbf{w})$ is separable in the arc flows and arc variables, its minimization decomposes by arc for the arc variables $\mathbf{y}$, and by arc and commodity for the flow variables $\mathbf{f}$. If $c_{ij} - \sum_{k\in K} w_{ij}^k \geq 0$, then $y_{ij} = 0$, and if $c_{ij} - \sum_{k\in K} w_{ij}^k < 0$, we can make $Z(\mathbf{v},\mathbf{w})$ as small as desired by setting y_{ij} to an arbitrarily large integer value. Similarly, if $v_i^k - v_j^k + w_{ij}^k \geq 0$, then $f_{ij}^k = 0$, and if $v_i^k - v_j^k + w_{ij}^k < 0$, then $Z(\mathbf{v},\mathbf{w})$ is unbounded from below. Therefore,

$$Z(\mathbf{v},\mathbf{w}) = \begin{cases} \sum_{k\in K}\sum_{i\in N} d_i^k v_i^k & \text{if } c_{ij} - \sum_{k\in K} w_{ij}^k \geq 0, \text{ and} \\ & v_i^k - v_j^k + w_{ij}^k \geq 0 \text{ for all } (i,j)\in A; \\ -\infty & \text{otherwise.} \end{cases}$$

The Lagrangian dual problem is to find a set of Lagrange multipliers that maximize the dual function. That is,

$$\begin{array}{rl} \text{Maximize} & Z(\mathbf{v},\mathbf{w}) \\ \text{subject to:} & \mathbf{w} \geq \mathbf{0} \\ & \mathbf{v} \quad \text{unrestricted in sign.} \end{array}$$

Suppose we are given a node potential vector $\bar{\mathbf{v}}$. Then, if (i) $\bar{v}_i^k - \bar{v}_j^k + w_{ij}^k \geq 0$, and (ii) $c_{ij} - \sum_{k \in K} w_{ij}^k \geq 0$, for all $(i,j) \in A$, the dual function $Z(\bar{\mathbf{v}}, \mathbf{w})$ is equal to $\sum_{k \in K} \sum_{i \in N} d_i^k \bar{v}_i^k$. The first condition together with $w_{ij}^k \geq 0$ can be stated as $w_{ij}^k \geq \max(0, \bar{v}_j^k - \bar{v}_i^k)$. Now observe that satisfying this condition at equality does not change the value of the Lagrangian function. Therefore, we can restrict our attention to solutions $(\mathbf{v}, \mathbf{w})$ of the form $w_{ij}^k = \max(0, v_j^k - v_i^k)$ for all $(i,j) \in A$ and $k \in K$. Consequently, we can write the Lagrangian dual problem as a maximization problem over the node potentials $\mathbf{v}$. The Lagrangian dual problem is

$$\begin{aligned} &\text{Maximize } Z(\mathbf{v}) \\ &\text{subject to no constraint on } \mathbf{v} \end{aligned}$$

with

$$Z(\mathbf{v}) = \begin{cases} \sum_{k \in K} \sum_{i \in N} d_i^k v_i^k & \text{if } c_{ij} - \sum_{k \in K} \max(0, v_j^k - v_i^k) \geq 0 \\ & \text{for all } (i,j) \in A; \\ -\infty & \text{otherwise.} \end{cases}$$

Since the sum of demands over all the nodes for any commodity is zero, increasing all the node potentials by the same amount does not change the value of the dual function. Therefore, for each commodity we can fix the node potential for one node at any desired value. We will set $v_0^k = 0$ for all commodities. We can transform any dual solution with v_0^k nonzero to one with v_0^k zero by subtracting v_0^k from v_j^k for each node j in the digraph.

Let us make a few observations concerning the Lagrangian dual problem. Consider any solution $(\mathbf{v}, \mathbf{w})$ to the Lagrangian dual problem with $w_{ij}^k = \max(0, v_j^k - v_i^k)$ for all $(i,j) \in A$. If the value of the Lagrangian dual function is bounded for this $(\mathbf{v}, \mathbf{w})$ pair, then $(\mathbf{v}, \mathbf{w})$ is a feasible solution to Formulation (3.5) with the same objective value as $Z(\mathbf{v})$. Note that any feasible solution $(\mathbf{v}, \mathbf{w})$ to Formulation (3.5) defines a feasible solution $\mathbf{v}$ to the Lagrangian dual problem with the same objective value. Therefore, *the LP dual and the Lagrangian dual problems are equivalent if the Lagrangian dual problem is bounded from below.* (If the Lagrangian-dual problem is unbounded from below, then for all node potentials, $c_{ij} - \sum_{k \in K} \max(0, v_j^k - v_i^k) < 0$ for some arc $(i,j) \in A$, and so the LP dual is infeasible.) Because of this equivalence, from now on we refer to Lagrangian dual function and the Lagrangian dual problem as the dual function and dual problem respectively.

Let us now interpret the dual-ascent algorithm of the previous section. Starting with a feasible dual solution $(\mathbf{v}, \mathbf{w}) = (\mathbf{0}, \mathbf{0})$, the dual-ascent algorithm sequentially updates the values of $\mathbf{v}$ and $\mathbf{w}$ at each iteration. The dual-ascent algorithm selects a loose dicut $\delta^-(T)$ and a commodity k and performs a basic dual-ascent step. It increases w_{ij}^k for all arcs in the loose dicut $\delta^-(T)$ and the node potentials for commodity k for all nodes in T by an equal amount, leaving all the other variables w_{ij}^k and node potential values unchanged.

We now show that starting with the solution $(\mathbf{v}, \mathbf{w}) = (\mathbf{0}, \mathbf{0})$, the dual-ascent algorithm for the minimum cost branching problem, described in Section 3.2, maintains the property

$$w_{ij}^k = \max(0, v_j^k - v_i^k).$$

First, observe that the algorithm couples any increase in w_{ij}^k with an equal increase in v_j^k, keeping v_i^k fixed. Moreover, once a node $j \in N(k)$, it is always in $N(k)$. The strongly connected components become larger in each iteration, implying that if we increase w_{ij}^k, we never change the value of w_{ji}^k (it stays at value zero) and so throughout the algorithm $v_j^k \geq v_i^k$, and $w_{ij}^k = v_j^k - v_i^k = \max(0, v_i^k - v_j^k)$ and $w_{ji}^k = 0 = \max(0, v_i^k - v_j^k)$. We can therefore interpret the dual-ascent algorithm solely in terms of node potential changes.

Mathematically, each iteration of the dual-ascent algorithm involves a node potential change along a direction of the form $\gamma_T^k = (0, \ldots, 0, \gamma_0^k, \gamma_i^k, \ldots, \gamma_{|N|-1}^k, 0, \ldots, 0)$, with

$$\gamma_j^l = \begin{cases} 1 & \text{if } j \in T \text{ and } l = k; \\ 0 & \text{otherwise.} \end{cases}$$

In this equation, T is a subset of nodes and k is a commodity. We refer to such a direction as a *basic direction.*

The direction γ_T^k improves the dual function $Z(\mathbf{v})$ if γ_T^k is a direction of ascent. That is, if the directional derivative of the the dual function $Z(\mathbf{v})$ in the direction γ_T^k is positive. The directional derivative is given by,

$$\begin{aligned} Z'(\mathbf{v}; \gamma_T^k) &= \lim_{\alpha \downarrow 0} \frac{Z(\mathbf{v} + \alpha\gamma_T^k) - Z(\mathbf{v})}{\alpha} \\ &= \begin{cases} \sum_{j \in T} d_j^k & \text{if for all } (i,j) \text{ with } j \in T, i \notin T, \\ & c_{ij} - \sum_{k \in K} \max(0, v_j^k - v_i^k) > 0; \\ \text{undefined} & \text{otherwise.} \end{cases} \end{aligned} \tag{3.6}$$

The quantity $c_{ij} - \sum_{k \in K} \max(0, v_j^k - v_i^k)$ for the dual-ascent algorithm is $c_{ij} - \sum_{k \in K} w_{ij}^k$, which equals the slack s_{ij} for arc (i,j). Also, observe that in the dual-ascent algorithm all the arcs in a loose dicut, say $\delta^-(N(k))$, have a positive slack. Furthermore, only node k, the destination of commodity k, in the set $N(k)$ has positive demand, of 1 unit, for commodity k. Therefore, equation (3.6) shows that the loose dicut $\delta^-(N(k))$ corresponds to an ascent direction $\gamma_{N(k)}^k$. The value of the directional derivative for a loose dicut is 1. Thus, every unit increase in the node potentials for all the nodes in the set $N(k)$ increases the objective function by 1. This result is consistent with the shortest path interpretation, that is, every unit increase in w_{ij}^k for all arcs in the loose dicut increases the shortest path length, and thus the dual objective function, by 1.

This discussion shows that we can interpret the dual-ascent algorithm as a search for basic directions that successively improve the dual function. By a judicious choice of these basic directions, for the special case of the minimum cost branching problem, we were able to obtain the optimal solution to the dual problem. The Lagrangian duality interpretation provides a slightly different framework for considering the dual-ascent algorithm. In Chapter 4 we will develop a dual-ascent algorithm for the network design problem with connectivity requirements based on this interpretation.

Our development of the Lagrangian duality interpretation of dual-ascent closely follows the approach in the book by Bertsekas [Ber91], which describes the dual-ascent approach for single commodity minimum cost network flow problems. Our description shows that the same approach is effective for solving other combinatorial optimization problems.

3.4 Dual-Ascent for the Steiner Branching Problem

In this section we generalize the ideas presented in the dual-ascent algorithm for the branching problem. We characterize the loose dicuts used in the dual-ascent algorithm for the branching problem, and apply this characterization to derive Wong's dual-ascent algorithm for the Steiner branching problem. Our derivation of Wong's algorithm is significantly different than the one originally presented in [Won84]. Finally, we discuss several different implementations of Wong's algorithm.

3.4.1 Formulation for the Steiner Branching Problem

The Steiner branching problem is a generalization of the branching problem. Given a directed graph $D = (N, A)$, a root node r, and a set T ($r \notin T$) of terminal nodes, we wish to design a minimum cost network that contains a directed path from the root node r to every node in T.

We can formulate this problem as a variation of the directed flow formulation (3.4) for the branching problem. In this directed flow formulation, let y_{ij} be 1 if arc (i, j) is in the design and be 0 otherwise. For each node $i \in T$, create a commodity k with the origin as the root node and node i as the destination node (i.e., $D(k) = i$), and require that the network design have the capability to flow a unit of flow of each commodity from the root node to this node. (Observe that the number of commodities in K is equal to the number of terminal nodes, i.e., the number of nodes in T.) The following mixed integer programming problem models these requirements.

Directed flow formulation for the Steiner problem:

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}y_{ij} \tag{3.7a}$$

$$\text{subject to:} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k = \begin{cases} -1 & \text{if } i = 0; \\ 1 & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } i \in N \text{ and } k \in K \tag{3.7b}$$

$$f_{ij}^k \le y_{ij} \quad \text{for all } (i,j) \in A \text{ and } k \in K \tag{3.7c}$$

$$f_{ij}^k \ge 0 \quad \text{for all } (i,j) \in A \text{ and } k \in K \tag{3.7d}$$

$$y_{ij} \ge 0 \quad \text{and integer for all } (i,j) \in A. \tag{3.7e}$$

We will apply the dual-ascent algorithm to the following dual of the linear programming relaxation of this formulation.

Dual of directed flow formulation for the Steiner branching problem:

$$\text{Maximize} \quad \sum_{k\in K} v_{D(k)}^k \tag{3.8a}$$

$$\text{subject to} \quad v_j^k - v_i^k \le w_{ij}^k \quad \text{for all } (i,j) \in A \text{ and } k \in K \tag{3.8b}$$

$$\sum_{k\in K} w_{ij}^k \le c_{ij} \quad \text{for all } (i,j) \in A \tag{3.8c}$$

$$w_{ij}^k \ge 0 \quad \text{for all } (i,j) \in A \text{ and } k \in K. \tag{3.8d}$$

3.4.2 Dual-Ascent Algorithm

Notice again that if we are given values for the **w** variables, the dual problem separates by commodity. The subproblem corresponding to commodity k is the dual to the directed shortest path problem between the root node and node $D(k)$, with arc lengths w_{ij}^k. This observation suggests a simple dual-ascent strategy. For any one of the commodities $k \in K$, identify and perform a basic dual-ascent step on a loose $O(k)$–$D(k)$ dicut. Continue in this fashion until for every commodity k, every $O(k)$–$D(k)$ dicut contains a tight arc. Each step of this strategy improves the dual objective value and at termination the auxiliary network contains a Steiner branching (because it contains a path from the root node to every node in T).

For the branching problem we identified a particular type of loose dicut and showed that by performing basic dual-ascent steps on these loose dicuts, we solve the branching problem optimally. The Steiner branching problem is $\mathcal{NP}$-hard. Therefore, we will not be able to describe a dual-ascent algorithm that solves the problem optimally. Instead we use the dual-ascent algorithm to develop a heuristic solution to the problem and a lower

bound on the cost of an optimal Steiner branching (and so, a guarantee on the degree of suboptimality of the heuristic solution).

Let Z denote the set of tight arcs in the digraph. If $\delta^-(S)$ is a loose $O(k)$-$D(k)$ dicut for some commodity k, we refer to S as an *ascent set* (that is, a set on which it is possible to perform a basic dual-ascent step).

In the dual-ascent algorithm for the branching problem, we performed basic dual-ascent steps on loose dicuts in the auxiliary network associated with strongly connected components with no incoming arc. Let S denote a strongly connected component with no incoming arc (an ascent set) and $\delta^-(S)$ its loose dicut. Observe that the auxiliary network contains at least one arc directed into every subset of S (i.e., for all $R \subset S$, $\delta^-(R) \cap Z \neq \phi$). Therefore, no subset of S defines a loose dicut. This observation suggests the following definition. A set of nodes S is a *minimal ascent set*, if S is an ascent set and no subset of S is an ascent set.

We will design a dual-ascent algorithm for the Steiner branching problem that uses minimal ascent sets. Start by setting $(\mathbf{v}, \mathbf{w}) = (\mathbf{0}, \mathbf{0})$ and set Z to be the set of arcs with zero slack. Consider the auxiliary network (i.e., D_Z) and identify a minimal ascent set S. Perform a basic dual-ascent step on the loose dicut $\delta^-(S)$ and update the auxiliary network. Continue in this fashion until the digraph contains no ascent sets. At the conclusion of the ascent steps, the auxiliary network contains a feasible Steiner branching and the dual solution provides a lower bound on the optimal Steiner branching. We can obtain a feasible Steiner branching by applying the LIFO drop rule to the auxiliary network (we refer to this method as the LIFO drop heuristic). The dual-ascent algorithm has the following formal description.

Dual-ascent algorithm for the Steiner branching problem (Wong's Algorithm)

1. Initialization. Set

   (a) $w_{ij}^k = 0$ for all $(i,j) \in A$ and $k \in K$,

   (b) $s_{ij} = c_{ij}$ for all $(i,j) \in A$,

   (c) $Z = \{(i,j) : (i,j) \in A; s_{ij} = 0\}$ (the set of zero-slack arcs),

   (d) $v_i^k = 0$ for all $i \in N$ and $k \in K$,

   (e) $L = \sum_{k \in K} v_{D(k)}^k$ (the lower bound).

2. While the digraph contains[3] a minimal ascent set S,

[3]We show how to identify minimal ascent set and consider other implementation issues in Section 3.4.3.

(a) Select a node $l \in T \cap S$ and identify the commodity k associated with it (i.e., $D(k) = l$). Set $N(k)$ to be the set of nodes in D_Z that have a directed path to node l.

(b) Find minimum slack.

- $\epsilon = \min_{\{(i,j):(i,j)\in\delta^-(N(k))\}} s_{ij}$.

(c) Update relevant dual variables, slacks, the auxiliary network D_Z, and the lower bound L.

- $w_{ij}^k = w_{ij}^k + \epsilon \qquad \forall\ (i,j) \in \delta^-(N(k))$,
- $s_{ij} = s_{ij} - \epsilon \qquad \forall\ (i,j) \in \delta^-(N(k))$,
- $v_i^k = v_i^k + \epsilon \qquad \forall\ i \in N(k)$,
- $L = L + \epsilon$,
- $Z = Z \cup \{(i,j) : (i,j) \in \delta^-(N(k)) \text{ and } s_{ij} = 0\}$.

3. The network defined by the arcs in Z contains a Steiner branching. Construct a Steiner branching by applying the LIFO dropping rule to Z.

Our development and presentation of Wong's algorithm is quite different from Wong's original development [Won84]. His dual-ascent procedure uses a *root component rule* to identify ascent sets on which to perform basic dual-ascent steps.[4] We show that the ascent sets that the root component rule identifies are minimal ascent sets.

To describe the root component rule, we introduce some notation. If the auxiliary network contains a directed path from node i to node j and no directed path from node j to node i, then we say that node i *dangles* from node j. A strongly connected component R is a *root component* if it contains a member of T and no member of $T \cup \{0\}$ dangles from a member of R. Let $S_R = \{j$: there is a directed path in D_Z from node j to a node in $R\}$. Wong's algorithm performs a basic dual-ascent step on the dicut $\delta^-(S_R)$ associated with a root component R.

Suppose R is a root component. Observe that every node in $T \cap S_R$ is also in R; otherwise, this node dangles from a member of R, and thus R is not a root component. Moreover, no subset of S_R is an ascent set since every subset of S_R that contains a node in T has an incoming tight arc (if the subset of S_R does not contain a node in T, then it cannot be an ascent set because it does not define an $O(k)$–$D(k)$ dicut for any commodity). Therefore, S_R is a minimal ascent set.

Suppose R is not a root component. Then some node $i \in T$ dangles from a member of R. Let $S_i = \{j$: there is a directed path in D_Z from node j to node $i\}$. Because node i

[4] For convenience, we use the phrase "perform a basic dual-ascent step on an ascent set S" to mean "perform a basic dual-ascent step on the loose dicut $\delta^-(S)$."

dangles from a member of R, every node in S_i is also in S_R and every member of R is not in S_i (i.e., $S_i \subset S_R$). As a result, S_R is not a minimal ascent set.[5]

We note that Wong's dual-ascent algorithm satisfies the property that for each commodity and arc in the network

$$w_{ij}^k = \max(0, v_j^k - v_i^k).$$

Observe that the algorithm couples any increase in w_{ij}^k with an equal increase in v_j^k, keeping v_i^k fixed. Moreover, once a node $j \in N(k)$, it is always in $N(k)$ since the set $N(k)$ is defined as the set of nodes in D_Z that have a directed path to $D(k)$ in the auxiliary network and the algorithm never drops any arc from D_Z (until Step 3). We can, therefore, interpret the dual-ascent algorithm solely in terms of node potential changes. Consequently, we need not maintain values of the w variables. Furthermore, if our interest is only in the value of the dual objective, that is the lower bound, we need not store the node potentials. The slacks on the arcs provide the necessary information to update the lower bound, auxiliary networks and slack (see Step 2 of Wong's algorithm).

Wong's dual ascent algorithm has proven to be very effective for the undirected Steiner tree problem. To apply the algorithm to this problem, we first convert the undirected Steiner tree problem to a (directed) Steiner branching problem in exactly the same way we converted the minimum spanning tree problem to a branching problem.[6] In his computational experiments, Wong solved Steiner tree problems with up to 60 nodes and 120 edges, with an average gap of less than 1 percent between the upper and lower bounds of the dual-ascent algorithm. In Section 3.4.3 we describe several implementations of Wong's algorithm and some preliminary computational experience with a particular implementation of the algorithm. In these experiments, we solved problems with as many as 300 nodes and 6000 edges (12000 arcs) to within 1 percent of optimality. These experiments confirm the effectiveness of Wong's algorithm, with the LIFO drop heuristic, as a heuristic method for Steiner tree problems.

[5]Observe that all minimal ascent sets are defined by the ascent sets associated with root components. To see this, consider any minimal ascent set Q. By definition, a minimal ascent set has no incoming tight arcs and contains at least one node, say i, from T. Let R denote the strongly connected component that node i belongs to and define $S_R = \{j$: the network contains a directed path in D_Z from node j to a node in $R\}$. We claim $S_R = Q$. This result is true because S_R is an ascent set and is a subset of Q (since the network has no arcs directed into Q, S_R only contains nodes from Q). But Q is a minimal ascent set, and therefore $S_R = Q$.

[6]An interesting question is whether the choice of the root node in this conversion affects the optimal objective value of the LP relaxation of the directed flow formulation (3.7). Goemans and Myung [GM93] proved that in converting an undirected Steiner tree problem to a Steiner branching problem, the choice of the root node does not change the optimal objective value of the LP relaxation of this formulation. Therefore, we can arbitrarily choose any one of the nodes as the root node without affecting the quality of the solution.

Example

To illustrate Wong's dual-ascent algorithm, we consider the example shown in Figure 3-4. Figure 3-4a shows the arc costs, node 0 is the root node, and $T = \{3, 5, 7, 8\}$. Initially, $Z = \{\phi\}$, $(\mathbf{v}, \mathbf{w}) = (\mathbf{0}, \mathbf{0})$, and $L = 0$.

The sets $\{3\}, \{5\}, \{7\}$ and $\{8\}$ are minimal ascent sets. The algorithm first performs a basic dual-ascent step on $\{3\}$. As a result, arc $(1, 3)$ becomes tight and is added to Z. The dual objective increases by 1 unit to $L = 1$. The algorithm then performs a basic dual-ascent step on the minimal ascent set $\{5\}$.[7] As a result, arc $(8, 5)$ becomes tight and is added to Z. The dual objective increases by 1 unit to $L = 2$. The algorithm then performs a basic dual-ascent step on the minimal ascent set $\{7\}$. Arcs $(3, 7)$ and $(5, 7)$ become tight and are added to Z. The dual objective increases by 5 units to $L = 7$. The algorithm then performs a basic dual-ascent step on the minimal ascent set $\{8\}$. As a result, arc $(5, 8)$ becomes tight and is added to Z, and the dual objective increases by 1 unit to $L = 8$. Figure 3-4b shows the digraph at the conclusion of the first four iterations.

At this point, sets $\{3, 1\}$ and $\{5, 8\}$ are minimal ascent sets. The dual-ascent algorithm performs a basic dual-ascent step on the minimal ascent set $\{3, 1\}$. As a result, arc $(4, 3)$ becomes tight and is added to Z. The dual objective increases by 3 units to $L = 11$. The algorithm then performs a basic dual-ascent step on the minimal ascent set $\{5, 8\}$. At the conclusion of this step, arc $(2, 5)$ becomes tight and is added it to Z. The dual objective increases by 2 units to $L = 13$. Figure 3-4c shows the digraph at the conclusion of these two iterations.

At this stage, $\{3, 1, 4\}$ and $\{5, 8, 2\}$ are minimal ascent sets. The algorithm performs a basic dual-ascent step on the ascent set $\{3, 1, 4\}$. As a result, arc $(0, 1)$ becomes tight and the algorithm adds it to Z. The dual objective increases by 6 units to $L = 19$. The algorithm then performs a dual-ascent step on the minimal ascent set $\{5, 8, 2\}$. The arc $(0, 2)$ become tight and the algorithm adds it to Z. The dual objective increases by 6 units to $L = 25$. Figure 3-4d shows the resulting digraph.

At this point, the network has no ascent set. Therefore, we apply the LIFO drop heuristic. The arcs of Z in LIFO order are $(1, 3)$, $(8, 5)$, $(3, 7)$, $(5, 7)$, $(5, 8)$, $(4, 3)$, $(2, 5)$, $(0, 1)$, and $(0, 2)$. If we delete arc $(0, 2)$ the network defined by the arcs in Z contains no path from the root node to terminal nodes 7 and 8. Therefore, we retain arc $(0, 2)$. Proceeding in this fashion, we find the solution shown in Figure 3-4e. The cost of this feasible solution is 25 units. Since the lower bound is 25 units, we have obtained an optimal solution to the

[7] In Section 3.4.3 we prove the following property. Suppose S is a minimal ascent set. Then after a sequence of basic dual-ascent steps, S is either a minimal ascent set or no longer an ascent set (i.e., it has a tight arc directed into it). Consequently, $\{5\}$ is a minimal ascent set because it has no tight arc directed into it (it is still an ascent set).

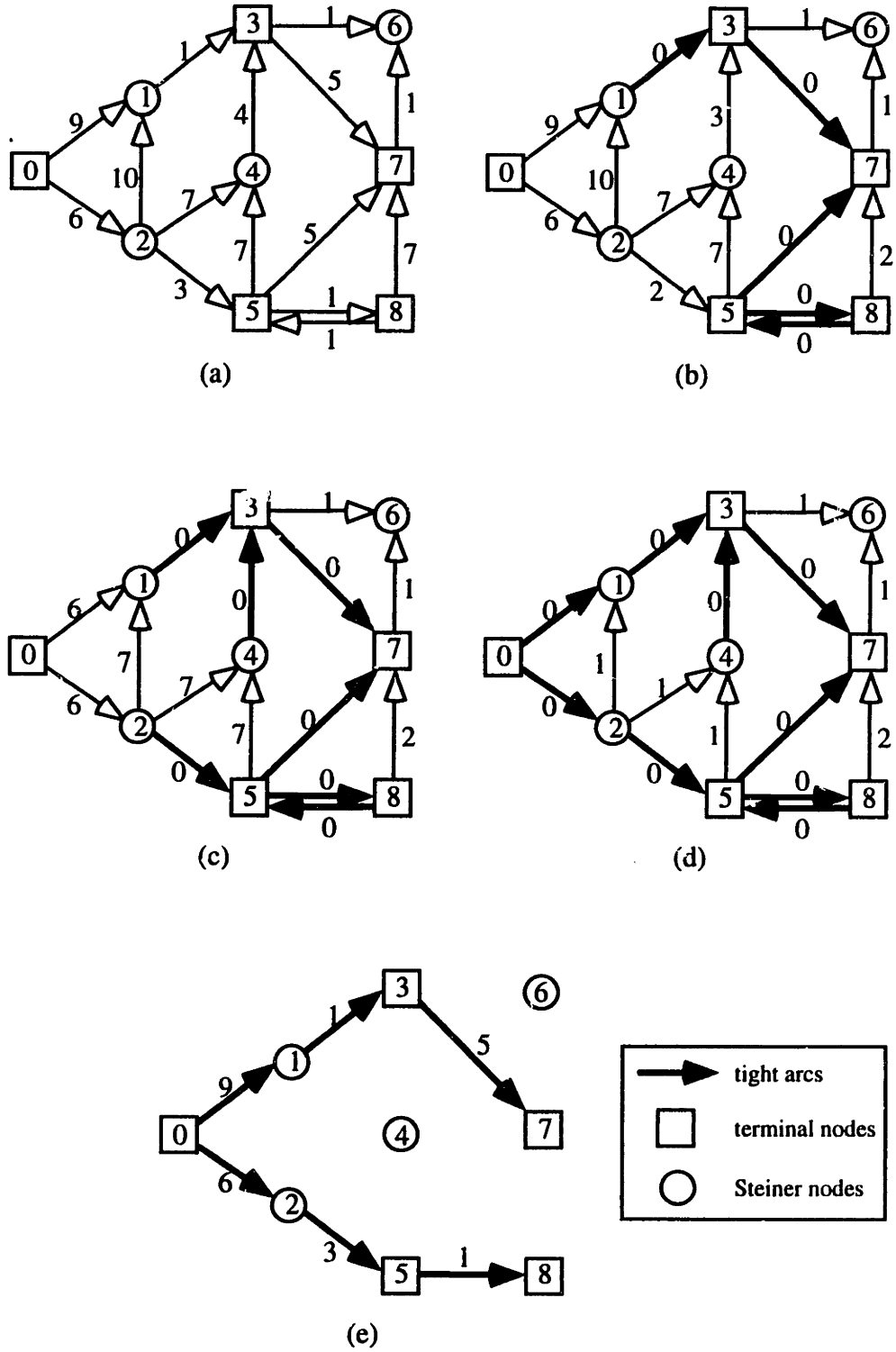


Figure 3-4: Wong's algorithm.

Steiner branching problem.

This example illustrates the effectiveness of performing dual-ascent steps only on minimal ascent sets. Suppose the algorithm performed a basic dual-ascent step on the set $S = \{1, 2, 3, 4, 5, 7, 8\}$ in Figure 3-4c. Note that S is not a minimal ascent set because it contains two ascent sets—$\{3, 1, 4\}$ and $\{5, 8, 2\}$. At the conclusion of a basic dual-ascent step on S, both arcs $(0, 1)$ and $(0, 2)$ become tight and the dual objective increases by 6 units to $L = 19$. No more ascent is possible since the resulting network has no ascent set. The primal solution obtained by performing the LIFO drop heuristic has cost 25, but the dual lower bound is only 19.

3.4.3 Identifying Minimal Ascent Sets: Alternative Implementations of the Dual-Ascent Algorithm

In this section we first show how to identify minimal ascent sets (actually root components). We then describe four different implementations of the dual-ascent algorithm. The first three implementations rely on explicit identification of minimal ascent sets. The fourth implementation (by Balakrishnan, Magnanti and Wong [BMW89]) does not explicitly identify minimal ascent sets but nevertheless simulates the minimal ascent set rule. Finally, we report on some (limited) computational experiments with the last implementation.

In this section we assume that the reader is familiar with elementary graph algorithms and data structures. For completeness, in Figure 3-5 we describe the following three algorithms from Cormen, Leiserson and Rivest [CLR90]:

- Depth first search,
- Identifying strongly connected components in a digraph, and
- Topological sort of a digraph.[8]

All three algorithms run in $\mathcal{O}(|A|)$ time.

The *DFS* algorithm produces the following output. The **pred** array defines the depth first search forest $D_{\mathbf{pred}} = (N, A_{\mathbf{pred}})$, with

$$A_{\mathbf{pred}} = \{(\mathbf{pred}[i], i) : i \in N \text{ and } \mathbf{pred}[i] \neq -1\}.$$

Each component of the forest is a rooted tree. The forest contains a directed path from the root of each tree to all the nodes in that tree. The **color** array keeps track of nodes scanned.

[8] A digraph can be topologically sorted if its nodes can be ordered in such a way that the tail of an arc appears on the sorted list before the head of that arc. A well-known result in graph theory is that a digraph can be topologically sorted if and only if it is acyclic.

DFS(D)
1. for $i \in N$
2. **pred**[i] = -1;
3. **color**[i] = green;
4. time = 0;
5. for $i \in N$
6. if (**color**[i] = green)
7. then *DFS-Visit*(i);

DFS-Visit(i)
1. **color**[i] = yellow.
2. time = time + 1;
3. **start**[i] = time;
4. while i's adjacency list is nonempty
5. select a node j from i's adjacency list;
6. if (**color**[j] = green)
7. then **pred**[j] = i;
8. *DFS-Visit*(j);
9. **color**[i] = red;
10. time = time + 1;
11. **finish**[i] = time;

Strongly-Connected-Components(D)
1. call *DFS*(D) to compute finishing times **finish**[i] for each node i.
2. compute D^T, the transpose[a] of digraph D.
3. call *DFS*(D^T), but in the main loop of DFS (line 5 of DFS), consider the nodes in order of decreasing **finish**[i] (as computed in line 1).
4. output the nodes of each tree in the depth first forest of step 3 as a separate strongly connected component.

Topological-Sort(D)
1. call *DFS*(D) to compute finishing times **finish**[i] for each node i.
2. as each node is finished, insert it onto a linked list.
3. return the linked list of vertices.

[a] D^T, the transpose of D, is the digraph $D^T = (N, A^T)$ defined by the arc set $A^T = \{(i,j) : (j,i) \in A\}$.

Figure 3-5: Graph algorithms: Depth first search, Strongly connected components, Topological sort.

(**Color**[i] is green if node i is unscanned, **cclor**[i] is yellow if the algorithm is in the process of scanning from node i, and **color**[i] is red if node i has been scanned.) The **start** array indicates the order in which the *DFS* algorithm first examined the nodes (i.e., indicates order in which the color of a node becomes yellow). The **finish** array indicates the order in which the algorithm finished scanning the adjacency list of the nodes (i.e., indicates order in which the color of a node becomes red). The **start** and **finish** array satisfy some useful properties which prove to be useful in reasoning the correctness of *Strongly-Connected-Components* and *Topological-Sort.*

A Linear Time Algorithm to Identify Root Components

We now provide a linear time algorithm to identify all root components in the auxiliary network D_Z. We will use this algorithm in three implementations of the dual-ascent algorithm. We note that since the nodes in a root component are strongly connected, it is sufficient to identify root components on the digraph obtained by contracting all strongly connected components. If a strongly connected component in D_Z contains a terminal node, then it is a terminal node in the contracted network.

Observe that a digraph with no strongly connected components is acyclic, and therefore can be topologically sorted. Consequently, we show how to identify root components on a topologically sorted digraph (i.e., for any arc (i, j) in this digraph $i < j$) in which the root node is first in the topological order.[9]

The algorithm for identifying root components is an adaptation of the depth first search algorithm and its correctness relies on the fact that the digraph is topologically sorted. In this algorithm, **req**[i] = 1 if node i is a terminal node or the root node; otherwise, **req**[i] = 0. The output of this algorithm is the array **root**. Except for the root node, **root**[i] = 1 if node i is a root component, and is 0 otherwise (**root**[i] is always 1 for the root node).

Theorem 3.4.1 *The algorithm* Root-Components *described in Figure 3-6 correctly identifies root components in linear time.*

Proof: Lines 1 through 4 of the algorithm are the initialization. Line 5 performs depth first search from the root node and identifies all nodes in D_Z that the root node *reaches.*[10] Any node that the root node reaches cannot be a root component. We now argue that any node that the algorithm considers in line 7 (besides the root node) is a root component. This is true because the algorithm initiates depth first search only from required nodes

[9]Recall the root node has no incoming arc, so in some topological ordering, the root node has the lowest index. To obtain such an ordering, topologically sort the network obtained by deleting the root node and append the root to the front of the topological order.

[10]If the network contains a directed path from node i to node j, we say node i *reaches* node j.

```
Root-Components
1.    for i = 0 to |N| − 1
2.          pred[i] = −1;
3.          color[i] = green;
4.          root[i] = 0;
5.    for i = 0 to |N| − 1
6.          if ((color[i] = green) and (req[i]=1))
7.                then root[i] = 1;
8.                     DFS-Visit(i);
```

Figure 3-6: Algorithm to find root components.

(i.e., $\mathbf{req}[i] = 1$). As a result, any node the algorithm considers in line 7 is not reachable from any lower numbered required node. Furthermore, because the nodes are in topological order, no such node is reachable from any higher numbered node. Therefore, the node is a root component.

The algorithm for identifying root components is a depth first search algorithm. Consequently, *Root-Components* is an $\mathcal{O}(|A|)$ procedure. ■

We also note that we can reduce the size of the digraph D in the dual-ascent procedure, by contracting every strongly connected component in the auxiliary network D_Z to a single node in the digraph D. If this contraction creates parallel arcs, we retain the arc with the smallest slack. Although contraction does not improve the complexity analysis of any of the implementations we discuss, it might prove to be useful in practice, especially when solving large-scale problems.

We now briefly describe four different algorithms that implement the minimal ascent set rule, but perform their dual-ascent steps in different ways.

Implementation 1

This implementation first identifies a single root component and performs a basic dual-ascent step on the minimal ascent set associated with it. The algorithm continues in this fashion (i.e., again identifies a root component [it could be the same root component or another one] and performs a basic dual-ascent step on the minimal ascent set associated with it) until no more ascent is possible. By performing depth first search from the root

component on the transpose[11] of D_Z, we can identify the minimal ascent set associated with a root component in $\mathcal{O}(|A|)$ (since $Z \subseteq A$) time. Note that we can terminate the algorithm for finding root components after finding the first root component, because this implementation performs ascent on a single root component. Performing a basic dual-ascent step requires identifying the arc with minimum slack that is directed into the minimal ascent set, a computation that requires $\mathcal{O}(|A|)$ time if we sequentially scan all the arcs in the digraph.

We provide two arguments to deduce the complexity of this algorithm. Identifying a minimal ascent set S in the digraph requires $\mathcal{O}(|A|)$ time, and identifying the minimum slack arc in $\delta^-(S)$ requires $\mathcal{O}(|A|)$ time. The dual-ascent algorithm performs at most $|A|$ iterations because, after $|A|$ iterations all the arcs in the original digraph D are tight. This argument shows the dual-ascent algorithm is an $\mathcal{O}(|A|^2)$ procedure. The second argument is as follows. At the conclusion of an iteration of the dual-ascent algorithm, one additional node can reach a node in T (that is, some node not in the minimal ascent set S can reach the terminal nodes in S). Therefore, after at most $|T||N|$ iterations the auxiliary network contains a path from every node in the network to every node in T. This analysis leads to a bound of $\mathcal{O}(|T||N||A|)$ operations. Combining these two arguments shows that Implementation 1 is an $\mathcal{O}(|A| \min(|T||N|, |A|))$ algorithm.

Implementation 2

In this implementation, we find the minimal ascent sets, $S_1, \ldots, S_p$, corresponding to all root components. We then sequentially perform a basic dual-ascent step on each of the sets $S_1, \ldots, S_p$. We continue by again finding all minimal ascent sets and sequentially performing basic dual-ascent steps on each set. (The example in Figure 3-4 is similar to this implementation). The correctness of this implementation follows from the following property that we mentioned earlier.

Proposition 3.4.2 *Suppose S is a minimal ascent set. Then after a sequence basic dual-ascent steps, S is either a minimal ascent set or no longer an ascent set (i.e., it has a tight arc directed into it).*

Proof: By definition, no subset of a minimal ascent set is an ascent set. Therefore, if S has no tight arc directed into it, it is still a minimal ascent set. ■

Consequently, if the dual-ascent algorithm attempts to perform a basic dual-ascent step on S, and S is not an ascent set, then it finds that the minimum slack of an arc in the dicut $\delta^-(S)$ is zero and makes no change to the dual variables and lower bound.

[11] See Figure 3-5 for the definition of transpose.

We can identify the minimal ascent sets corresponding to the root components by performing depth first search from each root component in the transpose of D_L^*.[12] If the network contains p root components, we can obtain all minimal ascent sets in $\mathcal{O}(p|A|)$ steps. The algorithm now performs p basic dual-ascent steps, one for each of the p sets, each requiring $\mathcal{O}(|A|)$ steps. Therefore, p iterations require $\mathcal{O}(|A| + p|A| + p|A|)$ steps. So on the average, each of the p iterations requires $\mathcal{O}(|A|)$ steps. Now, arguing similarly as in Implementation 1, we can prove that this implementation requires $\mathcal{O}(|A| \min(|T||N|, |A|))$ time.

Implementation 3

In this implementation we consider all minimal ascent sets simultaneously, and perform an ascent step on them *simultaneously*. For example, suppose S_1 and S_2 are the only minimal ascent sets. Then, this implementation increases w_{ij}^k and w_{ij}^h, for a commodity k with destination in S_1 and a commodity h with destination in S_2, for all arcs in $\delta^-(S_1)$ and $\delta^-(S_2)$ simultaneously. Suppose we increase w_{ij}^k by ϵ for all arcs in $\delta^-(S_1)$, and w_{ij}^h by ϵ for all arcs in $\delta^-(S_2)$. Observe that the dual objective increases by 2ϵ, the slack of an arc in $\delta^-(S_1) \cap \delta^-(S_2)$ decreases by 2ϵ, while the slack of an arc in in $\{\delta^-(S_1) \backslash \delta^-(S_2)\}$ or $\{\delta^-(S_2) \backslash \delta^-(S_1)\}$ decreases by ϵ.

In general if $S_1, \ldots, S_p$ are minimal ascent sets, this procedure decreases the slack of arcs (at varying rates) until one of the arcs in $\delta^-(S_1) \cup \cdots \cup \delta^-(S_p)$ is tight. If arc (i, j) is in exactly l of the dicuts $\delta^-(S_1), \cdots, \delta^-(S_p)$, then the algorithm decreases its slack at l times the rate of an arc that is in exactly one of the dicuts $\delta^-(S_1), \ldots, \delta^-(S_p)$. In a dual-ascent step of this implementation, we say the *effective slack* of an arc (i, j) contained in exactly l of the dicuts $\delta^-(S_1), \ldots, \delta^-(S_p)$ is s_{ij}/l. A dual-ascent step finds the arc with minimum effective slack in $\delta^-(S_1) \cup \delta^-(S_2) \cup \ldots \cup \delta^-(S_p)$. Suppose the minimum effective slack is ϵ. The dual-ascent algorithm decreases the slack of each arc, decreasing the slack of an arc that is in exactly l dicuts by $l\epsilon$, and increases the dual objective L by $p\epsilon$.

We can determine all minimal ascent sets in $\mathcal{O}(|T||A|)$ steps (the maximum number of root components is $|T|$). To determine the arc that first becomes tight, for each arc we need to determine the rate at which its slack decreases. We can do so by sequentially scanning the arcs and determining how many minimal ascent sets an arc is directed into. Therefore,

[12] Observe that the only nodes that can have a directed path to a root component are the nodes with **req**[i] = 0 and **color**[i] = green at the conclusion of *Root-Components*. Why? Any node i with **color**[i] = red is either a required node or a Steiner node that can be reached from a root component. If the network contains a path from a node with **color**[i] = red to a root component, then either it has a strongly connected component, which is not possible by assumption, or the root component dangles from another required node. This outcome is also not possible because our algorithm is correct. Therefore, we can reduce the size of the graph on which to perform depth first search.

each iteration requires $\mathcal{O}(|T||A|)$ time. Arguing similarly as in Implementation 1 shows that this implementation is an $\mathcal{O}(|T||A|\min(|T||N|,|A|))$ procedure.

Implementation 4

Balakrishnan, Magnanti and Wong [BMW89] described another implementation of the minimal ascent set rule, that we call the *round robin rule.*

This dual-ascent set procedure operates in a very simple, yet special way. Initially, for each commodity the dual-ascent algorithm sets $N(k) = \{D(k)\}$. In a dual-ascent step on commodity k, the algorithm identifies the arc (i^*, j^*) in $\delta^-(N(k))$ with minimum slack (in case of ties, it selects any one of the arcs with minimum slack). The algorithm then adds *exactly one node, the node i^*, to the set $N(k)$.* We say that a commodity k is *active* if the root node is not in $N(k)$; otherwise, commodity k is *inactive.* Note that for an inactive commodity k, the auxiliary network contains a path from the root node to $D(k)$.

The algorithm considers the commodities in a fixed order and cycles through the commodities, performing dual-ascent steps for active commodities, in this fixed order. For example, if the commodities are sorted from $1,\ldots,|K|$ then the algorithm considers commodity $k+1$ after it considers commodity k, and considers commodity 1 after it considers commodity $|K|$.

Balakrishnan, Magnanti and Wong showed that the round robin rule implicitly satisfies the root commodity rule. They showed that the round robin rule simulates the minimal ascent set rule, in the sense that if it ever attempts to perform a dual-ascent step on a set $N(k)$ that is not a minimal ascent set, it does not change the dual variables in that dual-ascent step (i.e., a tight arc is directed into $N(k)$).

Each dual-ascent step of this algorithm can be implemented in $\mathcal{O}(|A|)$ time, since we need to sequentially scan the arcs to find the arc with minimum slack and to update the slack for each arc in $\delta^-(N(k))$. After at most $|N|$ ascent steps for a commodity, the commodity is inactive because after at most $|N|$ ascent steps the root must belong to $N(k)$. Since the number of commodities is $|T|$, the algorithm performs at most $|N||T|$ dual-ascent iterations. Consequently, the dual-ascent algorithm with Balakrishnan, Magnanti and Wong's round robin rule is an $\mathcal{O}(|T||N||A|)$ procedure.

Comment on the Implementations

In this section we showed how to identify all root components in linear time (and how to identify a single minimal ascent set in linear time). We showed that it is possible to contract the digraph D during the course of the dual-ascent algorithm. We then gave some simple implementations of the dual-ascent algorithm. Our analysis of the running times of these algorithms was not very sophisticated and we believe that it is possible, by using

Size ($\|N\|, \|E\|$)	Terminal Nodes	Trials	Ave Gap	Max Gap	Optimally Solved	Ave Time	Max Time
(100,2000)	25	10	0.77%	1.68%	4	8.51	13.09
	50	10	0.00%	0.00%	10	12.06	22.89
	75	10	0.00%	0.00%	10	11.94	25.34
(200,4000)	50	10	0.25%	1.08%	6	23.24	61.76
	100	10	0.04%	0.22%	8	81.35	171.82
	150	10	0.00%	0.00%	10	90.94	208.22
(300,6000)	75	10	0.24%	0.96%	4	197.04	314.13
	150	10	0.02%	0.15%	9	250.63	578.77
	225	10	0.00%	0.00%	10	502.70	860.23

Table 3.2: Computational Experience with Wong's Algorithm: Random edge costs. Time is in seconds on a Sun SPARCstation 10 (Model 41).

with Euclidean edge costs. It optimally solved 71 out of 90 problems with random edge costs, and 50 out of 90 problems with Euclidean edge costs. The average gap between the upper and lower bounds for problems with random edge costs was 0.15 percent, compared with 0.27 percent for problems with Euclidean edge costs.

In these computations, Implementation 4 performs a large number of dual-ascent iterations that do not improve the lower bound. That is, it attempts to perform a large number of basic dual-ascent steps on sets that have a tight arc directed into them (we will elaborate more on this issue in Chapter 6). Consequently, we believe the first three implementations might provide better running times, especially Implementations 1 and 3, where every dual-ascent step strictly increases the lower bound.

3.5 Concluding Remarks

This chapter provides a glimpse at the simplicity, yet effectiveness of dual-ascent methods for network design. We first introduced the basic concepts of dual-ascent using the minimum spanning tree problem as an example. We discussed several models for the minimum spanning tree problem and obtained a better model by directing it. We then developed a dual-ascent algorithm that optimally solves the minimum spanning tree[13] and branching problem. Later, we interpreted the dual-ascent algorithms in the Lagrangian duality framework. This interpretation will prove to be invaluable in Chapter 4.

In the latter part of this chapter, we developed a dual-ascent algorithm for the $\mathcal{NP}$-hard Steiner branching problem by generalizing the dual-ascent algorithm for the branching

[13] Although we have not proved the following result in this thesis, we note that it is possible to interpret two well known algorithms for the minimum spanning tree problem, Prim's algorithm [Pri57] and Kruksal's algorithm [Kru56], as special cases of the dual-ascent algorithm.

problem. As a result, we rederived Wong's algorithm, and provided a different interpretation of his root commodity rule. We then discussed several implementations of Wong's algorithm, and our computational experience with a particular implementation of Wong's algorithm. Our results indicate that Wong's algorithm, with the LIFO dropping heuristic, is a very effective method for finding near optimal solutions for large Steiner tree problems. For reasons we discuss further in Chapter 6, we believe that using Implementation 1, 2, or 3 would lead to better running times.

In the next chapter we study the NDC problem and the DNDC problem, and discuss a dual-ascent approach for them. In Chapter 5 we consider the unitary NDC problem and show how to improve the flow formulation presented in Chapter 1 by directing the model. After that, in Chapter 6 we use the improved formulation presented in Chapter 5 to develop a dual-ascent algorithm for the NDLC problem. We will use many of the ideas developed in this chapter for this algorithm. Finally, in Chapter 7 we develop a directed model for the Steiner forest problem. Based on ideas introduced for this model, we then show how to improve the undirected flow model presented in Chapter 1 for the NDC problem (both unitary and nonunitary).

To conclude this discussion of dual-ascent for the branching and Steiner branching problem, we mention some recent research results concerning the Steiner tree problem and plans for future research.

Empirical evidence indicates that the directed flow formulation for the Steiner tree problem is very tight. This experience has generated interest in bounding the ratio between the values of the optimal IP solution and the optimal LP solution of the directed flow formulation. Recently, Goemans [Goe94a] showed that for the Steiner tree problem with 3 terminal nodes, the IP solution is no more than 6/5 times the optimal solution of the LP relaxation of the directed flow formulation (3.7). This bound is not tight, and the worst IP to LP ratio example that we are aware of is 16/15 [Goe94a].

Recently, we showed [Rag94] that for Steiner tree problems, i.e., Steiner branching problems with $c_{ij} = c_{ji}$, Implementation 3 can be implemented in a very simple way. We showed that *every* strongly connected component that contains a terminal node is a root component. Consequently, Implementation 3 can be implemented without the algorithm to identify root components.

In our future research we plan to continue to study various implementations of Wong's algorithm, aiming to solve very large Steiner tree problems to near optimality.

Chapter 4

A Dual-Ascent Algorithm for the Network Design Problem with Connectivity Requirements

In this chapter we consider the general network design problem with connectivity requirements (NDC). We first show that the cutset and flow formulations introduced in Chapter 1 are equivalent. Then we describe a dual-ascent algorithm, based on the flow formulation, that provides a lower bound and a heuristic solution for the edge-connectivity version of the NDC problem. Later we show how to modify the dual-ascent algorithm to handle node-connectivity problems. We also describe several extensions to the NDC problem and show how to modify the dual-ascent algorithm for these extensions. Next, we show that the dual-ascent algorithm solves any single commodity (in the flow formulation) NDC problem. Thus, the algorithm solves two special cases of the NDC problem: the k-edge-disjoint path problem, and the k-node-disjoint path problem. We also compare the dual-ascent algorithm with Goemans et al.'s algorithm [WGMV93, GGW93, GGP+94] (the GW algorithm) for the edge-connectivity version of the NDC problem and show that in some cases (the k-edge-disjoint path problem) the dual-ascent algorithm provides both a better lower bound and a better heuristic solution than the GW algorithm. Finally, we summarize the contributions of this chapter.

4.1 Comparison of Cutset and Flow-Based Formulations

In Chapter 1 we described two formulations for the NDC problem: a natural cutset formulation (1.1) and an extended flow-based formulation (1.2). In this section we (i) show the equivalence of the cutset and flow-based formulations, and (ii) describe methods for

reducing the number of commodities in the flow formulation. For convenience we restate the two formulations.

Cutset formulation for the NDC problem:

$$\text{Minimize} \quad \sum_{\{i,j\}\in E} c_{ij}x_{ij} \tag{4.1a}$$

$$\text{subject to:} \quad \sum_{\{i,j\}\in\delta(T)} x_{ij} \geq r_{st} \qquad \begin{array}{l}\text{for all pairs } s,t \in N,\ s \neq t \text{ and}\\ \text{for all } T \text{ with } s \in N\backslash T,\ t \in T;\end{array} \tag{4.1b}$$

$$\sum_{\{i,j\}\in\delta_{G-Z}(T)} x_{ij} \geq r_{st} - |Z| \qquad \begin{array}{l}\text{if } r_{st} \geq 2 \text{ with } r_{st} \in \mathbf{R}^{[n]}, \text{ and}\\ \text{for all pairs } s,t \in N,\ s \neq t \text{ and}\\ \text{for all } Z \subset N\backslash\{s,t\} \text{ with } 0 < |Z| < r_{st},\\ \text{and for all } T \subset N\backslash Z; \text{ with } s \notin T,\ t \in T;\end{array} \tag{4.1c}$$

$$x_{ij} \leq b_{ij} \qquad \text{for all } \{i,j\} \in E; \tag{4.1d}$$

$$x_{ij} \geq 0 \qquad \text{and integer, for all } \{i,j\} \in E. \tag{4.1e}$$

Undirected flow formulation for the NDC problem:

$$\text{Minimize} \quad \sum_{\{i,j\}\in E} c_{ij}x_{ij} \tag{4.2a}$$

$$\text{subject to:} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k = \begin{cases} -q_k & \text{if } i = O(k);\\ q_k & \text{if } i = D(k);\\ 0 & \text{otherwise;}\end{cases} \qquad \text{for all } i \in N \text{ and } k \in K \tag{4.2b}$$

$$\left.\begin{matrix} f_{ij}^k \\ f_{ji}^k \end{matrix}\right\} \leq x_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{4.2c}$$

$$\sum_{l\in N} f_{il}^k \leq 1 \qquad \text{for all } k \in K^{[n]} \text{ and all } i \neq O(k), D(k) \tag{4.2d}$$

$$f_{ij}^k, f_{ji}^k \geq 0 \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{4.2e}$$

$$x_{ij} \leq b_{ij} \qquad \text{for all } \{i,j\} \in E; \tag{4.2f}$$

$$x_{ij} \geq 0 \qquad \text{and integer, for all } \{i,j\} \in E. \tag{4.2g}$$

Notice that the cutset formulation is of exponential size. The flow formulation is compact: it has $|K|$ commodities, $|K||N|$ flow balance constraints, $2|K||E|$ constraints of type (4.2c), at most $|K|(|N| - 2)$ constraints of type (4.2d), $2|K||E|$ nonnegativity constraints for the flow variables, $|E|$ constraints of type (4.2f), and $|E|$ nonnegativity and integrality constraints for the ed_{ξ} variables.

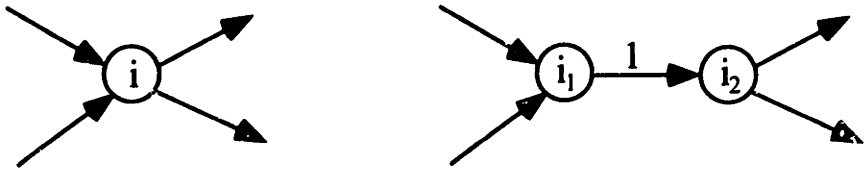


Figure 4-1: Splitting node i into two nodes i_1 and i_2.

4.1.1 Equivalence of Cutset and Flow Formulations

For the minimum spanning tree problem, the max-flow min-cut theorem easily establishes the equivalence of the cutset formulation and the flow formulation. For more general models, the cutset and the flow formulation are also equivalent. However, the result does not immediately follow from the max-flow min-cut theorem on the undirected graph G, but is based on the max-flow min-cut theorem on a related digraph.

Theorem 4.1.1 *The projection of the feasible space of the LP relaxation of the undirected flow formulation for the NDC problem onto the space of the edge variables is the feasible space of the LP relaxation of the cutset formulation for the NDC problem.*

Proof: For problems with only edge-connectivity requirements, this result is easy to establish (the argument is similar to that in Section 3.1). For problems with node-connectivity requirements, we first show that if $(\mathbf{x}, \mathbf{f})$ is feasible for the LP relaxation of the flow formulation, then $\mathbf{x}$ is feasible for the cutset formulation. Consider a pair of nodes s and t with node-connectivity requirement r_{st}. We want to show that the flow formulation implies constraint (4.1c). Because of constraint (4.2d), deleting any subset of p nodes can decrease the flow sent from node s to node t by at most p units. Using the max-flow min-cut theorem on the network obtained by deleting the p nodes gives the result.

Now we show the converse. That is, if $\mathbf{x}$ is feasible to the cutset formulation, then for some choice of flows $\mathbf{f}$, $(\mathbf{x}, \mathbf{f})$ satisfies the flow formulation. Suppose not. Transform the undirected network into the following directed network. Replace each edge $\{i, j\}$ by two arcs (i, j) and (j, i) each with capacity x_{ij}. Next perform the following well-known node splitting operation (see Figure 4-1). Replace each node i by two nodes i_1 and i_2. Arcs directed into node i are now directed into node i_1 and arcs directed out of node i are now directed out of node i_2. To complete the node splitting operation, add the arc (i_1, i_2) with capacity 1. This arc ensures that no more than 1 unit of flow of a commodity flows through node i. Denote the nodes and arcs of the transformed directed network D' by N' and A' respectively. Note that we can send t units of flow of commodity k from node $O(k)$ to node $D(k)$ in D' if and only if we can send t units of flow of commodity k from $O(k)$ to $D(k)$ in the undirected network G with (i) x_{ij} as edge capacities, (ii) constraint (4.2d) satisfied.

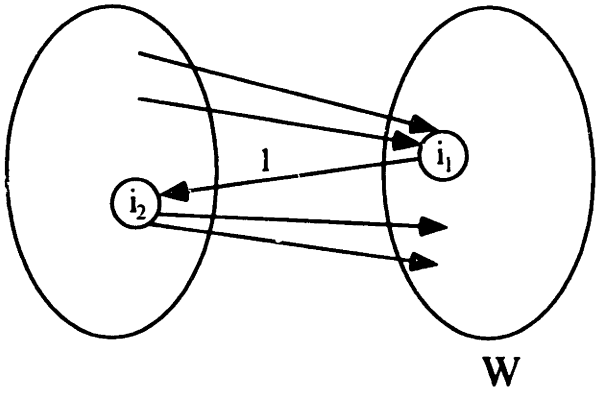
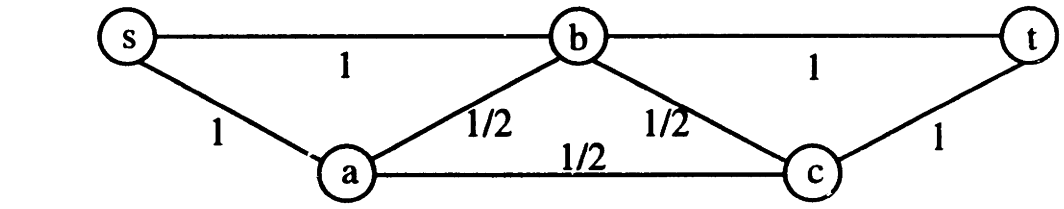


Figure 4-2: Minimum dicut $\delta^-(W)$ can be transformed into a dicut such that, if $i_1 \in W$, then $i_2 \in W$ by transferring either i_2 to W or i_1 out of W.

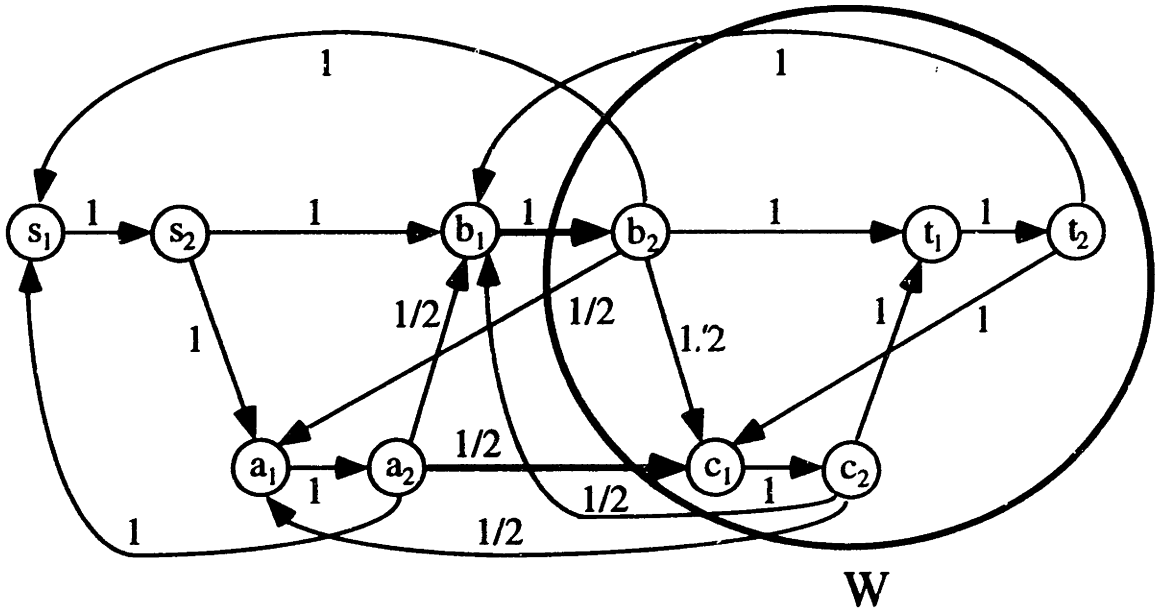
By assumption, for some commodity k, the maximum amount of flow that can be sent from node $O(k)$ to node $D(k)$ in the undirected network, while satisfying conditions (i) and (ii), is strictly less than q_k. The max-flow min-cut theorem on the directed network implies that the capacity of the minimum dicut between $O(k)$ and $D(k)$ is strictly less than q_k units. Let $\delta^-(W)$ be the minimum dicut. We claim that some minimum dicut $\delta^-(W)$ satisfies the property that if $i_1 \in W$, then $i_2 \in W$. If some minimum dicut has $i_1 \in W$ and $i_2 \notin W$, then either by transferring i_2 to W or by transferring i_1 out of W, we obtain a dicut whose capacity is no larger than that of $\delta^-(W)$ (see Figure 4-2).

Consider the arcs in the dicut $\delta^-(W)$ that enforce node capacities (that is, those of the form (i_1, i_2)). Let Z be the set of nodes in the original network G that created, by the node splitting operation, these arcs. The sum of the capacities of the arcs in $\delta^-(W)$ other than those that enforce node capacities is strictly less than $q_k - |Z|$. Nodes created by performing the node splitting operation on nodes in $G - Z$ are either both in W, or both in W's complement. Define T as follows: if j_1 and j_2 both belong to W, then $j \in T$. The nodes $s = O(k)$ and $t = D(k)$, together with the sets T and Z, violate constraint (4.1c). This contradiction establishes the theorem. ■

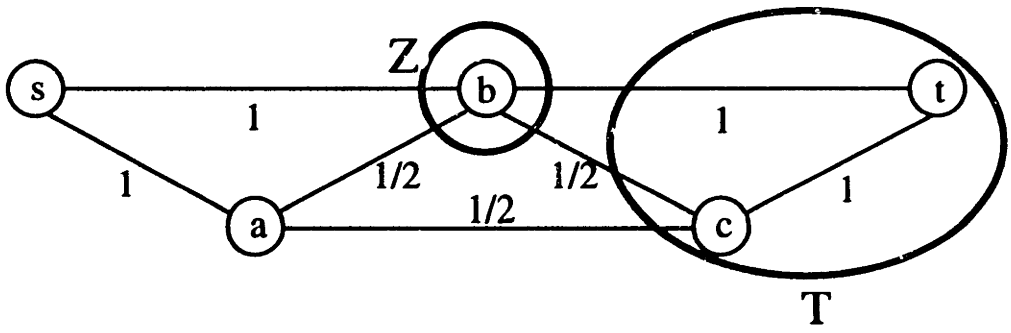
Figure 4-3 illustrates the cut transformation procedure described in the proof of Theorem 4.1.1. In the example, node s is the origin and node t is the destination of the commodity. The connectivity requirement r_{st} between node s and node t is $2^{\lfloor n \rfloor}$, and so the requirement for the commodity between nodes s and t is 2 units. Figure 4-3b shows the transformed directed network and the minimum dicut $\delta^-(W)$. The minimum dicut $\delta^-(W)$ has a capacity of 1.5 units. (b_1, b_2) is the only arc in the dicut $\delta^-(W)$ that enforces node capacities. Thus $Z = \{b\}$. Since t_1, t_2 and c_1, c_2 belong to W, $T = \{c, t\}$.



(a)



(b)



(c)

Figure 4-3: (a) The undirected network and edge capacities. Node s is the origin and node t is the destination. (b) The transformed directed network and its minimum dicut $\delta^-(W)$. W and the dicut $\delta^-(W)$ are in bold. (c) Using the dicut $\delta^-(W)$ to determine the sets Z and T in the undirected network.

4.1.2 Decreasing the Number of Commodities in the Flow Formulation

The flow formulation for the NDC problem has $\mathcal{O}(|K|(|E|+|N|))$ constraints and $\mathcal{O}(|K||E|)$ variables. A simple, and naive, way to determine the number of commodities in the flow formulation is to create a commodity for every pair of nodes that have a connectivity requirement. As an example, consider the minimum cost k-node-connected spanning subgraph problem. Suppose the underlying graph has 20 nodes and 200 edges. If the connectivity requirements are enforced by creating a commodity for every node pair, then the model has 190 commodities. Consequently, the model will contain $190 \times 20 = 3,800$ flow balance constraints, $190 \times 200 \times 2 = 76,000$ constraints of type (4.2c), $190 \times 18 = 3,520$ constraints of type (4.2d), 76,000 nonnegativity constraints for the flow variables, 200 constraints of type (4.2f), and 200 nonnegativity and integrality constraints for the edge variables.

As this example shows the flow formulation can be quite large. By using fewer commodities, if possible, we could reduce the size of the formulation. We now describe some edge-connectivity and node-connectivity properties that enable us to decrease the number of commodities needed to model the connectivity requirements.

Edge Connectivity

First consider edge-connectivity problems. Gomory and Hu [GH61] established the following result for the network synthesis problem.[1] Given the connectivity requirements matrix $\mathbf{R}$, create a "requirement" graph G^R on the node set N as follows: if $r_{ij} = w$ and r_{ij} is an edge-connectivity requirement, then G^R has an edge of weight w between node i and node j. Gomory and Hu showed that it is sufficient to consider the connectivity requirements only for the edges on the maximum spanning tree of this graph. It is easy to verify this result using the max-flow min-cut theorem and the maximum spanning tree optimality conditions. The path optimality condition for the maximum spanning tree states that a spanning tree is a maximum spanning tree if and only if it satisfies the following condition: For every nontree edge $\{k,l\}$ of G^R, $r_{ij} \geq r_{kl}$ for every edge $\{i,j\}$ contained in the (tree) path P on the maximum spanning tree connecting nodes k and l. As a result, if we satisfy the requirements of the maximum spanning tree, the network designed has sufficient capacity to satisfy the requirements of nontree edges. (By the max-flow min-cut theorem, the network has sufficient capacity to connect nodes k and l if every cut in the network design separating these nodes has capacity at least r_{kl}. Since some pairs i,j of nodes on the path P must lie on opposite sides of this cut, it must have capacity at least $r_{ij} \geq r_{kl}$.)

Gomory and Hu's result permits us to model the edge-connectivity requirements in any

[1] Recall, in the network synthesis problem we are given a symmetric requirement matrix $\mathbf{R}$ whose entries can be fractional. We need to decide on capacities b_{ij} for the edges so that for every pair of nodes s and t, the network can send at least r_{st} units of flow.

NDC problem with $|N| - 1$ or fewer commodities. We compute the maximum spanning tree of the requirement graph and create commodities only for those edges of the maximum spanning tree with nonzero weight. Since the spanning tree has $|N| - 1$ edges and finding it requires $\mathcal{O}(|E| + |N| \log |N|)$ time, this procedure creates at most $|N| - 1$ commodities (we will not create commodities for zero weight edges of the maximum spanning tree) and requires $\mathcal{O}(|E| + |N| \log |N|)$ time.

Node Connectivity

We now describe a result that enables us to decrease the number of commodities required to model node-connectivity requirements in the undirected flow formulation.

Theorem 4.1.2 *The minimum cost k-node-connected spanning subgraph problem can be modeled with* $\lceil k|N|/2 \rceil$ *commodities.*

Proof: Let $G = (N, E)$ be the given graph and let G^* be any arbitrary k-node-connected graph on the nodes N. We will choose commodities in the flow formulation (4.2) (on $G = (N, E)$) as follows. For every edge $\{i, j\} \in G^*$ create a commodity l in $K^{[n]}$ between nodes i and j with a demand of k units, arbitrarily choosing one of them as the origin and the other as the destination. Suppose that as a solution to model (4.2) we find a minimum cost subgraph G' of G that satisfies the connectivity requirements imposed by these commodities. We claim that G' is k-node connected. Suppose not. Then deleting some set S of $k - 1$ or fewer nodes disconnects G'. Let i and j be any two disconnected nodes in $G' - S$. Now consider G^*. Since G^* is k-node connected, the graph $G^* - S$ contains a path P from node i to node j. Because there is a commodity for every edge in G^*, the endpoints of any edge on the path P remain connected in $G' - S$. But this implies i and j are connected in $G' - S$, contradicting our assumption. Therefore, G' is k-node connected.

We now show that modeling the minimum cost k-node-connected spanning subgraph problem in this manner does not change the objective value of its LP relaxation. That is, if instead of defining a commodity for each pair of nodes in G, we define a commodity only for every node pair i, j corresponding to an edge in G^*, the objective value of the LP relaxation of model (4.2) does not decrease. Consider the solution $(\mathbf{x}, \mathbf{f})$ obtained by solving the LP relaxation of the undirected flow formulation with commodities by the edges in G^*. We show that it is possible to send k units of flow from any node s to any other node t in the graph G with edge capacity vector $\mathbf{x}$, so that no more than one unit flows into any intermediate node. By Theorem 4.1.1, this is possible if for every set of nodes Z, with $|Z| < k$ and $s, t \notin Z$, the value of the minimum s-t cut in the network $G - Z$ is greater than or equal to $k - |Z|$. Suppose we delete a set Z of $p < k$ nodes, other than s and t, from the graph G. Then, because G^* is k-node connected, G^* contains $k - p$ node-disjoint paths

from node s to node t. Consider any one of these paths P. Then, in the network $G - Z$ with edge capacity x_{ij} for each edge $\{i,j\}$, it is possible to send $k-p$ units of flow between the endpoints of every edge of the path P. By the max-flow min-cut theorem, every s-t cut in $G - Z$ has capacity greater than $k - p$. Therefore, it is possible to send $k - p$ units of flow between node s and node t while observing the edge capacities x_{ij} in $G - Z$. As a result, the objective value of the LP relaxation remains unchanged.

This argument shows that we can model the minimum cost k-node-connected spanning subgraph problem on a graph $G = (N, E)$, by considering any arbitrary k-node-connected graph G^* on the nodes N and creating a commodity for each edge in G^*. The fewest possible commodities, with this approach, depends on the minimum number of edges in a k-node-connected graph on the nodes N. Observing that the degree of each node in a k-node-connected graph is at least k, a trivial lower bound on the number of edges in a $|N|$ node k-node-connected graph is $\lceil k|N|/2 \rceil$. A very nice result due to Harary [Har62], that we mentioned in Chapter 2, constructively shows how to achieve this bound. ■

We now state Harary's result that permits us to construct a k-node-connected graph on $|N|$ nodes with exactly $\lceil k|N|/2 \rceil$ edges. Harary [Har62] or the textbook by Bondy and Murty [BM76] contains a proof of this result.

Theorem 4.1.3 (Harary) *The minimum number of edges in a $|N|$ node k-node-connected graph is equal to $\lceil k|N|/2 \rceil$.*

Construct k-node-connected graphs on N nodes, with $\lceil k|N|/2 \rceil$ edges, as follows.

1. If k is even, construct a graph $H(k, |N|)$ with nodes $0, 1, \ldots, |N| - 1$. The graph contains an edge between nodes i and j if $i - (k/2) \leq j \leq i + (k/2)$ (addition and subtraction are modulo $(k/2)$). Figure 4-4a shows $H(4, 8)$.

2. If k is odd and $|N|$ is even, construct $H(k, |N|)$ from the graph $H(k-1, |N|)$ by adding edges between node i and node $i + (|N|/2)$, for $i = 1, \ldots, |N|/2$. Figure 4-4b shows $H(5, 8)$.

3. If k is odd and $|N|$ is odd, then construct $H(k, |N|)$ from $H(k-1, |N|)$ by adding edges joining node 0 to nodes $\frac{|N|-1}{2}$ and $\frac{|N|+1}{2}$, and node i to node $i + \frac{|N|+1}{2}$, for $1 \leq i \leq \frac{|N|-1}{2}$. Figure 4-4c shows $H(5, 9)$.

The following lemma describes another useful property.

Lemma 4.1.4 *Suppose A is a k-node-connected set of nodes[2] and B is a l-node-connected set of nodes, and $k > l$. If $|A \cap B| = l$, then $A \cup B$ is l-node connected.*

[2]We say a set of nodes is k-node-connected if a network contains k node-disjoint paths between every pair of nodes in the set.

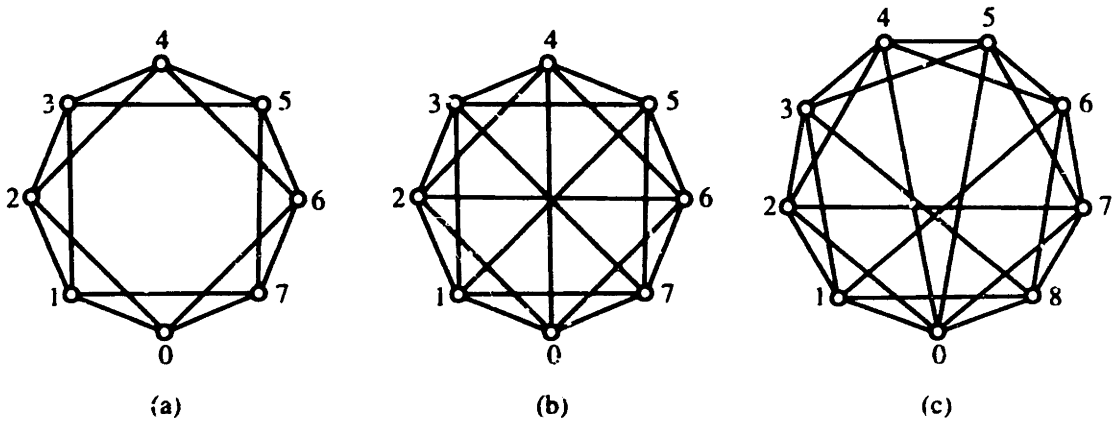


Figure 4-4: Harary's construction. (a) $H(4,8)$, (b) $H(5,8)$, and (c) $H(5,9)$.

Proof: Let $a_1, a_2, \ldots, a_l$ denote the nodes in $A \cap B$. Consider any node i in $A \backslash B$ and any node j in B. Since A is k-node connected, deleting a set S of $l-1$ nodes in the graph leaves a path between i and one of the nodes $a_1, a_2, \ldots, a_l$, say a_t. Since B is l-node connected, deleting S leaves a path between a_t and j. ∎

This lemma helps decrease the number of commodities because it implies that whenever $|A \cap B| \geq l$, instead of considering all the nodes in $A \cap B$, we can choose any l nodes in A, and ensure that these l nodes along with the nodes in $B \backslash A$ are l-node connected.

Survivable Network Design Problem

We now show how to model the node-connectivity version of the survivable network design problem (SND) using $\mathcal{O}(|N|)$ commodities, assuming a fixed upper bound on the maximum connectivity requirement. Recall that in the node-connectivity version of the SND problem, each node v in the graph has a node-connectivity requirement r_v. For each distinct pair $\{s,t\}$ of nodes, the network must contain at least $r_{st} = \min\{r_s, r_t\}$ node-disjoint s-t paths. Therefore, in the SND problem, all nodes with $r_v \geq k$ must be k-node connected. Let N_k denote the set of nodes with connectivity requirement k. Let $q = r_{\max}$. Observe that the nodes in N_q need to be q-node connected. Model these requirements as described in Theorem 4.1.3. Next find N_p, the highest indexed nonempty set in $N_1, \ldots, N_{q-1}$. If $|N_q| > p$ arbitrarily choose any p nodes in N_q and add these nodes to N_p. Otherwise, add all nodes in N_q to N_p. Set $q = p$ and continue until the connectivity requirements of the lowest indexed nonempty set N_i have been modeled.

This procedure creates $|N_{r_{\max}}|$ commodities for the set of nodes with the highest connectivity requirement. It adds at most p nodes to the set of nodes[3] N_p. Therefore, the

[3]We let N_p denote the set of nodes with connectivity requirement $r_v = p$.

procedure creates at most

$$\left\lceil \frac{r_{\max}|N_{r_{\max}}|}{2} \right\rceil + \sum_{\{i:i\neq r_{\max},|N_i|>0\}} \left\lceil \frac{i(N_i+i)}{2} \right\rceil$$

commodities. For a constant value of $r_{\max}$ the number of commodities is $\mathcal{O}(|N|)$.

Let us illustrate this procedure on a 30 node SND problem. Suppose 20 nodes have a connectivity requirement of 5 and 10 nodes have a connectivity requirement of 3. The procedure first creates $\lceil 5 \times 20/2 \rceil = 50$ commodities to model the connectivity requirements between nodes with $r_v = 5$. Then, since N_3 is the highest indexed nonempty set in $N_1, \ldots, N_4$, it arbitrarily adds any 3 nodes in N_5 (nodes with connectivity requirement 5) to N_3. N_3 now has $10 + 3 = 13$ nodes and the procedure creates $\lceil 3 \times 13/2 \rceil = 20$ commodities to model the three connectivity requirements in N_3, giving a total of 70 commodities to model the connectivity requirements. (Compare with $30 \times 29/2 = 435$ commodities to model the connectivity requirements with the naive approach.) Suppose we design a network that satisfies these requirements. Then, in this network all nodes with a connectivity requirement of five are 5-node connected and all nodes with a connectivity requirement of three are 3-node connected. Furthermore, because N_3 included three nodes from N_5, by Lemma 4.1.4 the network contains at least 3 node-disjoint paths between any node with a connectivity requirement of 5 and any node with a connectivity requirement of 3.

General NDC Problem

Consider node-connectivity requirements. Let $G^{R^{[n]}}$ be the graph defined by the node-connectivity requirements in $\mathbf{R}$ as follows. $G^{R^{[n]}}$ has an edge of weight w between node i and node j if $r_{ij} = w^{[n]}$. Let $G_q^{R^{[n]}}$ denote the subgraph defined by the edges of $G^{R^{[n]}}$ with weight q. In order to effectively use the reduction procedures to model the node-connectivity requirements of general NDC problems, we need an algorithm to identify the q-node-connected components of $G_q^{R^{[n]}}$. Unfortunately, to our knowledge, efficient algorithms to recognize q-node-connected subgraphs of a graph are known[4] only for $q = 1, 2, 3$. Therefore, it seems that although Theorem 4.1.3 and Lemma 4.1.4 are very useful in decreasing the number of commodities, an efficient algorithm to find a k-node-connected subgraph of a graph is needed in order to devise a general procedure to reduce the number of commodities to model node-connectivity requirements in NDC problems.

We now show how to use Theorem 4.1.3 and Lemma 4.1.4 once we have identified the

[4] A 1-node-connected component is a tree. Consequently, a breadth first search or a depth first search algorithm finds the 1-node-connected components. The textbook by Even [Eve79] gives an algorithm (based on depth first search) to identify 2-node-connected components of a graph. [FRT89] studies the problem of finding 3-node-connected subgraphs of a graph.

q-node-connected subgraphs of $G_q^{R[n]}$ for $q = 1, \ldots, r_{max}$. Set $q = r_{max}$. For each q-node-connected component select commodities as described in Theorem 4.1.3. For every edge $\{i, j\}$ in $G_q^{R[n]}$ whose endpoints are not in the same q-node-connected component, create a commodity in the graph G with connectivity requirement q choosing one of the endpoints as the origin and one of the endpoints as the destination. Set $q = q - 1$. Continue as before, except use Lemma 4.1.4 to decrease the number of nodes in each q-node-connected component in $G_q^{R[n]}$.

We have discussed one approach for decreasing the size of the flow model. Another approach decreases the size of both the flow and cutset formulation by decomposing the NDC problem into smaller problems that can be solved in a certain order or independently of each other. Typically, these decomposition procedures are useful and effective on sparse graphs. Grötschel, Monma and Stoer [GMS92a, Sto92] describe some decomposition procedures for SND problems. By combining the decomposition procedures described in [GMS92a, Sto92] and the commodity reduction procedures described in this section, we might be able to significantly decrease the size of the flow formulation for the SND problem.

4.1.3 Summary

In this section we showed the equivalence of the cutset and flow formulation. We also described methods for minimizing the number of commodities in the flow formulation. We showed

- how to model the edge-connectivity version of the NDC problem with $|N| - 1$ commodities,
- how to model the node-connectivity version of the SND problem with a linear number of commodities, and
- how to minimize the number of commodities in the general NDC problem.

Note, however, that for large-scale problems, even if the number of commodities is linear in the number of nodes, the flow formulation can be fairly large. For the general NDC problem, assuming the number of commodities is a linear function of the number of nodes in the graph, the flow model has $\mathcal{O}(|N|^2 + |N||E|)$ constraints and $\mathcal{O}(|E||N|)$ variables. Since the flow formulation can be fairly large, solving even its LP relaxation poses a significant computational challenge. However, due to the structure of the problem, dual-ascent (and Lagrangian relaxation) might be an attractive method for obtaining lower bounds and heuristic solutions.

In the next section we discuss a dual-ascent algorithm for the NDC problem. We first restrict our attention to problems with only edge-connectivity requirements, and then show how to handle node-connectivity requirements and other extensions.

4.2 A High-Level View of the Dual-Ascent Algorithm

Consider the undirected flow formulation (on page 84) for the NDC problem. Since flow costs are zero, there is no advantage in sending flow around a cycle. Therefore, we can replace constraint (4.2c) by $f_{ij}^k + f_{ji}^k \leq x_{ij}$ (for all $(i,j) \in E$ and $k \in K$). Although this change in the model does not alter the LP relaxation of the undirected flow formulation,[5] the dual-ascent algorithm is simpler for the modified model since it has only one dual variable, instead of two, for each f_{ij}^k, f_{ji}^k pair. Consider the dual to the LP relaxation of the undirected flow formulation for the NDC problem.

Dual to undirected flow formulation for the NDC problem:

$$\text{Maximize} \quad \sum_{k \in K} q_k v_{D(k)}^k - \sum_{\{i,j\} \in E} b_{ij} u_{ij} \tag{4.3a}$$

$$\text{subject to:} \quad \left.\begin{array}{r} v_j^k - v_i^k \\ v_i^k - v_j^k \end{array}\right\} \leq w_{ij}^k \quad \text{for all } \{i,j\} \in E \tag{4.3b}$$

$$\sum_{k \in K} w_{ij}^k - u_{ij} \leq c_{ij} \quad \text{for all } \{i,j\} \in E \tag{4.3c}$$

$$u_{ij} \geq 0 \quad \text{for all } \{i,j\} \in E \tag{4.3d}$$

$$w_{ij}^k \geq 0 \quad \text{for all } \{i,j\} \in E. \tag{4.3e}$$

In this model, u_{ij} is the dual variable for the constraint $x_{ij} \leq b_{ij}$, v_j^k is the dual variable for constraint (4.2b) at node j for commodity k, and w_{ij}^k is the dual variable for the $f_{ij}^k + f_{ji}^k \leq x_{ij}$ constraint. As before, since one of the flow balance constraints is redundant, we set $v_{O(k)}^k = 0$.

Given any values for $\mathbf{w}$ and $\mathbf{u}$, the dual separates by commodity as $\{\max q_k v_{D(k)}^k$, subject to $v_j^k - v_i^k \leq w_{ij}^k$, $v_i^k - v_j^k \leq w_{ij}^k$, for all $\{i,j\} \in E\}$. If q_k is 1, the subproblem for commodity k is the dual to a shortest path problem between the nodes $O(k)$ and $D(k)$ on a graph G with edge costs w_{ij}^k. If q_k exceeds 1, the optimal solution to the subproblem for commodity k is simply q_k times the shortest path length.

Let s_{ij} be the slack in the constraint $\sum_{k \in K} w_{ij}^k - u_{ij} \leq c_{ij}$ (i.e., $s_{ij} = c_{ij} + u_{ij} - \sum_{k \in K} w_{ij}^k$). As before, if the slack is zero, we say the edge is *tight*. We also define the *auxiliary network* by the tight edges.

Our subsequent development will use the dual, as viewed in the context of the following

[5] We can convert any solution to the undirected flow formulation to one that satisfies this constraint with no change in objective function value. Suppose $(\mathbf{x}, \mathbf{f})$ is a feasible solution to the undirected flow formulation. Then $(\mathbf{x}, \bar{\mathbf{f}})$, with $\bar{f}_{ij}^k = \max(0, f_{ij}^k - f_{ji}^k)$ and $\bar{f}_{ji}^k = \max(0, f_{ji}^k - f_{ij}^k)$ for all $\{i,j\}$ and $k \in K$, is a feasible solution to the undirected flow formulation that satisfies the constraint $\bar{f}_{ij}^k + \bar{f}_{ji}^k \leq x_{ij}$. Since only the edge variables, which we do not modify, have associated costs, $(\mathbf{x}, \bar{\mathbf{f}})$ has the same cost as $(\mathbf{x}, \mathbf{f})$.

Lagrangian duality approach. Let v_i^k, w_{ij}^k and u_{ij} be Lagrange multipliers for the flow balance constraint, the $f_{ij}^k + f_{ji}^k \leq x_{ij}$ constraint, and the $x_{ij} \leq b_{ij}$ constraint of the undirected flow formulation for the NDC problem; we obtain the following Lagrangian function:

$$\begin{aligned} L(\mathbf{x},\mathbf{f},\mathbf{v},\mathbf{w},\mathbf{u}) \;=\; & \sum_{\{i,j\}\in E} c_{ij}x_{ij} + \sum_{i\in N}\sum_{k\in K}(d_i^k - \sum_{j\in N} f_{ji}^k + \sum_{l\in N} f_{il}^k)v_i^k + \\ & \sum_{\{i,j\}\in E}\sum_{k\in K}(f_{ij}^k + f_{ji}^k - x_{ij})w_{ij}^k + \sum_{\{i,j\}\in E}(x_{ij} - b_{ij})u_{ij}. \end{aligned}$$

In this equation, d_i^k denotes the demand for commodity k at node i. Rearranging the terms, we obtain

$$\begin{aligned} L(\mathbf{x},\mathbf{f},\mathbf{v},\mathbf{w},\mathbf{u}) \;=\; & \sum_{i\in N}\sum_{k\in K} d_i^k v_i^k - \sum_{\{i,j\}\in E} b_{ij}u_{ij} + \sum_{\{i,j\}\in E}(c_{ij} + u_{ij} - \sum_{k\in K} w_{ij}^k)x_{ij} + \\ & \sum_{\{i,j\}\in E}\sum_{k\in K}((v_i^k - v_j^k + w_{ij}^k)f_{ij}^k + (v_j^k - v_i^k + w_{ij}^k)f_{ji}^k). \end{aligned} \tag{4.4}$$

We obtain the value $Z(\mathbf{v},\mathbf{w},\mathbf{u})$ of the Lagrangian dual function for any vector $(\mathbf{v},\mathbf{w},\mathbf{u})$ by minimizing the Lagrangian function.

$$Z(\mathbf{v},\mathbf{w},\mathbf{u}) = \min_{(\mathbf{x},\mathbf{f})} \left\{ L(\mathbf{x},\mathbf{f},\mathbf{v},\mathbf{w},\mathbf{u}) \middle| \begin{array}{l} x_{ij} \geq 0 \text{ and integer for all } \{i,j\} \in E, \\ f_{ij}^k, f_{ji}^k \geq 0 \text{ for all } \{i,j\} \in E \text{ and } k \in K. \end{array} \right\}.$$

Because the Lagrangian function $L(\mathbf{x},\mathbf{f},\mathbf{v},\mathbf{w},\mathbf{u})$ is separable in the flow and edge variables, its minimization decomposes by edge for the edge variables $\mathbf{x}$ and by edge and commodity for the flow variables $\mathbf{f}$. If $c_{ij} + u_{ij} - \sum_{k\in K} w_{ij}^k \geq 0$, then $x_{ij} = 0$, and if $c_{ij} + u_{ij} - \sum_{k\in K} w_{ij}^k < 0$, then $Z(\mathbf{v},\mathbf{w},\mathbf{u})$ is unbounded from below. Similarly, if $v_i^k - v_j^k + w_{ij}^k \geq 0$ and $v_j^k - v_i^k + w_{ij}^k \geq 0$, then $f_{ij}^k = 0$ and $f_{ji}^k = 0$. Otherwise, if either $v_i^k - v_j^k + w_{ij}^k < 0$ or $v_j^k - v_i^k + w_{ij}^k < 0$, then $Z(\mathbf{v},\mathbf{w},\mathbf{u})$ is unbounded from below. The Lagrangian dual problem is

$$\begin{array}{rl} \text{Maximize} & Z(\mathbf{v},\mathbf{w},\mathbf{u}) \\ \text{subject to:} & \mathbf{w} \geq \mathbf{0}; \\ & \mathbf{u} \geq \mathbf{0}; \\ & \mathbf{v} \quad \text{unrestricted in sign.} \end{array}$$

Suppose we are given a node potential vector $\bar{\mathbf{v}}$. Observe that by setting $w_{ij}^k \geq \max(\bar{v}_j^k - \bar{v}_i^k, \bar{v}_i^k - \bar{v}_j^k)$ and, once the value of $\mathbf{w}$ is chosen, by setting $u_{ij} \geq \max(0, \sum_{k\in K} w_{ij}^k - c_{ij})$, we ensure that the Lagrangian function is bounded. Furthermore, since u_{ij} has a negative

coefficient in the objective function of the Lagrangian function, and the minimum value of u_{ij} depends upon the w_{ij}^k values (once w_{ij}^k is sufficiently large, u_{ij} increases linearly with respect to w_{ij}^k), we obtain the optimal value of the Lagrangian function by setting w_{ij}^k and u_{ij} to their minimum possible values. Therefore, $w_{ij}^k = \max(\bar{v}_j^k - \bar{v}_i^k, \bar{v}_i^k - \bar{v}_j^k)$ and $u_{ij} = \max(0, \sum_{k \in K} w_{ij}^k - c_{ij})$. Substituting the expression for the optimal value of w_{ij}^k (given $\mathbf{v}$) in the equation for the optimal value of u_{ij} (given $\mathbf{w}$), we obtain $u_{ij} = \max(0, \sum_{k \in K}(\max(\bar{v}_j^k - \bar{v}_i^k, \bar{v}_i^k - \bar{v}_j^k)) - c_{ij})$. Therefore, we can write the Lagrangian dual problem as a maximization problem over the node potentials $\mathbf{v}$. The Lagrangian dual problem is

$$\begin{aligned} &\text{Maximize } Z(\mathbf{v}) \\ &\text{subject to no constraint on } \mathbf{v} \end{aligned}$$

with

$$Z(\mathbf{v}) = \sum_{k \in K} \sum_{i \in N} d_i^k v_i^k - \sum_{\{i,j\} \in E} b_{ij} \overbrace{\max(0, \sum_{k \in K} \underbrace{(\max(v_j^k - v_i^k, v_i^k - v_j^k))}_{w_{ij}^k} - c_{ij})}^{u_{ij}}. \quad (4.5)$$

Observe that for any commodity, increasing all node potentials by the same amount does not change the value of the Lagrangian dual function. Therefore, for each commodity we fix the value of the node potential for one node. We set $v_{O(k)}^k = 0$ for all commodities. With slight modifications, our derivation in Section 3.3 shows the LP dual and the Lagrangian dual problems are equivalent. So, as before, we refer to Lagrangian dual function and the Lagrangian dual problem as the dual function and dual problem respectively.

4.2.1 Fundamental Ideas

In designing the dual-ascent algorithm, we consider the complementary slackness conditions:

$$\begin{aligned} & x_{ij} > 0 \Longrightarrow \sum_{k \in K} w_{ij}^k - u_{ij} = c_{ij}; \\ & f_{ij}^k > 0 \Longrightarrow v_j^k - v_i^k = w_{ij}^k; \\ & u_{ij} > 0 \Longrightarrow x_{ij} = b_{ij}; \\ \text{and} \quad & w_{ij}^k > 0 \Longrightarrow f_{ij}^k + f_{ji}^k = x_{ij}. \end{aligned} \quad (4.6)$$

By design, the dual-ascent algorithm will maintain the first two complementary slackness conditions, which we refer to as the *primal complementary slackness conditions*. The first

complementary slackness condition permits the primal solution to include only tight edges in the auxiliary network. The second condition introduces a commodity-based notion of direction for the edges of the network. If commodity k flows on an edge $\{i,j\}$ from node i to node j, then $v_j^k - v_i^k = w_{ij}^k$. Since $w_{ij}^k \geq 0$, this condition states that commodity k can flow along edge $\{i,j\}$ from node i to node j only if $v_j^k \geq v_i^k$. Therefore, the node potentials effectively direct the edges. If $v_j^k > v_i^k$, commodity k can flow on edge $\{i,j\}$ only in the direction i to j. Similarly, if $v_i^k > v_j^k$, then commodity k can flow on edge $\{i,j\}$ only in the direction j to i. Note that since different commodities can send flow on edges in opposite directions, the direction of an edge is commodity dependent.

To simplify our notation, for each commodity we consider directed graphs $D^k = (N, A^k)$ for $k = 1, \ldots, K$, each defined on the same node set N. To obtain the set of arcs A^k, we replace each edge $\{i,j\}$ by two directed arcs $(i,j)^k$ and $(j,i)^k$, *each with the same* s_{ij}, w_{ij}^k and u_{ij} values as the undirected edge $\{i,j\}$. We say that an arc $(i,j)^k$ is *active* if $v_j^k \geq v_i^k$ and $s_{ij} = 0$. We refer to any arc that is not active as *inactive*. For any commodity k, we define its *active network* by the active arcs of D^k.

Note that if an arc $(i,j)^k$ is active, we can set $x_{ij} > 0$ and $f_{ij}^k > 0$ in a primal solution and satisfy the primal complementary slackness conditions (Recall $w_{ij}^k = \max(v_j^k - v_i^k, v_i^k - v_j^k)$. Thus if $v_j^k - v_i^k \geq 0$, $w_{ij}^k = v_j^k - v_i^k$). Observe that if arc $(i,j)^k$ is inactive and $f_{ij}^k > 0$, then the primal and dual solutions violate either the first, second or both the primal complementary slackness conditions. Thus, we can state primal complementary slackness in the following equivalent way, which will prove to be useful in our subsequent discussion.

Equivalent Definition of Primal Complementary Slackness: Every inactive arc for commodity k (that is an arc $(i,j)^k$ with a positive slack or $v_i^k > v_j^k$) has zero flow.

A Better Understanding of the Dual Function

Let us now consider the dual function $Z(\mathbf{v})$. First consider the effect of changing the node potential v_j^k of a single commodity k at a single node j on the term u_{ij} of the dual function ($u_{ij} = \max(0, \sum_{k \in K}(\max(v_j^k - v_i^k, v_i^k - v_j^k)) - c_{ij})$). Let s_{ij}^k denote the term $c_{ij} - \sum_{l \neq k} \max(v_j^l - v_i^l, v_i^l - v_j^l)$. Note that s_{ij}^k remains fixed if we fix all the node potentials except v_j^k. Consequently, $u_{ij} = \max(0, \max(v_j^k - v_i^k, v_i^k - v_j^k) - s_{ij}^k)$ and $s_{ij} = \max(0, s_{ij}^k - \max(v_j^k - v_i^k, v_i^k - v_j^k))$. Figure 4-5 shows the value of u_{ij} as a function of v_j^k with all the other node potentials held fixed. Notice that when v_j^k is very small, u_{ij} is positive since $v_i^k - v_j^k > v_j^k - v_i^k$ and $v_i^k - v_j^k > s_{ij}^k$. As v_j^k increases, the value of u_{ij} decreases linearly as a function of v_j^k until either (a) $v_j^k = v_i^k$, or (b) u_{ij} becomes zero (at $v_j^k = v_i^k - s_{ij}^k$). In case (a), $s_{ij}^k \leq 0$ and at $v_j^k = v_i^k$, $u_{ij} = -s_{ij}^k \geq 0$. Since the slack $s_{ij} = \max(0, s_{ij}^k - \max(v_j^k - v_i^k, v_i^k - v_j^k))$, at $v_j^k = v_i^k$ the slack is equal to zero and any further increase in v_j^k increases u_{ij}. Figure 4-5a

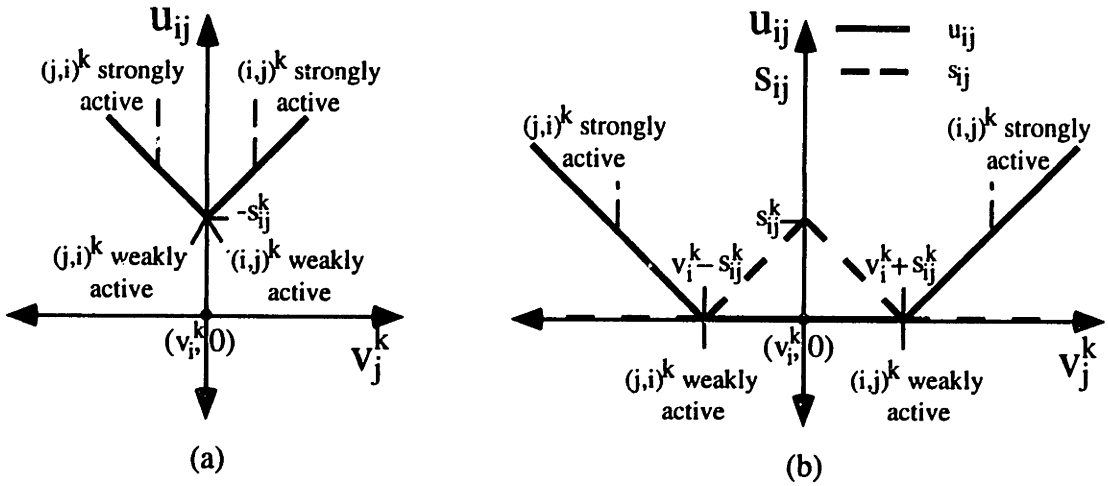


Figure 4-5: u_{ij} as a function of node potential v_j^k. (a) Case (a): $s_{ij}^k \leq 0$; (b) Case (b): $s_{ij}^k > 0$.

shows case (a). In case (b), $s_{ij}^k > 0$. The value of u_{ij} is equal to zero when $v_j^k - v_i^k \leq s_{ij}^k$ and $v_i^k - v_j^k \leq s_{ij}^k$. From $v_j^k = v_i^k - s_{ij}^k$ any further increase in v_j^k increases the slack, maintaining $u_{ij} = 0$, until $v_j^k = v_i^k$. After $v_j^k = v_i^k$ any further increase in v_j^k decreases the slack, maintaining $u_{ij} = 0$, until $v_j^k = v_i^k + s_{ij}^k$ where the slack is zero. From this point on, u_{ij} increases linearly as a function of v_j^k. Figure 4-5b illustrates case (b).

In Figure 4-5a arc $(i,j)^k$ is active when $v_j^k \in [v_i^k, \infty)$ and arc $(j,i)^k$ is active when $v_j^k \in (-\infty, v_i^k]$. In Figure 4-5b arc $(i,j)^k$ is active when $v_j^k \in [v_i^k + s_{ij}^k, \infty)$ and arc $(j,i)^k$ is active when $v_j^k \in (-\infty, v_i^k - s_{ij}^k]$. We further categorize an active arc into two types. When arc $(i,j)^k$ is *active*, $s_{ij} = 0$, $v_j^k \geq v_i^k$, and $u_{ij} \geq 0$. We will say that an active arc is *strongly active* when the dual solution $\mathbf{v}$ satisfies last two conditions as strict inequalities and is *weakly active* otherwise.

The following display summarizes the different states of an arc in the directed network associated with commodity k.

$$
\begin{aligned}
(i,j)^k \text{ active} &\Longrightarrow s_{ij} = 0, v_j^k \geq v_i^k, u_{ij} \geq 0; \\
(i,j)^k \text{ weakly active} &\Longrightarrow s_{ij} = 0; v_j^k \geq v_i^k, u_{ij} = 0, \text{ or} \\
&\qquad\quad s_{ij} = 0, v_j^k = v_i^k, u_{ij} > 0; \qquad (4.7) \\
(i,j)^k \text{ strongly active} &\Longrightarrow s_{ij} = 0; v_j^k > v_i^k, u_{ij} > 0; \\
(i,j)^k \text{ inactive} &\Longrightarrow s_{ij} > 0 \text{ or } v_j^k < v_i^k.
\end{aligned}
$$

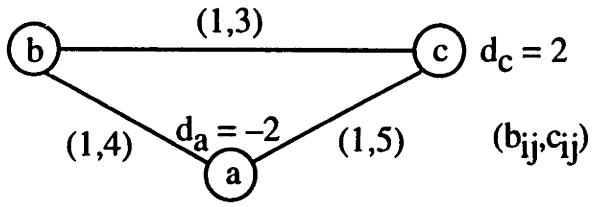
Note that when arc $(i,j)^k$ is strongly active, $v_j^k - v_i^k$ can decrease by some small amount $\delta > 0$ and the arc remains active. When arc $(i,j)^k$ is weakly active, it becomes inactive for any decrease in $v_j^k - v_i^k$. In Figure 4-5a arc $(i,j)^k$ and arc $(j,i)^k$ are both weakly active when $v_j^k = v_i^k$ (notice the condition $s_{ij} = 0$, $v_j^k = v_i^k$, $u_{ij} > 0$ is symmetric). In Figure 4-5b arc $(i,j)^k$ is weakly active when $v_j^k = v_i^k + s_{ij}^k$, and arc $(j,i)^k$ is weakly active when $v_j^k = v_i^k - s_{ij}^k$. Notice that over the region where an arc is active, it is weakly active at the limit points of the region and is strongly active otherwise.

The notion of a directed network associated with each commodity is conceptual. An algorithm need not explicitly construct the directed network associated with each commodity. Rather, to decide whether arc $(i,j)^k$ is active, an algorithm needs only to check that $\{i,j\}$ is an edge in the graph G, $s_{ij} = 0$, $v_j^k \geq v_i^k$, and $u_{ij} \geq 0$. These computations can be done on the fly.

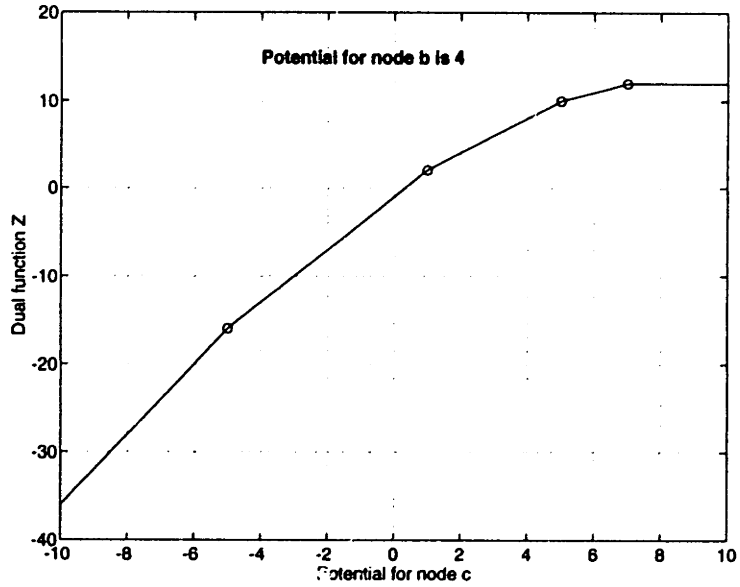
The effect on the dual function $Z(\mathbf{v})$ of increasing the node potential v_j^k for a single node and a single commodity is the sum of a linear function $d_j^k v_j^k$ and the piecewise linear functions $-b_{ij}u_{ij}$ for each edge $\{i,j\}$ that is incident to node j. Therefore, the dual function is piecewise linear. Its breakpoints, as a function of v_j^k, are the breakpoints of the piecewise linear function u_{ij} for edge $\{i,j\}$. In other words the breakpoints of $Z(\mathbf{v})$ as a function of v_j^k occur when some arc directed into node j, $(i,j)^k$, is weakly active, or some arc directed out of node j, $(j,i)^k$, is weakly active.

Figure 4-6 shows the dual function $Z(\mathbf{v})$ for a 2-edge-disjoint path problem on a three node graph. Figure 4-6a shows the capacity and costs of the edges. For convenience, since the problem is a single commodity problem, we drop the superscript k in this discussion. The demand of node c is 2 units and the demand of node a is -2 units. Figure 4-6b shows the dual objective value $Z(\mathbf{v})$ as a function of v_c. For this example, v_a is zero and v_b is 4. The figure shows the breakpoints of the dual function: at $v_c = -5$, $v_c = 1$, $v_c = 5$, and $v_c = 7$, corresponding to points where the arcs (c,a), (c,b), (a,c), and (b,c) are weakly active. Figure 4-6c shows the dual objective value $Z(\mathbf{v})$ as a function of v_b, when both v_a and v_c are zero. The breakpoints of the dual function are at $v_b = -4$, $v_b = -3$, $v_b = 3$, and $v_b = 4$ corresponding to points where the arcs (b,a), (b,c), (c,b), and (a,b) are weakly active.

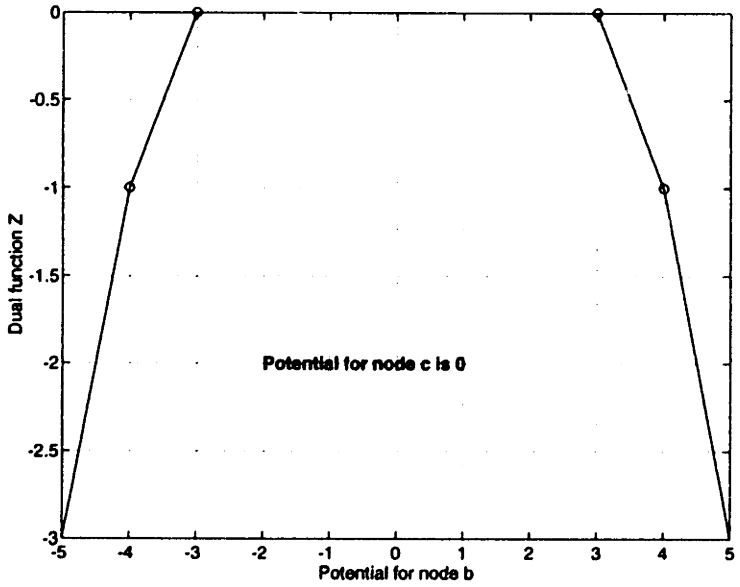
In general, the dual function is piecewise linear as a function of the dual variables. Figure 4-7 shows a plot of the dual objective as a function of the node potentials v_b and v_c for the example of Figure 4-6. In the triangular region, the dual function attains its maximum value.



(a)



(b)



(c)

Figure 4-6: (a) Network for 2-edge-disjoint problem on 3 node graph. (b) Dual objective as a function of v_c with v_b and v_a fixed. (c) Dual objective as a function of v_b with v_c and v_a fixed.

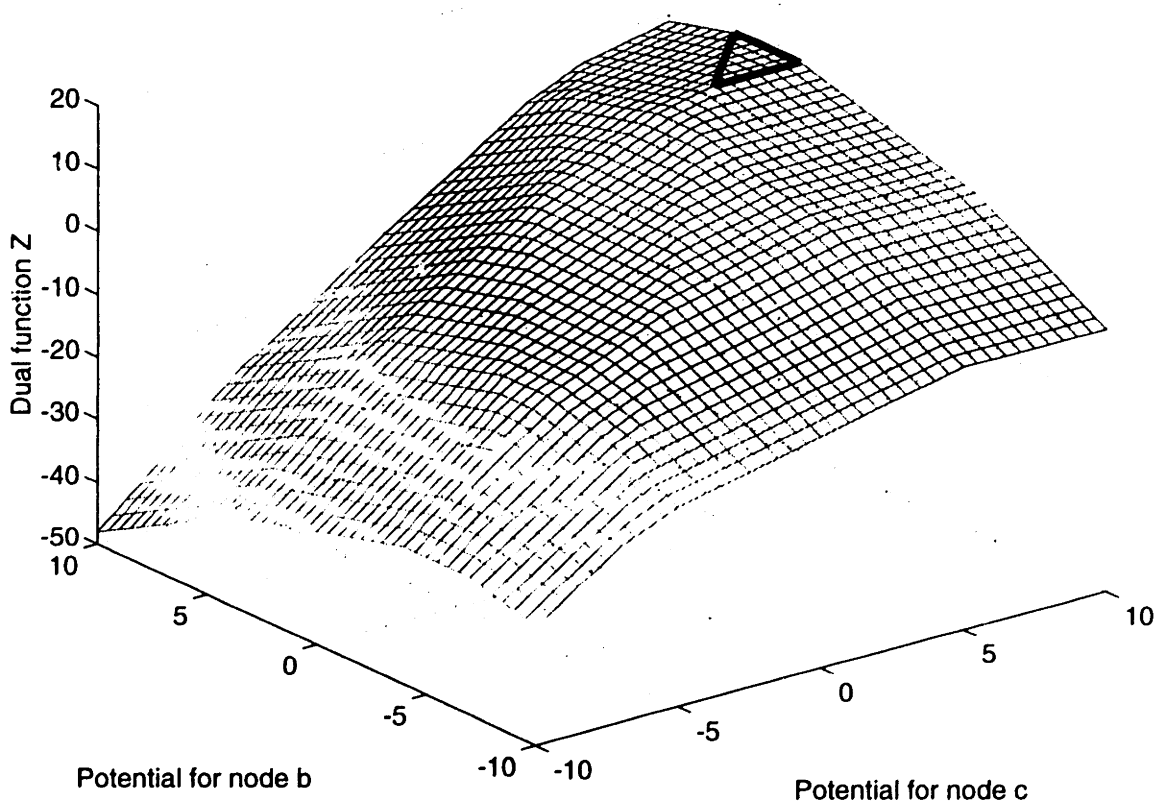


Figure 4-7: Dual function Z as a function of v_b and v_c.

Basic Ascent Directions

To obtain a good lower bound for the NDC problem, we try to find the maximum value of the dual function $Z(\mathbf{v})$. Any dual-ascent algorithm starts with some feasible dual solution $\mathbf{v}$ and iteratively finds ascent directions that improve the dual function. In our quest for a good lower bound, we *restrict our attention to basic directions*. Recall, from Chapter 3, that a set T and a commodity k define a basic direction γ_T^k when

$$\gamma_j^l = \begin{cases} 1 & \text{if } j \in T \text{ and } l = k; \\ 0 & \text{otherwise.} \end{cases}$$

At each step, after finding a basic ascent direction γ_T^k, the algorithm changes the current value of the node potential from the current value $\mathbf{v}$ to $\mathbf{v} + \alpha\gamma_T^k$ for an appropriate step size (scalar) α.

Calculating the directional derivative of the dual cost for any basic direction of the form

γ_T^k, with the aid of Figure 4-5 (and equation (4.5)), we obtain:

$$Z'(\mathbf{v};\gamma_T^k) = \sum_{i\in T} d_i^k + \sum_{\substack{(j,i)^k:\text{strongly active}\\ i\notin T, j\in T}} b_{ji} - \sum_{\substack{(i,j)^k:\text{active}\\ i\notin T, j\in T}} b_{ij}. \tag{4.8}$$

The dual-ascent algorithm is an iterative procedure that uses a *basic ascent direction*, i.e., a basic direction γ_T^k with $Z'(\mathbf{v};\gamma_T^k) > 0$, to perform an ascent step that increases the dual function. This algorithm design naturally leads to the following two issues: How can we find basic ascent directions? At the conclusion of the dual-ascent algorithm, is the auxiliary network feasible for the NDC problem?

The following lemma helps answer these questions.

Lemma 4.2.1 *Suppose that for some commodity's active network, the maximum flow from node $O(k)$ to node $D(k)$ with arc capacities b_{ij} is less than q_k; then, there is a basic ascent direction for the function $Z(\mathbf{v})$.*

Proof: If the maximum flow for commodity k in its active network is less than q_k, then by the max-flow min-cut theorem the network has an $O(k)$–$D(k)$ dicut T with capacity less than q_k. Then, in equation (4.8) the first term is q_k ($q_k = d_{D(k)}^k$). The second term is always nonnegative. Since the third term is just the capacity of the dicut T in commodity k's active network, it is less than q_k. Therefore $Z'(\mathbf{v};\gamma_T^k) > 0$. ■

We might note that the converse of Lemma 4.2.1 is not necessarily true.

Lemma 4.2.1 implies that if there is no basic ascent direction, then every $O(k)$–$D(k)$ dicut has a capacity greater than q_k. Since b_{ij} represents the number of copies of an edge permitted in the model, this conclusion assures us that when there is no basic ascent direction, installing b_{ij} copies of each edge in the auxiliary network will be a feasible design (we might be able to use fewer copies of some edges and still be feasible).

An immediate consequence of this lemma is a method for finding basic ascent directions. Consider the active network for each commodity. Using any max-flow algorithm on commodity k's active network, find a dicut with capacity less than q_k. Any such dicut gives a basic ascent direction. Figure 4-8 illustrates this fact. In the problem shown in Figure 4-8, we wish to construct, at minimum cost, two edge-disjoint paths between node s and node t. All edge capacities are 1. For this single commodity problem, node s is the origin and node t is the destination. Figure 4-8a shows the digraph associated with the commodity, and a dicut in the active network with fewer than two active arcs. This dicut provides a basic ascent direction. Figure 4-8b shows the digraph associated with the commodity at the end of the dual-ascent step (in this example the dual-ascent step ends when arc (e,f) becomes active, i.e., we reach a breakpoint of the dual function).

Another method for choosing a basic ascent direction is the *steepest ascent* approach:

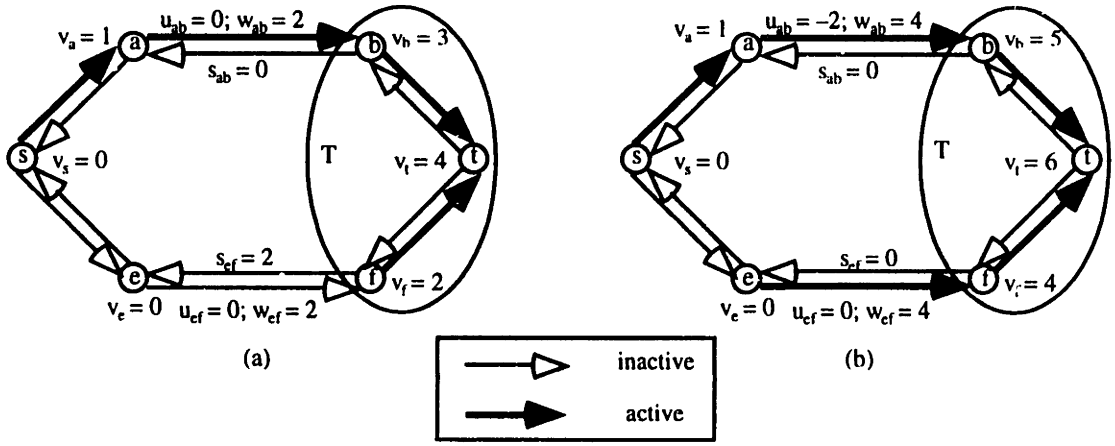


Figure 4-8: If the max-flow in the active network is less than the flow requirement, then there is a basic direction of ascent. Active arcs are in bold. (a) Dual variables before making an ascent along the basic direction. (b) Dual variables after the dual-ascent step.

choose a basic ascent direction that maximizes the rate of ascent. The steepest basic ascent direction appears to be an attractive choice for a search direction because it gives the greatest increase in the dual objective function per unit change in the dual variable.

Identifying the Steepest Basic Ascent Direction

To find the steepest basic ascent direction, over all commodities, we compare the steepest basic ascent direction for each commodity and select the ascent direction with the highest ascent rate.

For a given commodity k, we need to identify the set T with the greatest value of

$$\sum_{i \in T} d_i^k + \sum_{\substack{(j,i)^k: \text{strongly active} \\ i \notin T, j \in T}} b_{ji} - \sum_{\substack{(i,j)^k: \text{active} \\ i \notin T, j \in T}} b_{ij}. \tag{4.9}$$

To solve this problem, we transform it to a related problem called the *max-floor problem*. In the max-floor problem, we are given nonnegative lower and upper bounds, l_{ij} and u_{ij}, on the flow on arc (i,j). The floor of an s–t dicut[6] $\delta^-(T)$ is defined as $\sum_{(i,j) \in \delta^-(T)} l_{ij} - \sum_{(i,j) \in \delta^+(T)} u_{ij}$. Network flow theory tells us that if the problem is feasible, i.e., it is possible to send flow from node s to node t that satisfies flow balance at intermediate nodes and satisfies the lower and upper bound constraints on the flow on an arc, the max-floor of an s–t dicut is equal to the minimum flow from node s to node t (see Exercise 6.18 in Ahuja, Magnanti and Orlin [AMO93]). We can find the max-floor (of an s–t dicut) by applying

[6]Although our notation does not show this, in the max-floor problem, the dicut contains the arcs in $\delta^+(T)$ in addition to those in $\delta^-(T)$.

any max-flow algorithm twice: first to find a feasible flow from node s to node t, and then to decrease this feasible flow to its minimum possible value (see [AMO93]).

We will find the steepest basic ascent direction by solving (at most) three max-floor problems. Consider commodity k's active network. Set the lower bound l_{ij} of a weakly active arc $(i,j)^k$ to 0 and its upper bound u_{ij} to b_{ij}. Also, set both the lower and upper bound, l_{ij} and u_{ij}, of a strongly active arc $(i,j)^k$ to b_{ij}. Let $U = \sum_{(i,j)^k:\text{active}} b_{ij}$. Now solve the following problems:

Problem 1: Add two nodes s' and t' to the active network. For each node i, other than s' and t', add an arc from s' to i with lower bound l_{ij} equal to 0 and upper bound u_{ij} equal to U, and add an arc from i to t' with lower bound l_{ij} equal to 0 and upper bound u_{ij} equal to U. Now find the max-floor of an s'–t' dicut.

Problem 2: For each node $i \neq O(k)$ or $D(k)$, add arc $(D(k), i)$ with $l_{D(k)i} = 0$ and $u_{D(k)i} = U$, and arc $(i, O(k))$ with $l_{iO(k)} = 0$ and $u_{iO(k)} = U$. Find the max-floor of a $D(k)$–$O(k)$ dicut.

Problem 3: For each node $i \neq O(k)$ or $D(k)$, add arc $(O(k), i)$ with $l_{O(k)i} = 0$ and $u_{O(k)i} = U$, and arc $(i, D(k))$ with $l_{iD(k)} = 0$ and $u_{iD(k)} = U$. Find the max-floor of a $O(k)$–$D(k)$ dicut.

Observe that in each of these three problems, the arcs added to the problem ensure the problem is feasible[7] without affecting the value of the floor of a dicut. Also note that adding additional arcs might create parallel arcs in the network. However, this does not create any problem since all the max-flow algorithms apply to networks that contain parallel arcs (the algorithms replace parallel arcs between a pair of nodes by a single arc whose lower bound is the sum of the lower bounds of the parallel arcs and whose upper bound is the sum of the upper bounds of the parallel arcs).

Let $\nu^i_{\max}$ denote the max-floor of Problem i, and let S^i be the set of nodes on the destination side of the dicut (i.e., $t' \in S^1, O(k) \in S^2$, and $D(k) \in S^3$). Let

$$i^* = \text{argmax}_{i=1,2,3} \left\{ \nu^i_{\max} + \sum_{i \in N \setminus S^i} d^k_i \right\}.$$

[7]This result follows from Hoffman's circulation feasibility condition [Hof60] which states that a circulation problem is feasible if and only if for every set S of nodes

$$\sum_{(i,j) \in \delta^-(S)} l_{ij} \leq \sum_{(i,j) \in \delta^+(S)} u_{ij}.$$

To check feasibility, we could convert the problem to a feasible circulation problem by adding an arc from the destination node to the origin node with lower bound 0 and upper bound $u_{ij} = \infty$.

Then the steepest basic ascent direction for commodity k is given by the set $T^* = \{N\backslash S^{i^*}\}$. To see that this result is valid observe that only nodes $O(k)$ and $D(k)$ have nonzero demands. Therefore, the first term in equation (4.9) is nonzero only when one of the nodes $O(k)$ and $D(k)$ is in T. Problem 2 ensures that $D(k)$ is in T and $O(k)$ is not in T (note that $T = N\backslash S^2$). Problem 3 ensures that $O(k)$ is in T and $D(k)$ is not in T (note that $T = N\backslash S^3$). Problem 1 does not restrict either $O(k)$ or $D(k)$, and it finds the set ($T = N\backslash S^1$) that maximizes the latter two terms of equation (4.9). By comparing these three solutions, we obtain T^* that defines the steepest basic ascent direction $\gamma^k_{T^*}$ for commodity k.

Note that if we solve Problem 1 first, then in some cases we need not solve Problems 2 and 3. If in the solution to Problem 1, i.e., the s'–t' dicut, $O(k) \in S$ and $D(k) \notin S$, then the set $T = N\backslash S^1$ defines the steepest basic ascent direction for commodity k. Why? We have found the set T that maximizes the latter two terms of equation (4.9). If T contains $D(k)$ but does not contain $O(k)$, it maximizes the first term of equation (4.9). Therefore, the set T defines the steepest basic ascent direction for commodity k. Also note that if in the solution to Problem 1, $O(k) \notin S$ and $D(k) \in S$, then the solution to Problem 1 is also the solution to Problem 3, in which case we need not solve Problem 3.

Although we have shown how to find the steepest basic ascent direction, we will not pursue this approach for two reasons. First, in this approach we potentially solve six max-flow problems (two for each max-floor problem) for each commodity in order to identify the steepest basic ascent direction. This is an excessive computational burden. Second, following the steepest basic ascent direction is not guaranteed to provide the best lower bound. For example, following the steepest basic ascent direction does not solve the q-edge-disjoint path problem.[8] In the next section we describe a different approach for finding a basic ascent direction, which does not always find the steepest basic ascent direction, but nevertheless solves the q-edge-disjoint path problem.

4.2.2 An Alternate Method for Identifying Basic Directions

The dual-ascent algorithm we develop follows a somewhat different approach for finding basic ascent directions from that suggested by Lemma 4.2.1. It maintains a dual feasible solution $(\mathbf{v}, \mathbf{w}, \mathbf{u})$ and a primal solution (not necessarily feasible) $(\mathbf{x}, \mathbf{f})$ that satisfy the primal complementary slackness conditions. Every infeasible primal solution guides the search for basic directions, providing a basic ascent direction.

At any intermediate step in the algorithm, let $\bar{d}^k_i$ represent the residual demand of

[8] To be more specific, arbitrarily choosing a steepest basic ascent direction does not solve the q-edge-disjoint path problem. A particular choice of the steepest basic ascent direction solves the q-edge-disjoint path problem. The alternate method described in the next section does not always find the steepest basic ascent direction. However in one case, the q-edge disjoint path problem, it chooses the "right" steepest basic ascent directions and solves the problem.

commodity k at node i. That is,

$$\bar{d}_j^k = d_j^k - (\sum_{i \in N} f_{ij}^k - \sum_{l \in N} f_{jl}^k).$$

The residual demand for any set T and commodity k is

$$\sum_{j \in T} \bar{d}_j^k = \sum_{j \in T} (d_j^k - (\sum_{i \in N} f_{ij}^k - \sum_{l \in N} f_{jl}^k)).$$

Using equation (4.8) to substitute for $\sum_{j \in T} d_j^k$ in this equation, we obtain

$$\sum_{j \in T} \bar{d}_j^k = Z'(\mathbf{v}, \gamma_T^k) - \sum_{\substack{(j,i)^k \text{strongly active} \\ i \notin T, j \in T}} (b_{ji} - f_{ji}^k) + \sum_{\substack{(j,i)^k \text{weakly active} \\ i \notin T, j \in T}} f_{ji}^k + \sum_{\substack{(j,i)^k \text{inactive} \\ i \notin T, j \in T}} f_{ji}^k + \sum_{\substack{(i,j)^k \text{active} \\ i \notin T, j \in T}} (b_{ij} - f_{ij}^k) - \sum_{\substack{(i,j)^k \text{inactive} \\ i \notin T, j \in T}} f_{ij}^k.$$

If the primal-dual pair $(\mathbf{x}, \mathbf{f})$ and $(\mathbf{v}, \mathbf{w}, \mathbf{u})$ satisfy the primal complementary slackness conditions (4.6), we can drop the summations over inactive arcs (they will have no flow). Doing so, and rearranging terms, we obtain

$$Z'(\mathbf{v}, \gamma_T^k) = \sum_{j \in T} \bar{d}_j^k + \sum_{\substack{(j,i)^k \text{strongly active} \\ i \notin T, j \in T}} (b_{ji} - f_{ji}^k) - \sum_{\substack{(j,i)^k \text{weakly active} \\ i \notin T, j \in T}} f_{ji}^k - \sum_{\substack{(i,j)^k \text{active} \\ i \notin T, j \in T}} (b_{ij} - f_{ij}^k). \quad (4.10)$$

This equation shows that any set T of nodes with

1. positive residual demand for some commodity k,
2. no active arc $(i,j)^k$ directed into T with $f_{ij}^k < b_{ij}$,
3. no weakly active arc $(j,i)^k$ directed out of T with $f_{ji}^k > 0$, and
4. no flow of commodity k on any inactive arc $(i,j)^k$ directed into or out of T

provides a basic ascent direction. We refer to any such set T as an *ascent set*.[9] Note that any primal-dual pair $(\mathbf{x}, \mathbf{f})$ and $(\mathbf{v}, \mathbf{w}, \mathbf{u})$ satisfying primal complementary slackness also satisfy condition 4. Figure 4-9 displays the types of arcs that might be directed into and out of an ascent set.

[9]Note that the definition of an ascent set is different from the one we used in Chapter 3.

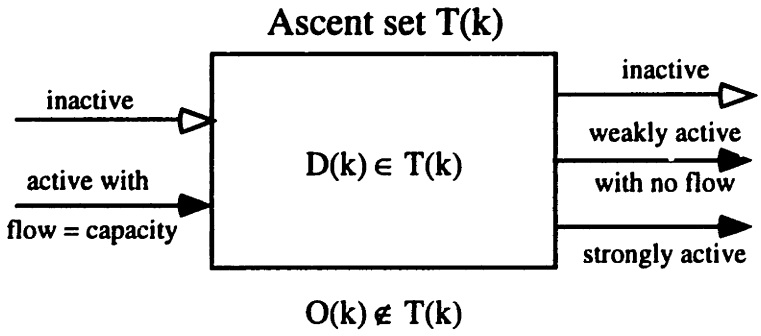


Figure 4-9: In commodity k's network (i) arcs directed into ascent set $T(k)$ must either be inactive or active with flow for commodity k equal to capacity, (ii) arcs directed out of ascent set $T(k)$ must be either inactive, weakly active with no flow for commodity k, or strongly active. Since the only node with positive residual demand is $D(k)$, and $\bar{d}^k_{D(k)} = -\bar{d}^k_{O(k)}$, $D(k) \in T(k)$ and $O(k) \notin T(k)$.

Lemma 4.2.2 *Suppose* $(\mathbf{x}, \mathbf{f})$ *and* $(\mathbf{v}, \mathbf{w}, \mathbf{u})$ *satisfy the primal complementary slackness conditions, and suppose the residual demand of the set of nodes* T *for commodity* k *is greater than zero. Then either the basic direction* γ^k_T *is an ascent direction, or the network contains nodes* $i \notin T$ *and* $j \in T$ *satisfying the property that either* $(i,j)^k$ *is active with* $f^k_{ij} < b_{ij}$ *or* $(j,i)^k$ *is weakly active with* $f^k_{ji} > 0$.

Proof: Follows from equation (4.10). ■

In our approach, typically, each iteration of the algorithm either improves the primal solution (decreases the residual demand for a commodity k at node $D(k)$) or finds a basic ascent direction—that is, an ascent set—and improves the value of the dual function. There might be several different ways to choose an ascent set. The steepest ascent approach finds the ascent set that maximizes

$$\sum_{j \in T} \bar{d}^k_j + \sum_{\substack{(j,i)^k \text{ strongly active} \\ i \notin T, j \in T}} (b_{ji} - f^k_{ji}). \tag{4.11}$$

It seems to be difficult to find the ascent set with the maximum rate of ascent. The approach we took to find the steepest basic ascent direction does not apply here. We conjecture that finding the ascent set with the maximum rate of ascent is $\mathcal{NP}$-hard.

The dual-ascent algorithm we develop uses a heuristic to find the best rate of ascent. First, it ensures the primal solution $(\mathbf{x}, \mathbf{f})$ satisfies the flow balance constraints at all nodes except the origin and destination of any commodity. Second, it considers the first term in equation (4.11), the residual demand, as a measure of the rate of ascent. Because the primal solution satisfies the flow balance constraints at each node except the origin and destination, it is easy to find an ascent set with maximum residual demand. Consider commodities with

the maximum value of $\bar{d}^k_{D(k)}$. For each commodity, the algorithm performs the following *identify-ascent-set* procedure, which either finds an ascent set or decreases the commodity's residual demand (we refer to $\bar{d}^k_{D(k)}$ as the commodity k's residual demand).

Procedure identify-ascent-set(k):

1. Set $T(k) = \{D(k)\}$.
2. If an active arc $(i,j)^k$ with $f^k_{ij} < b_{ij}$ is directed into the set $T(k)$, add node i to $T(k)$.
3. If a weakly active arc $(j,i)^k$ with $f^k_{ji} > 0$ is directed out of the set $T(k)$, add node i to $T(k)$.
4. Repeat Steps 2 and 3 until either $O(k) \in T(k)$ or no more nodes can be added to $T(k)$.
5. If Step 4 terminates with $O(k) \notin T(k)$, then $T(k)$ is an ascent set. Otherwise, send one unit of flow from $O(k)$ to $D(k)$.

Note that when $O(k) \in T(k)$ in Step 4, it is possible to send a unit of flow from node $O(k)$ to node $D(k)$. Steps 2 and 3 add a node i to the set $T(k)$ if arc $(i,j)^k$ is active and $f^k_{ij} < b_{ij}$, or if arc $(j,i)^k$ is weakly active and $f^k_{ji} > 0$. This choice implies the existence of a path from $O(k)$ to $D(k)$ in commodity k's active network consisting of "forward" arcs, that is, active arcs that have the same direction as the path, and "reverse" arcs, that is, weakly active arcs that have the opposite direction than the path. Moreover, on all forward arcs, $f^k_{ij} < b_{ij}$, and on all reverse arcs, $f^k_{ji} > 0$. Therefore, it is possible to send 1 unit of flow along this path by increasing the flow of all forward arcs, and decreasing the flow of all reverse arcs by 1.

If the procedure *identify-ascent-set* does not find an ascent set, it has reduced the residual demand for commodity k by 1 unit. We repeat the procedure for commodities with the highest amount of residual demand until it either finds an ascent set or a primal feasible solution.[10]

So far, we have described how to find an ascent set T with the maximum residual demand. The dual-ascent algorithm now improves the dual function by moving along the basic ascent direction γ^k_T associated with the ascent set T. Since the algorithm performs ascent steps using ascent sets, it can increase the dual function along the direction γ^k_T until T no longer is an ascent set for commodity k. This occurs when an arc $(i,j)^k$ directed into

[10]Note that we do not need to explicitly keep track of $\mathbf{x}$. Given the flow $\mathbf{f}$, the equation $x_{ij} = \max_{k \in K}(f^k_{ij}, f^k_{ji})$ permits us to determine the value of $\mathbf{x}$.

T or an arc $(j,i)^k$ directed out of T satisfies the conditions needed to add node i to T (i.e., conditions 2 and 3 in *identify-ascent-set*).

We need to consider one important remaining detail. Recall that the equivalent definition of primal complementary slackness (on page 97) implies that inactive arcs for every commodity must have zero flow for that commodity. When we change the node potentials for commodity k in a dual-ascent step, using an ascent set for commodity k, we maintain the primal complementary slackness condition for that commodity. (It is easy to see that this is true by considering Figure 4-9. Since inactive arcs directed into and out of the ascent set T carry no flow of commodity k, a node potential change does not violate a primal complementary slackness condition for these arcs. Because active arcs directed into the ascent set T remain active, an ascent step will not violate any primal complementary slackness condition for these arcs. Active arcs directed out of the ascent set T can become inactive at the end of an ascent step. If this happens, then the arc carries no flow of commodity k because otherwise it would have no longer been an ascent set when the arc became weakly active.) However, the new solution pair $(\mathbf{x},\mathbf{f})$, $(\mathbf{v},\mathbf{w},\mathbf{u})$ might violate this condition for some other commodity h. How can this happen? Recall that an active arc $(i,j)^k$ satisfies two conditions (i) $s_{ij}=0$ and (ii) $v_j^k \geq v_i^k$. In an ascent step along the basic ascent direction γ_T^k, when we increase v_j^k for $j \in T$ for commodity k, we do not alter the node potentials for any other commodity $h \neq k$. So the only way an active arc for commodity h becomes inactive in an ascent step for commodity k, is when the slack s_{ij} becomes positive. This outcome is possible (see Figure 4-5) only if an arc $(j,i)^k$, directed out of T, were active. As a result, both arcs $(j,i)^h$ and $(i,j)^h$ might become inactive (if $s_{ij}>0$ both are inactive), violating primal complementary slackness if either $f_{ji}^h > 0$ or $f_{ij}^h > 0$.

In order to rectify this problem, our dual-ascent algorithm updates the flow so that it (i) satisfies primal complementary slackness, and (ii) satisfies the flow balance constraints at all intermediate nodes. Suppose commodity h flows on some inactive arcs. Then we can change the flow of commodity h to satisfy conditions (i) and (ii) as follows. Solve a maximum flow problem from node s to node t on commodity h's active network, setting the capacity of an arc $(i,j)^h$ to f_{ij}^h. Because we solve the maximum flow problem on commodity h's active network, its solution satisfies condition (i) and condition (ii).

This maximum flow computation requires considerable work. However, it is not obvious whether there is a better method to update the flow. We might note that if the model does not permit edge replication, i.e., $b_{ij}=1$, then we could use more efficient versions of max-flow algorithms (see Chapter 8 of Ahuja, Magnanti and Orlin [AMO93]) for unit capacity networks. In models that permit edge replication, but have small edge capacities (i.e., b_{ij} is small), or if the maximum flow on any arc is small, we could use a special max-flow algorithm designed specially for problems with small integer capacities on the arcs (see

Fernandez-Baca and Martel [FBM89]).

The flow update necessitates a recalculation of the residual demand for commodities whose flow has been changed. The dual-ascent procedure recalculates the residual demand for each commodity and continues by considering ascent sets with the highest residual demand.

In summary, the dual-ascent algorithm for the NDC problem uses three main ideas:

1. It considers ascent sets to improve the dual function.
2. It finds ascent sets by considering commodities with the highest residual demand.
3. It maintains the primal complementary slackness conditions. If a dual-ascent step causes the primal solution to violate the primal complementary slackness conditions (by making an active arc with flow on it inactive), we update the primal solution and the residual demand.

In the next section we investigate in greater detail the implementation of these ideas. We also show that when the edge costs are integer,[11] the dual-ascent algorithm terminates finitely. Furthermore, at termination it will either prove the problem is infeasible or obtain a feasible primal solution $(\mathbf{x}, \mathbf{f})$.

Before we conclude this section we note that in this presentation each commodity k has a single source and a single destination. Much of our discussion applies as well to problems with multiple sources and/or destinations for each commodity.

4.3 Algorithmic Details

In this section we describe in detail the dual-ascent algorithm. We note that many design decisions will be made in the algorithm (e.g., stepsize rules, order of processing commodities). Other design choices are possible as well. We provide (intuitive) reasons for our particular design decisions, but note that an empirical evaluation is required to decide on the best alternatives.

In the description of the algorithm we will maintain only the flow-node potential pair $(\mathbf{f}, \mathbf{v})$ because $\mathbf{x}, \mathbf{w}, \mathbf{u}, \mathbf{s}$ and $Z(\mathbf{v})$ are defined by the values of $\mathbf{f}$ and $\mathbf{v}$. For clarity, we first recall the equations that define $\mathbf{x}, \mathbf{w}, \mathbf{u}, \mathbf{s}$ and $Z(\mathbf{v})$.

$$x_{ij} = \max_{k \in K}(f_{ij}^k, f_{ji}^k)$$

[11] This assumption is not particularly restrictive. Note that we can always transform rational edge costs to integer edge costs by multiplying them by a suitably large number. Moreover, in practice, we need to convert irrational numbers to rational numbers to represent them on a computer.

$$
\begin{aligned}
w_{ij}^k &= \max(v_j^k - v_i^k, v_i^k - v_j^k) \\
u_{ij} &= \max(0, (\sum_{k \in K} \max(v_j^k - v_i^k, v_i^k - v_j^k)) - c_{ij} \\
s_{ij} &= \max(0, c_{ij} - (\sum_{k \in K} \max(v_j^k - v_i^k, v_i^k - v_j^k))) \\
Z(\mathbf{v}) &= \sum_{k \in K} \sum_{i \in N} d_i^k v_i^k - \sum_{\{i,j\} \in E} b_{ij} \max(0, \sum_{k \in K} (\max(v_j^k - v_i^k, v_i^k - v_j^k)) - c_{ij}).
\end{aligned} \tag{4.12}
$$

Before we begin the detailed discussion of the algorithm, we once again emphasize that the notion of directed networks associated with each commodity is conceptual. By storing the undirected graph G and the dual variables, it is easy to calculate the state of an arc $(i,j)^k$ as needed.

The dual-ascent algorithm consists of two subroutines: *update* and *phase*. The subroutine *update* corrects primal complementary slackness violations. It (i) corrects the flow so that it satisfies the primal complementary slackness conditions, and (ii) computes the maximum residual demand after the flow change. The subroutine *phase* considers commodities with the maximum residual demand. We refer to commodities whose residual demand is equal to the maximum residual demand as *active commodities.* The subroutine *phase* performs ascent steps on the ascent sets of active commodities. It ends when either the value of the maximum residual demand decreases by one, or the flow-node potential pair violates a primal complementary slackness condition.

To find an ascent set, the subroutine *phase* could use the procedure *identify-ascent-set* described in the previous section. A drawback of this procedure is that it does not use information about previous ascent sets for commodity k. Each time the algorithm calls *identify-ascent-set*, it starts from scratch, i.e., with $T(k) = \{D(k)\}$. This could be very time consuming. The dual-ascent algorithm we describe uses a slightly different procedure, one that uses the previous ascent set for commodity k to find a new ascent set for commodity k.

At the start of the subroutine *phase*, it sets $T(k) = \{D(k)\}$ for each active commodity. It then calls (the new procedure) *find-ascent-set* to identify an ascent set for each active commodity, and selects the ascent set with the maximum ascent rate to perform an ascent step. In subsequent iterations the subroutine *phase* calls *find-ascent-set* and provides it with the most recent ascent set $T(k)$ for commodity k.

Like *identify-ascent-set*, the *find-ascent-set* procedure either identifies an ascent set for commodity k or increments the flow for commodity k by one unit. It starts with the set $T(k)$ and adds a node i to the set $T(k)$ only if (i) $f_{ij}^k < b_{ij}$ and $(i,j)^k$ is an active arc directed into $T(k)$, or if (ii) $f_{ji}^k > 0$ and $(j,i)^k$ is a weakly active arc directed out of $T(k)$. This choice ensures that the directed network associated with commodity k has sufficient capacity for every node in $T(k)$ to send a unit of flow to $D(k)$. Moreover, this flow can be

sent along a path that is contained in $T(k)$ (i.e., both endpoints of each arc on the path are in $T(k)$). If $O(k) \in T(k)$, then *find-ascent-set* calls the procedure *increment-flow* to increase the flow from $O(k)$ to $D(k)$ by one unit.

The procedure *increment-flow* first identifies a path P from $O(k)$ to $D(k)$ in $T(k)$ that has sufficient capacity to send a unit of flow. Next, it checks to ensure that every arc on the path P is active (i.e., sending one unit of flow on P does not violate the primal complementary slackness conditions). Why is this check necessary? Because we keep track of $T(k)$ across iterations, an arc on P might become inactive when the algorithm performs an ascent step (i.e., changes node potentials) for another commodity h—after the algorithm has added node i to $T(k)$, but before it has added $O(k)$ to $T(k)$. When i is added to $T(k)$ either $(i,j)^k$ is active or $(j,i)^k$ is weakly active. In the ascent step for commodity h, the arcs $(i,j)^k$ and $(j,i)^k$ might become inactive. If the arcs $(i,j)^k$ and $(j,i)^k$ carry no flow, then this change does not result in a primal complementary slackness violation. However, sending a unit of flow on this path now will violate a primal complementary slackness condition. Therefore, along the path P, the procedure *increment-flow* checks that the forward arcs are active and the reverse arcs are weakly active. If the check fails, it resets $T(k)$ to $\{D(k)\}$ and continues to search for an ascent set. Otherwise, it sends a unit of flow along P.

Actually the check does more than enforce the primal complementary slackness conditions. For example, if a reverse arc on the path P is strongly active, decreasing its flow will not violate a complementary slackness condition. However, the check will not send flow on the path, but will reset $T(k)$ to $\{D(k)\}$ and continue its search for an ascent set. This stronger condition has the following rationale. Recall that if there is no path from $O(k)$ to $D(k)$, in commodity k's active network, consisting of forward arcs with $f_{ij}^k < b_{ij}$ and weakly active reverse arcs with $f_{ji}^k > 0$, *find-ascent-set* identifies an ascent set. If the reverse arc $(j,i)^k$ on the path P is strongly active, then after resetting $T(k)$, the algorithm might find an ascent set for commodity k because of this reason.

Ascending on an Ascent Set

The dual-ascent algorithm determines the amount of ascent for an ascent set $T(k)$ by examining each edge in the cut $\delta(T(k))$. There are five possible cases (see Figures 4-9 and 4-5).

1. $(j,i)^k$ is strongly active: We permit the algorithm to increase v_j^k until $(j,i)^k$ becomes weakly active. There are two reasons for doing so. First, the algorithm needs to check that $T(k)$ is still an ascent set, i.e., whether $f_{ji}^k = 0$. Second, assuming $T(k)$ is still an ascent set, the dual variable change has just decreased the rate of ascent for the set $T(k)$ because $(j,i)^k$ is no longer strongly active (see equation (4.10)). By terminating the ascent step at this point, the algorithm can compare the ascent sets associated

with the commodities and select the ascent set with the highest ascent rate. Recall from Figure 4-5 that in case (a) $(j,i)^k$ becomes weakly active when $v_j^k = v_i^k$, and in case (b) this occurs when u_{ij} decreases to zero. Therefore, the increase is determined by $\min(v_i^k - v_j^k, u_{ij})$.

2. $(j,i)^k$ is weakly active: If $(j,i)^k$ is weakly active, then $f_{ji}^k = 0$; otherwise, $T(k)$ is not an ascent set. Therefore, increasing v_j^k does not violate the primal complementary slackness condition for commodity k. We determine the maximum ascent by the value of v_j^k that maximizes the slack on edge $\{i,j\}$. That is, when $v_j^k = v_i^k$. Actually, we could increase v_j^k beyond $v_j^k = v_i^k$. However, limiting the ascent to this choice might create greater ascent for the dual function. Why? Since this change created an inactive arc from a (weakly) active arc, and made the slack $s_{ij} > 0$, arcs $(i,j)^h$ and $(j,i)^h$ for any other commodity h are also inactive. Suppose another commodity with the same value of residual demand sends flow on this arc. Then because the arc is inactive it violates primal complementary slackness. Therefore, the algorithm can send no flow for commodity h along a path that uses this arc. In order to satisfy primal complementary slackness, the algorithm decreases the flow of commodity h from $O(h)$ to $D(h)$ by at least one unit. But then its residual demand increases by at least one unit. Since the dual-ascent algorithm uses the residual demand as a measure of the rate of ascent, terminating the ascent step when $v_j^k = v_i^k$ is appropriate.

3. $(j,i)^k$ is inactive and $v_i^k > v_j^k$:[12] As v_j^k increases, the slack increases until $v_j^k = v_i^k$. Then the slack decreases until the slack becomes zero and arc $(i,j)^k$ becomes active. From Figure 4-5b, arc $(i,j)^k$ becomes active when $v_j^k = v_i^k + s_{ij}^k$. The increase in v_j^k before the slack becomes zero is $v_i^k + s_{ij}^k - v_j^k$. Since $s_{ij} = s_{ij}^k - \max(v_i^k - v_j^k, v_j^k - v_i^k)$, this choice implies that $s_{ij}^k = s_{ij} + \max(v_i^k - v_j^k, v_j^k - v_i^k)$. Therefore, the increase in v_j^k until arc $(i,j)^k$ becomes active is $s_{ij} + 2(v_i^k - v_j^k)$.

4. $(i,j)^k$ is inactive and $v_j^k \geq v_i^k$: As v_j^k increases, arc $(i,j)^k$ eventually becomes weakly active and thus $T(k)$ will no longer be an ascent set. The maximum amount of increase is determined by the value of v_j^k at which $(i,j)^k$ becomes weakly active. Therefore, the maximum increase is the slack.

5. $(i,j)^k$ is (weakly or strongly) active. We need not consider these edges since they will remain active.

[12] Since $\{i,j\}$ is really an edge and the directed network a conceptual construction we do not consider overlapping cases (e.g., $(j,i)^k$ is inactive and $(i,j)^k$ is active). The five cases considered do not overlap in Figure 4-5.

The stepsize is determined by the minimum of these quantities over all edges in the cut. If no edge in the cut $\delta(T(k))$ satisfies one of the first four cases, then for all edges $\{i,j\}$ in the cut $\delta(T(k))$ with $i \notin T(k)$ and $j \in T(k)$, $f_{ij}^k = b_{ij}$ and $f_{ji}^k = 0$. Since the residual demand of $T(k)$ is greater than zero,

$$\sum_{j \in T(k)} \bar{d}_j^k = \sum_{j \in T(k)} (d_j^k - (\sum_{i \in N} f_{ij}^k - \sum_{l \in N} f_{jl}^k)) \quad > \quad 0.$$

By rearranging terms we see that

$$d_{D(k)}^k - \sum_{j \in T(k)} (\sum_{i \notin T(k)} f_{ij}^k - \sum_{i \notin T(k)} f_{ji}^k) > 0;$$

or

$$\sum_{\{i,j\} \in \delta(T(k))} b_{ij} < q_k.$$

Therefore, $\delta(T(k))$ is an $O(k)$-$D(k)$ cut whose capacity is less than commodity k's requirement, or the problem is infeasible. Another interpretation is as follows. Since the domain of the minimization is empty, the stepsize is infinite, and the amount of ascent is unbounded. From linear programming duality, if the dual is feasible and unbounded, the primal is infeasible.

We now describe another condition that also indicates that the problem is infeasible. Although the above condition is sufficient to prove a problem is infeasible, it is not obvious whether the dual-ascent algorithm will always find a cut that proves infeasibility. We claim that if $Z(\mathbf{v}) > \sum_{\{i,j\} \in E} c_{ij} b_{ij}$, then the problem is infeasible. This is true because were the problem feasible, then a solution that contains b_{ij} copies of each edge would be feasible. Since a feasible dual solution is always a lower bound on any feasible primal solution this result follows.

Therefore, if the dual-ascent algorithm terminates (we show it terminates in Section 4.3.2), it does one of the following:

1. Finds a feasible primal solution and a feasible dual solution that satisfies primal complementary slackness.

2. Shows the problem is infeasible by exhibiting an $O(k)$-$D(k)$ cut whose capacity is less than commodity k's requirement.

3. Shows the problem is infeasible because $Z(\mathbf{v}) > \sum_{\{i,j\} \in E} c_{ij} b_{ij}$.

4.3.1 Formal Description of the Dual-Ascent Algorithm

The dual-ascent algorithm can be described as follows:

Dual-ascent algorithm for the NDC problem

1. Initialization. Set $\mathbf{v} = \mathbf{0}$, $\mathbf{f} = \mathbf{0}$, and counter = 1.
2. While counter $\leq r_{\max}$ do
   - Perform *phase*(counter).
   - If *phase* terminated because of a complementary slackness violation, perform *update*(flow,counter). Otherwise, set counter = counter + 1 and continue.

The procedure *phase* can be described as follows:

Phase (counter)

1. Consider all commodities k, called *active*, whose residual demand $\bar{d}^k_{D(k)} = r_{\max} - counter + 1$. For all commodities, set $T(k) = \{D(k)\}$. Mark all nodes as unscanned: **scan**$[k][j] = 0$ for all nodes $j \in N$. All **next** labels are undefined.
2. For every active commodity (see definition following *phase*) k, *find-ascent-set*$(T(k))$. If no commodity is active, return.
3. Among the active commodities, select the commodity with the highest ascent rate: i.e., the ascent set $T(k)$ with the highest rate of ascent, and perform *ascent-step*$(T(k))$. If no commodity is active, return.
4. If as a result of *ascent-step*, a weakly active arc with flow becomes inactive, return. Otherwise, go to Step 2.

The subroutine *phase* (and the procedure *find-ascent-set*) use an array **scan** to keep track of the unscanned arcs into the set $T(k)$. As long as $O(k) \notin T(k)$, the procedure *phase* tries to find an ascent set for commodity k. The **next** array is used to identify the path from $O(k)$ to $D(k)$.

The value of the dual function, i.e., the lower bound provided by the dual-ascent algorithm, is sensitive to the order in which the algorithm considers the commodities. For example, Sastry [Sas87] shows that dual-ascent can provide a lower bound that is arbitrarily bad for the Steiner branching problem. A good heuristic, proposed by Balakrishnan, Magnanti and Wong [BMW89] for the uncapacitated fixed charge network design problem, is to consider commodities in a round robin fashion. That is, order the commodities in some fashion and then cycle through this list performing a single ascent step for each commodity. We could implement this rule in the dual-ascent algorithm in a couple of ways. One possibility is to use the round robin rule in the case of ties in Step 3.

Another possibility is to replace Steps 2 and 3 by a single ascent step for a single commodity. That is, choose an active commodity, find an ascent set for the commodity (or decrease its residual demand), and perform an ascent step on this ascent set. The round robin rule could then be used to select the active commodity.

The subroutine *update* can be described as follows:

Update(flow,counter)

1. For each commodity $k \in K$ solve the maximum flow problem between $O(k)$ and $D(k)$ on the commodity's active network with arc capacities f_{ij}^k. Let maxflow(k) denote the value of the maximum flow between $O(k)$ and $D(k)$. Replace the flow values f_{ij}^k for commodity k by the solution of the maximum flow problem for commodity k.

2. Set $t = \max_{k \in K}(d_{D(k)}^k - \text{maxflow}(k))$.

3. Set counter $= r_{\max} - t + 1$.

The procedure *find-ascent-set* can be described as follows:

Find-ascent-set($T(k)$)

1. If all nodes in $T(k)$ are scanned, $T(k)$ is an ascent set; terminate *find-ascent-set.* Otherwise, select an unscanned node $j \in T(k)$ and go to Step 2.

2. Add any node $i \notin T(k)$ to $T(k)$ if either $(j,i)^k$ is weakly active with $f_{ji}^k > 0$, or $(i,j)^k$ is active with $f_{ij}^k < b_{ij}$. For each node i added to $T(k)$, set **next**$[i][k] = j$ if $(i,j)^k$ is active with $f_{ij}^k < b_{ij}$, and **next**$[i][k] = -j$ if $(j,i)^k$ is weakly active with $f_{ji}^k > 0$. Mark j as scanned: **scan**$[j][k] = 1$. If $T(k)$ does not contain $O(k)$, go to Step 1.

3. $T(k)$ contains $O(k)$. Perform the procedure *increment-flow*(k). If *increment-flow(k)* was unsuccessful (i.e., it did not increase the flow from $O(k)$ to $D(k)$), go to Step 1. Otherwise, terminate *find-ascent-set.*

The procedure *increment-flow* can be described as follows:

Increment-flow(k)

1. The path from $O(k)$ to $D(k)$ is identified by starting from $O(k)$ and following the **next** labels: the node following node i on the path is the absolute value of **next**$[i][k]$. *Check to see if sending flow on the path satisfies primal complementary slackness conditions.* If for any node i on the path **next**$[i][k] = j$ and $(i,j)^k$ is inactive, or **next**$[i][k] = -j$ and $(j,i)^k$ is not weakly active; then (i) set $T(k) = \{D(k)\}$, (ii) mark

all nodes as unscanned: $\mathbf{scan}[k][j] = 0$ for all nodes $j \in N$, and (iii) set all **next** labels for commodity k, i.e. $\mathbf{next}[i][k]$, as undefined; and terminate *increment-flow(k)* with a flag to indicate that a flow increase was unsuccessful.

2. *(flow increase)* Send 1 unit of flow on the path from $O(k)$ to $D(k)$ (doing so will not violate primal complementary slackness). For every node i is on the path with $\mathbf{next}[i][k] = j$, set $f_{ij}^k = f_{ij}^k + 1$. For every node i on the path with $\mathbf{next}[i][k] = -j$ set $f_{ji}^k = f_{ji}^k - 1$. Terminate *increment-flow(k)* with a flag to indicate that a flow increase was successful.

Formally, the procedure *ascent-step*$(T(k))$ can be described as follows:

Ascent-step$(T(k))$

1. The stepsize is given by the following equation:

$$\alpha = \min_{\substack{i \notin T(k) \\ j \in T(k)}} \left\{ \begin{array}{ll} \min(u_{ij}, v_i^k - v_j^k) & \text{if } (j,i)^k \text{ is strongly active;} \\ v_i^k - v_j^k & \text{if } (j,i)^k \text{ is weakly active;} \\ s_{ij} + 2(v_i^k - v_j^k) & \text{if } (j,i)^k \text{ is inactive and } v_i^k > v_j^k; \\ s_{ij} & \text{if } (i,j)^k \text{ is inactive and } v_j^k \geq v_i^k. \end{array} \right\} \quad (4.13)$$

   If the domain of the minimization in equation (4.13) is empty, the problem is infeasible. Stop.

2. Set, $\mathbf{v} = \mathbf{v} + \alpha \gamma_{T(k)}^k$. If $Z(\mathbf{v}) > \sum_{\{i,j\} \in E} c_{ij} b_{ij}$ the problem is infeasible. Stop.

3. Add to $T(k)$ all nodes i for which (a) the minimum in equation (4.13) is achieved, and (b) either $(i,j)^k$ is active, or $(j,i)^k$ is weakly active with $f_{ji}^k > 0$. In the first case of part (b), set $\mathbf{next}[i][k] = j$ and in the second case set $\mathbf{next}[i][k] = -j$. If the node $O(k)$ is added to $T(k)$, perform *increment-flow*(k).

Example

To conclude this section, consider the example in Figure 4-10 that illustrates the dual-ascent algorithm for the 3-edge-disjoint path problem (the paths are between s and t). Figure 4-10a shows the undirected network, edge costs, and the associated directed network.

Initially, all dual variables are zero, $T(k) = \{t\}$, and counter = 1. The dual-ascent iteration finds that $T(k)$ is an ascent set and increases the node potential v_t^k. The stepsize,

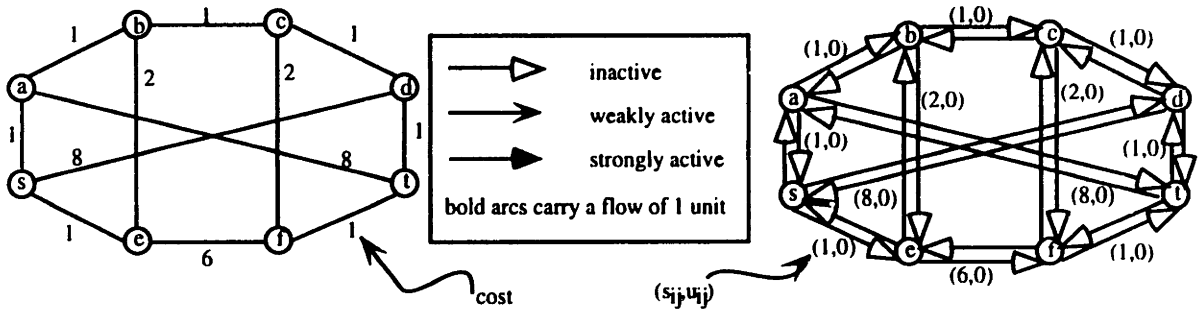
determined by the minimum slack of the incoming arcs, is 1. The algorithm adds nodes d and f to $T(k)$ making $T(k) = \{f, d, t\}$. **next**$[f][k]$ and **next**$[d][k]$ are set equal to t. Proceeding in this fashion, the algorithm successively encounters the ascent sets $T(k) = \{c, f, d, t\}$ (**next**$[c][k] = d$), $T(k) = \{b, c, f, d, t\}$ (**next**$[b][k] = c$), $T(k) = \{a, b, c, f, d, t\}$ (**next**$[a][k] = b$), and $T(k) = \{s, e, a, b, c, f, d, t\}$ (**next**$[e][k] = b$, **next**$[s][k] = a$). Since node s belongs to $T(k)$, the algorithm performs a flow increase step and sends 1 unit of flow on the path *s-a-b-c-d-t* (obtained by the **next** labels). Figure 4-10b shows these iterations. (Note that for a single commodity problem, procedure *increment-flow* need not check for complementary slackness violations). At the end of the flow increase step, the residual demand is 2 units and the dual objective is 15 units. The algorithm resets $T(k) = \{t\}$ and the **next** and **scan** labels. Figure 4-10c shows the dual-ascent iterations performed until the next flow increase step. The dual-ascent iterations find the path *s-e-f-t* and send a unit of flow on the path, in the flow increase step. At the end of the flow increase step, the dual objective is 21 units and the residual demand is 1 unit. Again, reset $T(k) = \{t\}$ and the **next** and **scan** labels. Figure 4-10d shows the dual-ascent iterations until the next flow increase step. The dual-ascent iterations improve the dual objective to 25 units and send one unit of flow on the path *s-d-c-f-e-b-a-t*. Since (c, d), (e, f) and (a, b) carried a unit of flow, the flow increase step decreases their flow to zero. At the end of the flow increase step, the residual demand is zero and the algorithm terminates. The solution obtained by the dual-ascent algorithm has the edges (s, a), (a, t), (s, d), (d, t), (s, e), (e, b), (b, c), (c, f) and (f, t). The solution is feasible (the three paths are *s-a-t*, *s-d-t*, and *s-e-b-c-f-t*) with a cost of 25 units. Since the dual objective is 25 units, the primal solution is optimal. In Section 4.6 we will show that the dual-ascent algorithm optimally solves the q-edge-disjoint path problem.

4.3.2 Finite Termination of the Dual-Ascent Algorithm

We now show that if the edge costs are integer, the algorithm terminates finitely. Recall that in arguments preceding the description of the algorithm, we showed that if the algorithm terminates, it either shows the problem is infeasible or finds a pair of primal and dual solutions $(\mathbf{x}, \mathbf{f})$ and $(\mathbf{v}, \mathbf{w}, \mathbf{u})$ that satisfy the primal complementary slackness conditions.

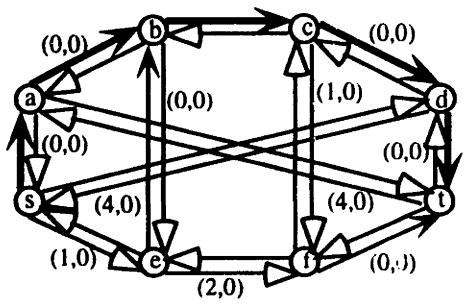
Theorem 4.3.1 *If the edge costs are integer, in a finite number of steps the dual-ascent algorithm for the NDC problem either finds a pair of primal and dual solutions* $(\mathbf{x}, \mathbf{f})$ *and* $(\mathbf{v}, \mathbf{w}, \mathbf{u})$ *that satisfy the primal complementary slackness conditions, or shows the problem is infeasible.*

Proof: To prove that the algorithm terminates finitely, we first show that whenever the algorithm performs the procedure *ascent-step*, it increases the dual objective function by an



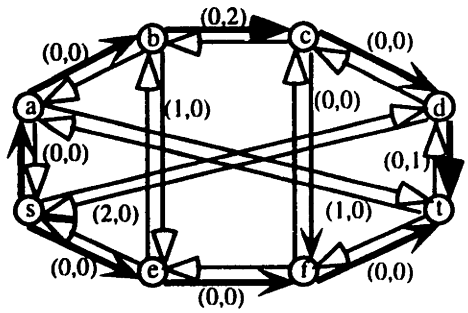
(a) Undirected network with costs and the associated directed network.

Z(v)	v_a	v_b	v_c	v_d	v_e	v_f	v_t	T(k)
0	0	0	0	0	0	0	0	{t}
3	0	0	0	0	0	0	1	{f,d,t}
6	0	0	0	1	0	1	2	{c,f,d,t}
9	0	0	1	2	0	2	3	{b,c,f,d,t}
12	0	1	2	3	0	3	4	{a,b,c,f,d,t}
15	1	2	3	4	0	4	5	{s,e,a,b,c,f,d,t}



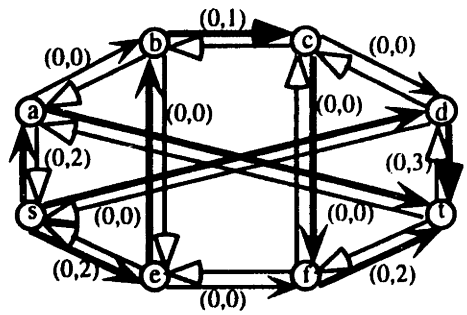
(b) Dual-ascent iterations. Flow is sent on the path *s-a-b-c-d-t*.

Z(v)	v_a	v_b	v_c	v_d	v_e	v_f	v_t	T(k)
15	1	2	3	4	0	4	5	{t}
15	1	2	3	4	0	4	5	{f,t}
17	1	2	3	4	0	5	6	{c,f,t}
17	1	2	3	4	0	5	6	{d,c,f,t}
19	1	2	4	5	0	6	7	{e,d,c,f,t}
21	1	2	5	6	1	7	8	{s,e,d,c,f,t}



(c) Dual-ascent iterations. Flow is sent on the path *s-e-f-t*.

Z(v)	v_a	v_b	v_c	v_d	v_e	v_f	v_t	T(k)
21	1	2	5	6	1	7	8	{t}
22	1	2	5	6	1	7	9	{a,t}
22	1	2	5	6	1	7	9	{b,a,t}
23	2	3	5	6	1	7	10	{e,b,a,t}
23	2	3	5	6	1	7	10	{f,e,b,a,t}
23	2	3	5	6	1	7	10	{c,f,e,b,a,t}
23	2	3	5	6	1	7	10	{d,c,f,e,b,a,t}
25	4	5	7	8	3	9	12	{s,d,c,f,e,b,a,t}



(d) Dual-ascent iterations. Flow is sent on the path *s-d-c-f-e-b-a-t*.

Figure 4-10: Example of dual-ascent algorithm for the 3-edge-disjoint path problem.

integer amount. Next, we show that within a finite number of steps, the dual-ascent algorithm either terminates with a feasible solution or calls the procedure *ascent-step*. Consequently, if the problem is feasible, the dual objective is bounded and so the algorithm finitely terminates. On the other hand, if the problem is infeasible, once $Z(\mathbf{v}) > \sum_{\{i,j\}\in E} c_{ij}b_{ij}$, we have established the problem is infeasible. Thus, if we show that the objective function strictly increases by an integer amount within a finite number of steps, the algorithm finitely terminates when the problem is infeasible.

Initially, all the primal and dual variables are zero and thus integer. Because the edge costs are integer the slacks are also initially integer. First, observe from equation (4.12) that if the edge costs are integer and $\mathbf{v}$ is an integer vector, then $\mathbf{w}$, $\mathbf{u}$ and $\mathbf{s}$ are also integer vectors. Next, notice if $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{s}$ are integer vectors, then equation (4.13) implies that the stepsize, in procedure *ascent-step*, is integer. Now, if $\mathbf{v}$ is an integer vector and we move along a basic ascent direction using an integer stepsize, then the new value of $\mathbf{v}$ will also be integer. Since $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{s}$ are integer at the outset, the stepsize is always integer.

We now argue that the algorithm either terminates or performs the procedure *ascent-step* within a finite number of steps. The algorithm performs the subroutine *update* only after an ascent step. This procedure performs at most $|K|$ max-flow computations. Therefore, assuming we use the successive shortest path max-flow algorithm for the max-flow computation, the procedure requires $\mathcal{O}(|K||N|^2|E|)$ time. The procedure *find-ascent-set* examines the arcs of the directed network associated with commodity k. Because it examines an arc at most once it terminates in $\mathcal{O}(|E|)$ time. Thus, within $\mathcal{O}(|E|)$ time, it either finds an ascent set or adds the origin $O(k)$ to the set $T(k)$. Suppose no ascent steps are performed. Then once the origin $O(k)$ is in the set $T(k)$ the algorithm performs the procedure *increment-flow*. Because the algorithm has performed no ascent steps (i.e., no node potential changes), it will be able to increase the flow without violating complementary slackness. *Increment-flow* examines only arcs on the path from node $O(k)$ to node $D(k)$. Hence it requires $\mathcal{O}(|E|)$ time. Consequently, *find-ascent-set* either identifies an ascent set (and thus the dual-ascent algorithm performs an ascent step) or decreases the residual demand of a commodity by a unit in $\mathcal{O}(|E|)$ time. Since the sum of the demands $(d^k_{D(k)})$ is a constant, the dual-ascent algorithm either terminates with a feasible solution or calls the procedure *ascent-step* within $\mathcal{O}(\sum_{k\in K} d^k_{D(k)}|E| + |K||N|^2|E|)$ time. ■

In this section we described in detail a dual-ascent procedure for the NDC problem with edge-connectivity requirements. We showed that the dual-ascent procedure terminates finitely and at termination either shows the problem is infeasible or provides primal and dual solutions, thereby providing a feasible heuristic solution and a lower bound. In the next two sections we describe how to modify the dual-ascent algorithm to other connectivity problems. First, in Section 4.4 we show how to modify the dual-ascent algorithm

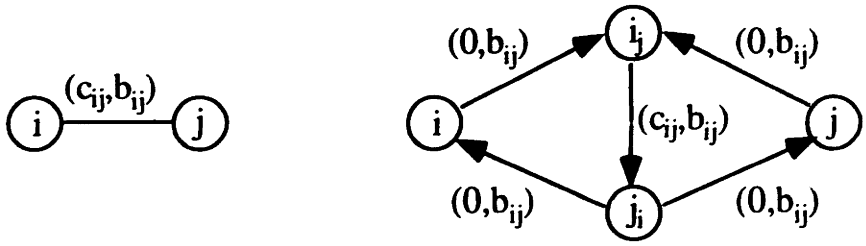


Figure 4-11: Transformation of a (undirected) NDC problem to a DNDC problem.

to the directed network design problem with connectivity requirements (DNDC). Then, in Section 4.5 we consider node connectivity. Finally we describe some extensions to the NDC model and show how to apply the dual-ascent algorithm to them.

4.4 Dual-Ascent for the DNDC Problem

In Chapter 2 we described the directed network design problem with connectivity requirements (DNDC), a problem of some interest. As briefly mentioned in Chapter 2, it is possible to model several real-world telecommunications applications of the DNDC problem. In this section, we show how to modify the dual-ascent procedure for the DNDC problem.

The NDC problem arises in another way: it is possible to transform an NDC problem to a DNDC problem by the following familiar graph transformation procedure for directing undirected multicommodity flow problems (see Exercise 17.21 in Ahuja, Magnanti and Orlin [AMO93]). Transform the undirected graph to a digraph $D = (N, A)$ as follows. For every edge $\{i, j\} \in E$, create two nodes i_j and j_i in the digraph. Replace edge $\{i, j\}$ by the arcs (i, i_j), (j, i_j), (i_j, j_i), (j_i, j), and (j_i, i). The capacity of each of these arcs is equal to the capacity of edge $\{i, j\}$. The cost of arc (i_j, j_i) is equal to the cost of edge $\{i, j\}$, and the cost of the arcs (i, i_j), (j, i_j), (j_i, j) and (j_i, i) are zero. Figure 4-11 shows this construction. If nodes s and t have a connectivity requirement r_{st} in the NDC problem, then nodes s and t have a connectivity requirement of r_{st} in the digraph D. The resulting problem is a DNDC problem. It is easy to see that this directed model is valid for the NDC. The optimal solution for the NDC problem contains edge $\{i, j\}$ only if the optimal solution to the DNDC problem contains arc (i_j, j_i). Consequently, we could transform any NDC problem to a DNDC problem and apply the dual-ascent algorithm of this section to the resulting problem. However, as we shall show shortly, directing the NDC problem does not improve the model. Moreover, since the transformed network contains $5|E|$ arcs, and contains $2|E|$ more nodes than the original network, the dual-ascent algorithm might be computationally more expensive on this transformed network than in the original undirected network.

We can formulate the DNDC problem as follows. Let y_{ij} represent the number of copies

of arc (i,j) in the design and let f_{ij}^k represent the number of units of flow of commodity k on arc (i,j). As before, for each pair $\{s,t\}$ of nodes, with $r_{st} \geq 1$, create a commodity $k \in K$, arbitrarily choosing one of the nodes as the origin and the other node as the destination, with $q_k = r_{st}$. Let $K^{[n]}$ denote the subset of commodities that correspond to node-connectivity requirements.

Directed flow formulation for the DNDC problem:

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}y_{ij} \tag{4.14a}$$

$$\text{subject to:} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k = \begin{cases} -q_k & \text{if } i = O(k); \\ q_k & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } i \in N \text{ and } k \in K \tag{4.14b}$$

$$f_{ij}^k \leq y_{ij} \qquad \text{for all } (i,j) \in A \text{ and } k \in K \tag{4.14c}$$

$$\sum_{l\in N} f_{il}^k \leq 1 \qquad \text{for all } k \in K^{[n]} \text{ \& all } i \neq O(k) \text{ or } D(k) \tag{4.14d}$$

$$f_{ij}^k \geq 0 \qquad \text{for all } (i,j) \in A \text{ and } k \in K \tag{4.14e}$$

$$y_{ij} \leq b_{ij} \qquad \text{for all } (i,j) \in A; \tag{4.14f}$$

$$y_{ij} \geq 0 \qquad \text{and integer, for all } (i,j) \in A. \tag{4.14g}$$

This formulation is similar to the undirected flow formulation for the NDC problem (4.2) except that instead of edge variables, the formulation contains arc variables. Because of the similarity in the models, the subsequent derivation of the dual-ascent algorithm for the DNDC problem is nearly identical to the derivation of the dual-ascent algorithm for the NDC problem. Nevertheless, we provide details of the method to pinpoint the necessary modifications to the dual-ascent algorithm.

Before describing the dual-ascent algorithm for the DNDC problem, we show that formulating the NDC problem as a DNDC problem does not create a better model. In fact, the LP relaxation of the undirected flow formulation for the NDC problem (4.2) and the LP relaxation of the directed flow formulation for the NDC problem (4.14) have the same optimal objective value. It is rather easy to establish this result; for completeness, we state and prove it.

Theorem 4.4.1 *The optimal objective value of the LP relaxation of the undirected flow formulation for the NDC problem (4.2) equals the optimal objective value of the LP relaxation of the directed flow formulation for the NDC problem (4.14).*

Proof: We begin by converting any feasible solution to the LP relaxation of the undirected flow formulation for the NDC problem to a feasible solution to the LP relaxation of the

directed flow formulation for the NDC problem with the same objective value. For each edge $\{i,j\} \in E$, set $y_{ii_j} = y_{ji_j} = y_{j_ii} = y_{j_ij} = y_{ij_i} = x_{ij}$. For each edge $\{i,j\} \in E$ and commodity $k \in K$, set $f^k_{ii_j} = f^k_{j_ij} = f^k_{ij}$, $f^k_{ji_j} = f_{j_ii} = f^k_{ji}$, and $f^k_{ij_i} = f^k_{ij} + f^k_{ji}$ (recall from Section 4.6 we can replace constraint (4.2c) in the undirected flow formulation for the NDC problem by $f^k_{ij} + f^k_{ji} \leq x_{ij}$ [for all $(i,j) \in E$ and $k \in K$]).

Converting any feasible solution to the LP relaxation of the directed flow formulation for the NDC problem to a feasible solution to the LP relaxation of the undirected model for the NDC problem is easy. For each edge $\{i,j\} \in E$ set $x_{ij} = y_{ij_i}$. For each edge $\{i,j\} \in E$ and commodity $k \in K$, set $f^k_{ij} = f^k_{j_ij}$ and $f^k_{ji} = f^k_{j_ii}$. ■

We are now ready to derive the dual-ascent algorithm for the DNDC problem. Again, we first consider only arc-connectivity requirements (so the formulation does not contain constraint (4.14d)). In the next section we consider node-connectivity requirements. As before, let s_{ij} be the slack in the constraint $\sum_{k\in K} w^k_{ij} - u_{ij} \leq c_{ij}$. We say arc (i,j) is *tight* if its slack is zero. The auxiliary network is defined by the tight arcs. Viewing v^k_i, w^k_{ij} and u_{ij} as Lagrange multipliers for constraints (4.14b), (4.14c) and (4.14f), we obtain the following Lagrangian function:

$$\begin{aligned} L(\mathbf{y},\mathbf{f},\mathbf{v},\mathbf{w},\mathbf{u}) \;=\; & \sum_{(i,j)\in A} c_{ij}y_{ij} + \sum_{i\in N}\sum_{k\in K}(d^k_i - \sum_{j\in N} f^k_{ji} + \sum_{l\in N} f^k_{il})v^k_i + \\ & \sum_{(i,j)\in A}\sum_{k\in K}(f^k_{ij} - y_{ij})w^k_{ij} + \sum_{(i,j)\in A}(y_{ij} - b_{ij})u_{ij}. \end{aligned}$$

Rearranging the terms, we obtain

$$\begin{aligned} L(\mathbf{y},\mathbf{f},\mathbf{v},\mathbf{w},\mathbf{u}) \;=\; & \sum_{i\in N}\sum_{k\in K} d^k_i v^k_i - \sum_{(i,j)\in A} b_{ij}u_{ij} + \sum_{(i,j)\in A}(c_{ij} + u_{ij} - \sum_{k\in K} w^k_{ij})y_{ij} + \\ & \sum_{(i,j)\in A}\sum_{k\in K}(v^k_i - v^k_j + w^k_{ij})f^k_{ij}. \end{aligned}$$

The value of the Lagrangian dual function $Z(\mathbf{v},\mathbf{w},\mathbf{u})$ for any vector $(\mathbf{v},\mathbf{w},\mathbf{u})$ is given by

$$Z(\mathbf{v},\mathbf{w},\mathbf{u}) = \min_{(\mathbf{y},\mathbf{f})} \left\{ L(\mathbf{y},\mathbf{f},\mathbf{v},\mathbf{w},\mathbf{u}) \middle| \begin{array}{l} y_{ij} \geq 0 \text{ and integer for all } (i,j) \in A, \\ f^k_{ij} \geq 0 \text{ for all } (i,j) \in A \text{ and } k \in K. \end{array} \right\}.$$

The Lagrangian dual problem is

$$\begin{array}{ll} \text{Maximize} & Z(\mathbf{v},\mathbf{w},\mathbf{u}) \\ \text{subject to:} & \mathbf{w} \geq \mathbf{0};\ \mathbf{u} \geq \mathbf{0};\ \mathbf{v} \text{ unrestricted in sign.} \end{array}$$

An identical argument to the one used in Section 4.2 shows that for a given node potential vector $\bar{\mathbf{v}}$, we obtain the optimal value of the Lagrangian function by setting $w_{ij}^k = \max(0, \bar{v}_j^k - \bar{v}_i^k)$ and $u_{ij} = \max(0, \sum_{k \in K} w_{ij}^k - c_{ij})$. Substituting the expression for the optimal value of w_{ij}^k (given $\mathbf{v}$) in the equation for the optimal value of u_{ij} (given $\mathbf{w}$), we obtain $u_{ij} = \max(0, \sum_{k \in K} \max(0, \bar{v}_j^k - \bar{v}_i^k) - c_{ij})$. Therefore, we can write the Lagrangian dual problem as a maximization problem over the node potentials $\mathbf{v}$. The Lagrangian dual problem is

$$\begin{aligned} &\text{Maximize } Z(\mathbf{v}) \\ &\text{subject to no constraint on } \mathbf{v} \end{aligned}$$

with

$$Z(\mathbf{v}) = \sum_{k \in K} \sum_{i \in N} d_i^k v_i^k - \sum_{(i,j) \in A} b_{ij} \overbrace{\max(0, \underbrace{\sum_{k \in K} \max(0, v_j^k - v_i^k)}_{w_{ij}^k} - c_{ij})}^{u_{ij}}. \tag{4.15}$$

To best understand the modifications needed in the dual-ascent algorithm, it is worthwhile observing the effect of a change in the node potential of a single commodity k at a single node j, i.e., v_j^k, on the term u_{ij} of the dual function. Let s_{ij}^k denote $c_{ij} - \sum_{l \neq k} \max(0, v_j^l - v_i^l)$. As before, s_{ij}^k remains fixed because we are fixing all node potentials except for v_j^k. Consequently, $u_{ij} = \max(0, \max(0, v_j^k - v_i^k) - s_{ij}^k)$, and $s_{ij} = \max(0, s_{ij}^k - \max(0, v_j^k - v_i^k))$. Figure 4-12 shows the value of u_{ij} as a function of v_j^k with all the other node potentials held fixed. As long as $v_j^k \leq v_i^k$ the term $\max(0, v_j^k - v_i^k)$ is zero, so u_{ij} is either zero (case (b)), or $-s_{ij}^k$, if s_{ij}^k is negative (case (a)). In case (a), since the slack $s_{ij} = \max(0, s_{ij}^k - \max(0, v_j^k - v_i^k))$ is equal to zero at $v_j^k = v_i^k$, any increase in v_j^k increases u_{ij}. Figure 4-12a shows case (a). In case (b) the value of u_{ij} is equal to zero when $v_j^k - v_i^k \leq s_{ij}^k$. Any increase in v_j^k beyond $v_j^k = v_i^k$ decreases the slack, leaving $u_{ij} = 0$, until $v_j^k = v_i^k + s_{ij}^k$ where the slack is zero. From this point on, u_{ij} increases linearly as a function of v_j^k. Figure 4-12b illustrates case (b).

As before, we associate with each commodity $k \in K$ a directed network $D^k = (N, A^k)$, defined on the nodes N and arcs $A^k = A$. An arc in the network D^k can either be active, i.e., weakly active or strongly active, or inactive. Equation (4.7) and Figure 4-12 specify the conditions for each state (they are the same as for the NDC model). We also define the active network for commodity k by the active arcs in D^k. Consider the complementary

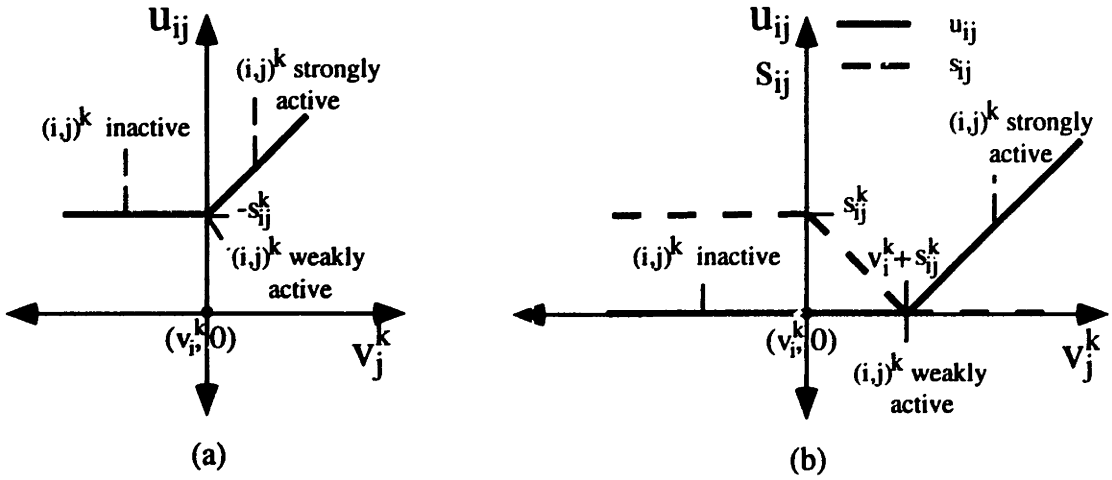


Figure 4-12: u_{ij} as a function of node potential v_j^k. (a) Case (a): $s_{ij}^k \leq 0$; (b) Case (b): $s_{ij}^k > 0$.

slackness conditions.

$$
\begin{aligned}
& y_{ij} > 0 \Longrightarrow \sum_{k \in K} w_{ij}^k - u_{ij} = c_{ij}; \\
& f_{ij}^k > 0 \Longrightarrow v_j^k - v_i^k = w_{ij}^k; \\
& u_{ij} > 0 \Longrightarrow y_{ij} = b_{ij}; \qquad (4.16) \\
\text{and} \quad & w_{ij}^k > 0 \Longrightarrow f_{ij}^k = y_{ij}.
\end{aligned}
$$

The first two conditions, i.e., primal complementary slackness, permit the primal solution to include only tight arcs, and permit commodity k to flow on arc (i,j) only if $v_j^k - v_i^k = w_{ij}^k$ (or $v_j^k \geq v_i^k$). Therefore, primal complementary slackness again implies every inactive arc has zero flow.

Restricting our attention to basic directions and calculating the directional derivative of the dual cost for any basic direction of the form γ_T^k (with the aid of Figure 4-12 and equation (4.15)), we obtain

$$
Z'(\mathbf{v}; \gamma_T^k) = \sum_{i \in T} d_i^k + \sum_{\substack{(j,i)^k \text{:strongly active} \\ i \notin T, j \in T}} b_{ji} - \sum_{\substack{(i,j)^k \text{:active} \\ i \notin T, j \in T}} b_{ij}. \qquad (4.17)
$$

Notice that this equation is identical to equation (4.8), the directional derivative of the dual cost of the NDC problem for any basic direction of the form γ_T^k. Since we derived the

dual-ascent algorithm for the NDC problem from equation (4.8) and the directed network associated with each commodity, the same dual-ascent algorithm, with a few minor changes to calculate the values of the dual variables and to calculate the stepsize in the procedure ascent-step, provides a feasible heuristic solution and a lower bound for the DNDC problem.

To be more specific, we modify the equations that define $\mathbf{y}, \mathbf{w}, \mathbf{u}, \mathbf{s}$ and $Z(\mathbf{v})$ as follows.

$$\begin{aligned} y_{ij} &= \max_{k \in K} f_{ij}^k \\ w_{ij}^k &= \max(0, v_j^k - v_i^k) \\ u_{ij} &= \max(0, \sum_{k \in K} \max(0, v_j^k - v_i^k) - c_{ij}) \\ s_{ij} &= \max(0, c_{ij} - \sum_{k \in K} \max(0, v_j^k - v_i^k)) \\ Z(\mathbf{v}) &= \sum_{k \in K} \sum_{i \in N} d_i^k v_i^k - \sum_{\{i,j\} \in E} b_{ij} \max(0, \sum_{k \in K} \max(0, v_j^k - v_i^k) - c_{ij}) \end{aligned} \tag{4.18}$$

To determine the amount of ascent in an ascent step, consider the type of arcs directed into and out of an ascent set (see Figure 4-9 on page 107). Recall, the dual-ascent procedure determines the stepsize of an ascent direction by moving along that direction until the rate of ascent changes (the slope of the dual function changes).

1. $(j,i)^k$ is strongly active: The algorithm increases v_j^k until $(j,i)^k$ becomes weakly active. At this point the rate of ascent changes. Recall from Figure 4-12 that in case (a), $(j,i)^k$ becomes weakly active when $v_j^k = v_i^k$, and in case (b), this occurs when u_{ij} decreases to value zero. Therefore, the increase is determined by $\min(v_i^k - v_j^k, u_{ij})$.

2. $(j,i)^k$ is weakly active: As v_j^k increases, the arc will become inactive. Moreover, as v_j^k increases, the arc cannot become active. Therefore, we need not consider these arcs (because they cannot change the rate of ascent).

3. $(j,i)^k$ is inactive: We need not consider these arcs because increasing v_j^k cannot change the state of $(j,i)^k$ (and so cannot change the rate of ascent).

4. $(i,j)^k$ is inactive: We may increase v_j^k until the arc becomes weakly active. This occurs when v_j^k is increased by $s_{ij} + \max(0, v_i^k - v_j^k)$.

5. $(i,j)^k$ is (weakly or strongly) active: We need not consider these arcs because they remain active as v_j^k increases.

Consequently, the following equation specifies the stepsize in the procedure *ascent-step*:

$$\alpha = \min_{\substack{i \notin T(k) \\ j \in T(k)}} \left\{ \begin{array}{ll} \min(u_{ij}, v_i^k - v_j^k) & \text{if } (j,i)^k \text{ is strongly active;} \\ \\ s_{ij} + \max(0, v_i^k - v_j^k) & \text{if } (i,j)^k \text{ is inactive.} \end{array} \right\} \quad (4.19)$$

By replacing equation (4.13) with equation (4.19) in the procedure *ascent-step* and equation (4.6) with equation (4.16), we obtain a dual-ascent algorithm for the DNDC problem (with arc connectivity requirements).

Arguing as in Section 4.3, we can establish several properties, e.g., finite termination, of the dual-ascent algorithm. As an example, we show that if no arc in the dicut $\delta^-(T(k))$ satisfies one of the two conditions in equation (4.19), then the problem is infeasible. Because no arc satisfies the conditions in equation (4.19), for all arcs (i,j) in $\delta^-(T(k))$, $f_{ij}^k = b_{ij}$ and for all arcs (j,i) in $\delta^+(T(k))$, $f_{ji}^k = 0$. Since the residual demand of $T(k)$ is greater than zero

$$\sum_{j \in T(k)} \bar{d}_j^k = \sum_{j \in T(k)} (d_j^k - (\sum_{i \in N} f_{ij}^k - \sum_{l \in N} f_{jl}^k)) \quad > \quad 0.$$

Rearranging terms gives

$$d_{D(k)}^k - \sum_{(i,j) \in \delta^-(T(k))} f_{ij}^k + \sum_{(i,j) \in \delta^+(T(k))} f_{ji}^k > 0;$$

or

$$\sum_{(i,j) \in \delta^-(T(k))} b_{ij} < q_k.$$

Therefore, $\delta^-(T(k))$ is an $O(k)$–$D(k)$ dicut with capacity less than commodity k's requirement, and so the problem is infeasible.

The results in this section demonstrate the flexibility and adaptability of the dual-ascent approach. By incorporating a few simple changes, in equation (4.12) and (4.13), the same dual-ascent algorithm applies to both the NDC and DNDC problem. Therefore, from an implementation perspective, the same code (with minor modifications) is applicable for both problems.

4.5 Node-Connectivity Requirements

In this section we show how to modify the dual-ascent algorithm to the NDC model and the DNDC model that have both edge/arc- and node-connectivity requirements. Later, we describe several modeling extensions to the NDC and DNDC models, and show how to apply the dual-ascent algorithm to them. In the subsequent discussion we restrict our

attention to the DNDC model and explain the necessary modifications to the dual-ascent algorithm. Modifying the dual-ascent algorithm for the NDC model is similar.

By following the Lagrangian duality approach on Formulation (4.14), we could develop the dual-ascent algorithm for the DNDC problem with connectivity matrix **R** that contains both arc- and node-connectivity requirements. For a more appealing derivation of the dual-ascent algorithm, we pursue a slightly different approach and develop the dual-ascent algorithm by considering the DNDC problem on a transformed digraph.

First, we make a slight modification to constraint (4.14d) in the directed flow formulation for the DNDC (4.14). Replace constraint (4.14d) by

$$\sum_{i \in N} f_{il}^k \leq b^k \quad \text{for all } k \in K \text{ and all } i \neq O(k) \text{ or } D(k), \tag{4.20}$$

with $b^k = 1$ if commodity k models a node-connectivity requirement, and $b^k = q_k$ if commodity k models an arc-connectivity requirement. Notice that if commodity k models a node-connectivity requirement, this constraint permits only a unit of flow through any transhipment node (i.e., a node that is not the origin or destination of the commodity) in the network and thus ensures q_k node-disjoint paths. On the other hand, if commodity k models an arc-connectivity requirement, the constraint imposes no limit (beyond the arc capacity limits) on the flow of commodity k through any transhipment node in D.

Now apply the node splitting procedure, described in Section 4.1.1, to the digraph $D = (N, A)$. The node splitting procedure replaces node i by two nodes i_1 and i_2. Arcs directed out of node i are now directed out of node i_2. Arcs directed into node i are now directed into node i_1. The transformation also introduces arcs of the form (i_1, i_2). In the new digraph $\bar{D} = (\bar{N}, \bar{A})$, obtained by the node splitting procedure, let $\bar{A}_N$ denote the set of arcs that represent nodes in the original digraph D (e.g., arcs of the form (i_1, i_2)), and let $\bar{A}_A$ denote the set of arcs that represent arcs in the original digraph (i.e., $\bar{A} \backslash \bar{A}_N$). The capacity and cost of an arc in $\bar{A}_A$, say (i_2, j_1), is equal to the capacity and cost of its counterpart in A, i.e., arc (i, j). Since arcs in $\bar{A}_N$ correspond to nodes in D, they have zero cost.

Now translate the directed flow formulation for the DNDC problem (4.14) on D to the digraph $\bar{D}$. Suppose node s is the origin and node t is the destination for commodity k in D; then node s_2 is the origin and node t_1 is the destination of commodity k in $\bar{D}$. To transform constraint (4.20) to the directed flow model on $\bar{D}$, let b_{ij}^k denote the maximum amount of flow of commodity k that can flow on an arc (i, j) in $\bar{A}_N$. If commodity k models a node-connectivity requirement of D, set $b_{ij}^k = 1$ for all arcs in $\bar{A}_N$. Otherwise, if commodity k models an arc-connectivity requirement in D, set $b_{ij}^k = q_k$ for all arcs in $\bar{A}_N$. The transformation of the remaining constraints is straightforward. We, therefore, obtain

the following formulation of the DNDC problem on the transformed digraph $\bar{D}$.

Directed flow model for the DNDC problem on $\bar{D}$:

$$\text{Minimize} \quad \sum_{(i,j)\in\bar{A}_A} c_{ij}y_{ij} \tag{4.21a}$$

$$\text{subject to:} \quad \sum_{j\in\bar{N}} f_{ji}^k - \sum_{l\in\bar{N}} f_{il}^k = \begin{cases} -q_k & \text{if } i = O(k); \\ q_k & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } i \in \bar{N} \text{ and } k \in K \tag{4.21b}$$

$$f_{ij}^k \le y_{ij} \qquad \text{for all } (i,j) \in \bar{A}_A \text{ and } k \in K \tag{4.21c}$$

$$f_{ij}^k \le b_{ij}^k \qquad \text{for all } (i,j) \in \bar{A}_N \text{ and for all } k \in K \tag{4.21d}$$

$$f_{ij}^k \ge 0 \qquad \text{for all } (i,j) \in \bar{A}_A \cup \bar{A}_N \text{ and } k \in K \tag{4.21e}$$

$$y_{ij} \le b_{ij} \qquad \text{for all } (i,j) \in \bar{A}_A; \tag{4.21f}$$

$$y_{ij} \ge 0 \qquad \text{and integer, for all } (i,j) \in \bar{A}_A. \tag{4.21g}$$

Since we obtained this model from Formulation (4.14) and the node-splitting procedure, it is easy to see that it is a valid representation of the DNDC problem. We can obtain the optimal solution to the DNDC problem on D from the optimal solution to the DNDC problem on $\bar{D}$ as follows: include arc $(i,j) \in A$ in the optimal solution to model (4.14) if the corresponding arc $(i_2, j_1) \in \bar{A}_A$ is in the optimal solution to model (4.21). Since we obtained Formulation (4.21) from Formulation (4.14), these two formulations are also equivalent as LPs.

Although doing so might be repetitive, to be complete we now briefly derive the dual problem to Formulation (4.21). Viewing v_i^k, w_{ij}^k, t_{ij}^k and u_{ij} as Lagrange multipliers for constraints (4.21b), (4.21c), (4.21d) and (4.21f), we obtain the following Lagrangian function:

$$\begin{aligned} L(\mathbf{y},\mathbf{f},\mathbf{v},\mathbf{w},\mathbf{u},\mathbf{t}) = {} & \sum_{(i,j)\in\bar{A}_A} c_{ij}y_{ij} + \sum_{(i,j)\in\bar{A}_A}\sum_{k\in K}(f_{ij}^k - y_{ij})w_{ij}^k + \sum_{(i,j)\in\bar{A}_A}(y_{ij} - b_{ij})u_{ij} + \\ & \sum_{(i,j)\in\bar{A}_N}\sum_{k\in K}(f_{ij}^k - b_{ij}^k)t_{ij}^k + \sum_{i\in\bar{N}}\sum_{k\in K}(d_i^k - \sum_{j\in\bar{N}} f_{ji}^k + \sum_{l\in\bar{N}} f_{il}^k)v_i^k. \end{aligned}$$

Rearranging the terms, we obtain

$$\begin{aligned} L(\mathbf{y},\mathbf{f},\mathbf{v},\mathbf{w},\mathbf{u},\mathbf{t}) = {} & \sum_{i\in\bar{N}}\sum_{k\in K} d_i^k v_i^k - \sum_{(i,j)\in\bar{A}_A} b_{ij}u_{ij} - \sum_{(i,j)\in\bar{A}_N} b_{ij}^k t_{ij}^k + \\ & \sum_{(i,j)\in\bar{A}_A}\sum_{k\in K}(v_i^k - v_j^k + w_{ij}^k)f_{ij}^k + \sum_{(i,j)\in\bar{A}_A}(c_{ij} + u_{ij} - \sum_{k\in K} w_{ij}^k)y_{ij} + \\ & \sum_{(i,j)\in\bar{A}_N}\sum_{k\in K}(v_i^k - v_j^k + t_{ij}^k)f_{ij}^k. \end{aligned}$$

An identical argument to the one used in Section 4.2 shows that for a given node potential vector $\bar{\mathbf{v}}$, we obtain the optimal value of the Lagrangian dual function, $Z(\mathbf{v},\mathbf{w},\mathbf{t},\mathbf{u}) = \min_{(\mathbf{y},\mathbf{f})}\{L(\mathbf{y},\mathbf{f},\mathbf{v},\mathbf{w},\mathbf{u},\mathbf{t}) | \text{subject to (4.21e) and (4.21g)}\}$, by setting $w_{ij}^k = \max(0, \bar{v}_j^k - \bar{v}_i^k)$, $t_{ij}^k = \max(0, \bar{v}_j^k - \bar{v}_i^k)$, and $u_{ij} = \max(0, \sum_{k\in K} \max(0, \bar{v}_j^k - \bar{v}_i^k) - c_{ij})$. Therefore, we can write the Lagrangian dual problem as a maximization problem over the node potentials $\mathbf{v}$. The Lagrangian dual problem is

$$\begin{aligned} &\text{Maximize } Z(\mathbf{v}) \\ &\text{subject to no constraint on } \mathbf{v} \end{aligned}$$

with

$$Z(\mathbf{v}) = \sum_{k\in K}\sum_{i\in\bar{N}} d_i^k v_i^k - \sum_{(i,j)\in\bar{A}_A} b_{ij} \overbrace{\max\Big(0, \sum_{k\in K} \underbrace{\max(0, v_j^k - v_i^k)}_{w_{ij}^k} - c_{ij}\Big)}^{u_{ij}} - \sum_{(i,j)\in\bar{A}_N}\sum_{k\in K} b_{ij}^k \underbrace{\max(0, v_j^k - v_i^k)}_{t_{ij}^k}. \tag{4.22}$$

Notice, because there is a one-to-one correspondence between the arcs in A and the arcs in $\bar{A}_A$, and between the constraints defined on A in Formulation (4.14) and the constraints defined on $\bar{A}_A$ in Formulation (4.21), the second term in the dual function is similar to the second term in the dual function (4.15) (i.e., for arcs A). The third term of the function is due to the arcs in $\bar{A}_N$ and constraint (4.21d).

As before, we associate with each commodity $k \in K$ a directed network $\bar{D}^k = (\bar{N}, \bar{A}^k)$ defined on the nodes $\bar{N}$ and arcs $\bar{A}^k = \bar{A}$. Now, consider the complementary slackness conditions for Formulation (4.21).

$$\begin{aligned} & y_{ij} > 0 \Longrightarrow \sum_{k\in K} w_{ij}^k - u_{ij} = c_{ij} && \text{if } (i,j) \in \bar{A}_A; \\ & f_{ij}^k > 0 \Longrightarrow v_j^k - v_i^k = w_{ij}^k && \text{if } (i,j) \in \bar{A}_A; \\ & f_{ij}^k > 0 \Longrightarrow v_j^k - v_i^k = t_{ij}^k && \text{if } (i,j) \in \bar{A}_N; \\ & u_{ij} > 0 \Longrightarrow y_{ij} = b_{ij} && \text{if } (i,j) \in \bar{A}_A; \\ & w_{ij}^k > 0 \Longrightarrow f_{ij}^k = y_{ij} && \text{if } (i,j) \in \bar{A}_A; \\ \text{and} \quad & t_{ij}^k > 0 \Longrightarrow f_{ij}^k = b_{ij}^k && \text{if } (i,j) \in \bar{A}_N. \end{aligned} \tag{4.23}$$

The first three conditions are the primal complementary slackness conditions. For arcs in $\bar{A}_A$, the primal complementary slackness conditions are identical to the primal complementary slackness condition for arcs (i,j) in A in (4.16). Therefore, we define the states of the arcs in $\bar{A}_A$, i.e., active, inactive, etc., as in equation (4.1). The third primal complementary condition permits flow on an arc in $\bar{A}_N$ only if $v_j^k - v_i^k = t_{ij}^k$, or if $v_j^k \geq v_i^k$. For arcs in $\bar{A}_N$ we define $(i,j)^k$ as active if $v_j^k \geq v_i^k$ and inactive otherwise. If $v_j^k = v_i^k$, we say $(i,j)^k$ is weakly active, and if $v_j^k > v_i^k$, we say arc $(i,j)^k$ is strongly active. Observe again, primal complementary slackness implies every inactive arc has zero flow.

Calculating the directional derivative of the dual cost for a basic direction of the form γ_T^k, we obtain

$$Z'(\mathbf{v};\gamma_T^k) = \sum_{i \in T} d_i^k + \sum_{\substack{(j,i)^k:\text{strongly active} \\ i \notin T, j \in T, (j,i) \in \bar{A}_A}} b_{ji} - \sum_{\substack{(i,j)^k:\text{active} \\ i \notin T, j \in T, (i,j) \in \bar{A}_A}} b_{ij} + \sum_{\substack{(j,i)^k:\text{strongly active} \\ i \notin T, j \in T, (j,i) \in \bar{A}_N}} b_{ji}^k - \sum_{\substack{(i,j)^k:\text{active} \\ i \notin T, j \in T, (i,j) \in \bar{A}_N}} b_{ij}^k. \tag{4.24}$$

Suppose we interpret b_{ij} as the "capacity" of an arc $(i,j)^k$ when $(i,j) \in \bar{A}_A$, and interpret b_{ij}^k as the capacity of an arc $(i,j)^k$ when $(i,j) \in \bar{A}_N$. Then, equation (4.24) is identical to equation (4.17), the dual function for the DNDC problem, when we interpret b_{ij} as the capacity of an arc $(i,j)^k$ in equation (4.17) (b_{ij} and b_{ij}^k are constants). Therefore, the dual-ascent algorithm for the arc-connectivity version of the DNDC problem can be applied to the DNDC problem on $\bar{D}$.

Accordingly, the necessary changes to the dual-ascent algorithm are as follows.

1. For the arcs in $\bar{A}_N$, the variable $t_{ij}^k = \max(0, v_j^k - v_i^k)$. For these arcs, $(i,j)^k$ is active if $v_j^k \geq v_i^k$, weakly active if $v_j^k = v_i^k$, strongly active if $v_j^k > v_i^k$, and inactive otherwise.
2. Any reference to b_{ij} in the dual-ascent algorithm, becomes b_{ij}^k for arcs in $\bar{A}_N$.

Before we conclude this section, we point out a relationship between the duals of Formulations (4.14) and (4.21). Consider the following LP dual to Formulation (4.14).

Dual to Formulation (4.14):

$$\text{Maximize} \quad \sum_{k \in K} q_k v_{D(k)}^k - \sum_{(i,j) \in A} b_{ij} u_{ij} - \sum_{k \in K} \sum_{i \in N} t_i^k \tag{4.25a}$$

$$\text{subject to:} \quad v_j^k - t_i^k - v_i^k \leq w_{ij}^k \quad \text{for all } (i,j) \in A \tag{4.25b}$$

$$\sum_{k \in K} w_{ij}^k - u_{ij} \leq c_{ij} \quad \text{for all } (i,j) \in A \tag{4.25c}$$

$$u_{ij} \geq 0 \quad \text{for all } (i,j) \in A \tag{4.25d}$$

$$w_{ij}^k \geq 0 \quad \text{for all } (i,j) \in A \tag{4.25e}$$

$$t_j^k = 0 \quad \text{if } j = O(k), D(k) \tag{4.25f}$$

$$t_j^k \geq 0 \quad \text{if } j \neq O(k), D(k). \tag{4.25g}$$

To transform a feasible solution to the dual of the DNDC model on $\bar{D}$, i.e., the dual function (4.22), to a feasible solution of equal cost to this dual, we argue as follows. For every node i in N and commodity k in K, set $v_i^k = v_{i_1}^k$ (i_1 is the tail of the arc that represents node i in $\bar{D}$), and set $t_i^k = w_{i_1,i_2}^k$. For every arc in A, set $w_{ij}^k = w_{i_2 j_1}^k$ and $u_{ij} = u_{i_2 j_1}$ (arc (i_2, j_1) is the counterpart of arc (i,j) in $\bar{D}$). Then,

$$v_j^k - t_i^k - v_i^k = v_{j_1}^k - (v_{i_2}^k - v_{i_1}^k) - v_{i_1}^k = v_{j_1}^k - v_{i_2}^k = w_{i_2 j_1}^k = w_{ij}^k.$$

Therefore the solution satisfies constraint (4.25b). It is easy to see that it also satisfies the remaining constraints in Formulation (4.25), and that this dual solution has equal cost in both models.

Consequently, by interpreting the operations of the dual-ascent algorithm on the digraph D, we can obtain a dual-ascent algorithm that operates on the original digraph D. Since the algorithmic details are rather straightforward, but the description tedious, we omit them.

For NDC problems with node-connectivity requirements, we could transform the NDC problem to a DNDC problem and apply this algorithm. Alternatively we could either interpret this dual-ascent algorithm on the undirected network, or directly derive a dual-ascent algorithm on the undirected network following the Lagrangian dual approach. Since both these procedures are relatively easy, and result in a dual-ascent algorithm similar to the DNDC problem (except for the shape of the u_{ij} function, and thus the equations for the dual variables), we omit the details.

4.5.1 Combined Edge- and Node-Connectivity Requirements Between Nodes

In some instances, network planners need to protect against both node and edge failures. Sometimes network planners design networks that survive fewer node failures than edge failures. We describe two models, introduced by Grötschel and Monma [GM90], for combined edge- and node-connectivity require.nents.

The first model has the following connectivity requirements. For each pair of nodes s and t, construct r_{st} edge-disjoint paths and p_{st} node-disjoint paths between nodes s and t ($1 \leq p_{st} \leq r_{st}$). Edge connectivity and node connectivity are two extreme instances of this model. For edge connectivity, set $p_{st} = 1$, and for node connectivity set $p_{st} = r_{st}$. It is easy

to model these combined requirements in the NDC model (or the DNDC model). Create two commodities k and k', one to model edge connectivity (or arc connectivity for the DNDC model) and one to model node connectivity, both with origin node s and destination node t, setting $q_k = r_{st}$ and $q_{k'} = p_{st}$. Since commodity k' models a node-connectivity requirement, it belongs to the set of commodities $K^{[n]}$.

In this model, we created two commodities for each pair of nodes. If p_{st} is 1, 2, or r_{st}, we can decrease the number of commodities by creating only a single commodity between nodes s and t. It is clear why one commodity is sufficient when p_{st} is 1 or r_{st}. If $p_{st} = 2$, create a single commodity $k \in K^{[n]}$ between nodes s and t with $q_k = r_{st}$. In addition, set the righthand side of constraint (4.2d) in the NDC model (or constraint (4.14d) in the DNDC model) to $r_{st} - 1$. This choice ensures that no transhipment node allows more than $r_{st} - 1$ units of commodity k to flow through it, which implies that deleting a node will destroy at most $r_{st} - 1$ paths leaving one path between nodes s and t intact.

To develop the second model, suppose a network contains r_{st} edge-disjoint paths between nodes s and t. Network planners might want a certain fraction of the r_{st} paths between these nodes to survive a single node failure. For example, suppose the network has 10 edge-disjoint paths between nodes s and t. Subsequent to a node failure, network planners might want to maintain at least 8 edge-disjoint paths between nodes s and t. This model specifies connectivity requirements between nodes s and t as a pair $(r_{st}, \bar{p}_{st})$, with r_{st} the number of required edge-disjoint paths between nodes s and t, and $\bar{p}_{st}$ the number of required edge-disjoint paths that would survive a node failure.[13] Again edge and node connectivity are special cases of this model. We can model edge-connectivity requirements by setting $\bar{p}_{st} = 0$, and node-connectivity requirements by setting $\bar{p}_{st} = r_{st} - 1$. This model might be appropriate in situations when the number of edge-disjoint paths between nodes s and t reflect the amount of traffic between nodes s and t. Therefore, guaranteeing the network contains $\bar{p}_{st}$ edge-disjoint paths between nodes s and t after a node failure, ensures $\bar{p}_{st}/r_{st}$ units of the traffic between nodes s and t survives a node failure.

To model the connectivity requirements in this model, create a commodity $k \in K^{[n]}$, and set $q_k = r_{st}$, and $b^k = r_{st} - \bar{p}_{st}$. Then, as in equation (4.20), set the righthand side of equation (4.2d) to b^k in the undirected flow formulation for the NDC problem (and the righthand side of equation (4.14d) the directed flow formulation for the DNDC problem).

These two models are similar in three cases. Edge connectivity, node connectivity, and when $p_{st} = 2$ in the first model and $\bar{p}_{st} = 1$ in the second model (we assume that the r_{st} values are equal in this comparison). In other cases, the two models are different. In

[13]This model is a special case of a very general model that Grötschel and Monma introduced. Their model specifies a triplet, $(r_{st}, \bar{p}_{st}, \bar{q}_{st})$, with r_{st} the number of required edge-disjoint paths between nodes s and t, and $\bar{p}_{st}$ the number of required edge-disjoint paths that would survive at most $\bar{q}_{st}$ node failures.

the first model, p_{st} prescribes the number of node-disjoint paths between a pair of nodes. Consequently, the only way to ensure a large fraction of the paths between s to t survive a single node failure is to make them node disjoint. This requirement might be a stronger than necessary, especially if node failures are fairly rare. In the second model, $\bar{p}_{st}$ prescribes the number of edge-disjoint paths that survive a single node failure. The model will, therefore, ensure a large fraction of the edge-disjoint paths survive a single node failure. However, unlike the first model, this model does not consider the consequences of more than a single node failure.

Since both these models are special versions of Formulation (4.2) (and Formulation (4.14) for the DNDC problem), the dual-ascent algorithm described earlier in this section is applicable to these models.

4.5.2 Node Costs

Nodes in telecommunication networks often represent the location of switching equipment. Switching equipment (and other transmission equipment, like concentrators and repeaters) is very expensive. The cost of a switch can be as much as several million dollars. Optimization models that permit node costs allow network planners to analyze facility location decisions (i.e., whether or not to install a switch) in addition to the survivability issues.

To consider NDC and DNDC problems with node costs, we assume that whenever we select an edge or arc in the network adjacent to any node, we need to install transmission equipment at that node. Moreover, once the transmission equipment is in place at a node, it is capable of handling traffic from all edges or arcs incident to it. Nodes that have connectivity requirements, i.e., those with at least one positive entry in the row or column corresponding to the node in the connectivity matrix $\mathbf{R}$, must necessarily incur the costs at the node. Thus, no optimization is necessary for these nodes. Consequently, we assume these node costs are zero and optimize only over the Steiner nodes in the network.

In this discussion, we consider node costs for the DNDC problem. Modeling node costs for the NDC is similar. The models and dual-ascent algorithm described in this chapter can incorporate node costs only if all the connectivity requirements in the DNDC problem (or NDC problem) are node-connectivity requirements. Then, the node splitting procedure will permit us to convert the problem to one with only arc-connectivity requirements.

As before, let $\bar{D} = (\bar{N}, \bar{A})$ denote the digraph obtained by the node-splitting procedure, with $\bar{A}_N$ denoting the arcs in $\bar{A}$ that correspond to nodes in D and $\bar{A}_A$ denoting the arcs that correspond to arcs in D. Set the cost and capacity of an arc in $\bar{A}_A$ as before. For arcs in $\bar{A}_N$, set $b_{ij} = 1$ and set c_{ij} equal to the cost of the node it models. The resulting problem is an arc-connectivity problem on $\bar{D}$. Therefore, we can apply the dual-ascent algorithm for the DNDC problem with arc-connectivity requirements (described in Section 4.4).

If the problem contains arc-connectivity requirements, the digraph transformation procedure to capture node costs is not valid. Because this model permits only a unit of flow through an arc in $\bar{A}_N$, the procedure converts arc-connectivity requirements in the original digraph D to node-connectivity requirements. Thus, a different model (and dual-ascent algorithm) is necessary to model node costs in problems that contain arc-connectivity requirements. In this thesis, besides this discussion, we do not investigate models with node costs. This topic is an important issue that merits further investigation.

4.6 Single Commodity Problems

In the foregoing discussion, we described a dual-ascent algorithm for NDC and DNDC problems with general connectivity requirements. The dual-ascent algorithm is a heuristic method. Does it solve certain special cases of the NDC problem (or DNDC problem) optimally? In this section we provide a partial answer to this question by showing that the dual-ascent procedure solves single commodity problems optimally. Actually, we will show this result only for the q-edge-disjoint path problem. The proofs for the other single commodity problems[14] are similar. We also show that the dual-ascent algorithm is closely related to the successive shortest path algorithm for the minimum cost flow problem.

Theorem 4.6.1 *The dual-ascent algorithm for the q-edge-disjoint path problem either finds optimal primal and dual solutions* $(\mathbf{x}, \mathbf{f})$ *and* $(\mathbf{v}, \mathbf{w}, \mathbf{u})$ *or shows that the problem is infeasible.*

Proof: For convenience, we restate the complementary slackness conditions (4.6). Since this is a single commodity problem, we drop the superscript k.

$$
\begin{aligned}
& x_{ij} > 0 \implies w_{ij} - u_{ij} = c_{ij}; \\
& f_{ij} > 0 \implies v_j - v_i = w_{ij}; \\
& u_{ij} > 0 \implies x_{ij} = b_{ij}; \\
\text{and} \quad & w_{ij} > 0 \implies f_{ij} + f_{ji} = x_{ij}.
\end{aligned}
$$

Recall that we refer to the first two conditions as primal complementary slackness conditions and the last two conditions as dual complementary slackness conditions. Initially, the solution satisfies all four complementary slackness conditions. The algorithm permits only

[14]The other single commodity problems are (i) the q-node-disjoint path problem, (ii) the q-node-disjoint path problem with node costs, (iii) all the problems discussed in Section 4.5.1 that can be formulated with a single commodity, (iv) the q-arc-disjoint path problem, and directed versions of (i), (ii), and (iii).

one of the two variables f_{ij} and f_{ji} to be positive. Since the problem has a single commodity, the equation $x_{ij} = \max(f_{ij}, f_{ji})$ implies $f_{ij} + f_{ji} = x_{ij}$ for all edges. The solution always satisfies the second dual complementary slackness condition. In our discussion in Section 4.2.2 (on page 109), we showed that a dual-ascent step for a commodity k maintains the primal complementary slackness conditions for this commodity. Therefore, for a single commodity problem, the dual-ascent algorithm always maintains primal complementary slackness. Now we show if any arc (i, j) is strongly active, then $f_{ij} = b_{ij}$. Observe that before an arc becomes strongly active, it must have been weakly active. The dual-ascent algorithm adds node i to T if (i, j) is a weakly active arc directed into T and $f_{ij} < b_{ij}$. Therefore, arc (i, j) can become strongly active only after $f_{ij} = b_{ij}$. Also observe that the algorithm (in the procedure *increment-flow*) decreases the flow on strongly active arcs. Therefore, for all strongly active arcs $f_{ij} = b_{ij}$. This result implies that $x_{ij} = b_{ij}$ for all strongly active arcs (i, j), and so the solution satisfies the first dual complementary slackness condition as well.

Since the final primal and dual solutions are feasible and satisfy all the complementary slackness conditions, the final primal solution is optimal. ■

We have shown the dual-ascent algorithm solves the q-edge-disjoint path problem. This result shows that for all nonnegative-cost vectors, the LP relaxation of the undirected flow formulation for the q-edge-disjoint path problem has an integer optimal solution. Since the projection of the LP relaxation of the undirected flow formulation is the LP relaxation of the cutset formulation, this result implies that the LP relaxation of the cutset formulation for the q-edge-disjoint path problem also has integer optimal solutions whenever the cost vector is nonnegative.

Similarly, the LP relaxation of the undirected flow formulation for the q-node-disjoint path problem has an integer optimal solution and its projection, the LP relaxation of the cutset formulation for the q-node-disjoint path problem, also has integer optimal solutions whenever the cost vector is nonnegative. (We can make similar statements for the directed flow model for the DNDC problem and its projection, a dicut formulation).

4.6.1 Dual-Ascent and the Successive Shortest Path Algorithm for the Minimum Cost Flow Problem

For a single commodity problem, because $x_{ij} = \max(f_{ij}, f_{ji})$ and all costs are nonnegative, we can transfer the costs on the edges to the flows. That is, we can cast the problem as a minimum cost flow problem. Using this transformation, the 3-edge-disjoint problem in Figure 4-10 becomes the following minimum cost flow problem: send 3 units of flow from node s to node t when the cost of sending a unit of flow on an edge $\{i, j\}$ equals to c_{ij}. Therefore, any algorithm for the minimum cost flow problem is capable of solving the q-edge-disjoint path problem. In this section we show that the algorithm developed in

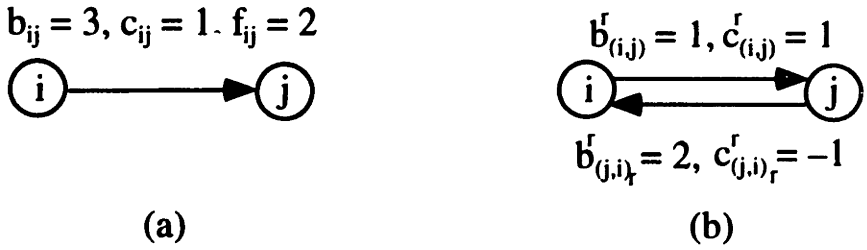


Figure 4-13: Residual network. (a) Arc (i, j) has a flow of 2 units and a capacity of 3 units. (b) Residual arcs. (i, j) has a capacity of 1 unit and $(j, i)_r$ has a capacity of 2 units.

Section 4.2 closely resembles the successive shortest path algorithm for the minimum cost flow problem.

First, let us formulate the q-edge-disjoint path problem as a minimum cost flow problem on a directed network as follows. Convert the undirected graph to a digraph $D = (N, A)$ by replacing each undirected edge $\{i, j\}$ by two directed arcs (i, j) and (j, i). The cost c_{ij} of sending a unit of flow on an arc (i, j) equals the cost of the undirected edge $\{i, j\}$ and the capacity b_{ij} of each arc equals the capacity of the undirected edge $\{i, j\}$. Set the supply of node s to q units, and the demand of node t to q units.

The successive shortest path algorithm operates on the *residual network* $D(\mathbf{f})$ corresponding to flow $\mathbf{f}$, defined as follows. Replace arc (i, j) with a capacity b_{ij} by two arcs (i, j) (the forward arc) and $(j, i)_r$ (the reverse arc[15]). Arc (i, j) has cost $c_{(i,j)}$ $(= c_{ij})$ and *residual capacity* $b^r_{(i,j)} = b_{ij} - f_{ij}$, and arc $(j, i)_r$ has cost $c_{(j,i)_r} = -c_{(i,j)}$ and residual capacity $b^r_{(j,i)_r} = f_{ij}$. The residual network contains those arcs with positive residual capacity. Figure 4-13 illustrates a simple residual network.

The successive shortest path algorithm selects a supply node and a demand node (s and t for the q-edge-disjoint path problem). It then finds the shortest directed path between nodes s and t in the residual network, sends a unit of flow on this path, and updates the residual network and demands. It proceeds in this fashion until the demand for the commodity is zero.

Owing to the presence of reverse arcs, the residual network might contain negative-cost arcs. The successive shortest path algorithm cleverly transforms the costs of the arcs in the residual network so that (i) all arc costs are positive, (ii) the shortest path between a pair of nodes remain unchanged. To transform the costs, it associates with each node in the residual network a *node price* v_i. The *reduced cost* of a forward arc (i, j) with respect to a node price vector $\mathbf{v}$ is $c^v_{(i,j)} = c_{ij} + v_i - v_j$. The reduced cost of a reverse arc $(i, j)_r$ with respect to node price vector $\mathbf{v}$ is $c^v_{(i,j)_r} = -c^v_{(j,i)}$. The successive shortest path algorithm maintains $c^v_{ij} \geq 0$ for all arcs (i, j) in the residual network. The shortest paths with respect

[15]We use the subscript r to differentiate the reverse arc (j, i) from the forward arc (j, i)

to the reduced costs are identical to the shortest paths with respect to the original costs. Therefore, we can apply a shortest path algorithm like Dijkstra's [Dij59] to the residual network to find the shortest paths. At termination, the successive shortest path algorithm obtains a feasible flow $\mathbf{f}$ whose residual network has no arcs with negative reduced cost. These are the optimality conditions for minimum cost flow problems (see Ahuja, Magnanti and Orlin [AMO93]).

The dual-ascent algorithm for the q-edge-disjoint path problem is identical to the successive shortest path algorithm. We establish this result in two parts. First we show that if $\mathbf{f}^p$ is the flow vector and $\mathbf{v}^p$ the node potential vector at the start of subroutine *phase*(p) in the dual-ascent algorithm, then all the arcs in the residual network $D(\mathbf{f}^p)$ with node price vector equal to the node potential vector $\mathbf{v}^p$ have positive reduced cost. To show that the node prices for the residual network are $\mathbf{v}^p$, we change our notation slightly and refer to the residual network as $D(\mathbf{f}^p, \mathbf{v}^p)$. Second, we show that at the conclusion of subroutine *phase*(p) the dual-ascent algorithm sends a unit of flow on the shortest path between nodes s and t in $D(\mathbf{f}^p, \mathbf{v}^p)$.

Before establishing these results, we clarify some notation. To differentiate between the arc (i,j) in the directed network (in the dual-ascent algorithm) and the arc (i,j) in the residual network, we add a superscript k to the arcs in the directed network. Now, consider the directed network at the beginning of subroutine *phase*(p) in the dual-ascent algorithm. Observe that:

1. If $(i,j)^k$ is inactive, arc (i,j) has no flow. Therefore, $D(\mathbf{f}^p, \mathbf{v}^p)$ contains arc (i,j), but not $(j,i)_r$. Either $v_j \geq v_i$, in which case $s_{ij} > 0$, and so $c_{ij} + v_i - v_j > 0$, or $v_j < v_i$, in which case $c_{ij} + v_i - v_j > 0$. Therefore, arc (i,j) in $D(\mathbf{f}^p, \mathbf{v}^p)$ will always have positive reduced cost.

2. If $(i,j)^k$ is weakly active, then $s_{ij} = 0$. So $c_{ij} + v_i - v_j = 0$. If $f_{ij} < b_{ij}$, then the forward arc (i,j) is in $D(\mathbf{f}^p, \mathbf{v}^p)$ and its reduced cost is zero. If $f_{ij} > 0$, then the reverse arc $(j,i)_r$ is in $D(\mathbf{f}^p, \mathbf{v}^p)$ and its reduced cost is also zero.

3. If $(i,j)^k$ is strongly active, $s_{ij} = 0$ and $u_{ij} > 0$. Therefore, $c_{ij} + v_i - v_j < 0$. Because $f_{ij} = b_{ij}$ (in Theorem 4.6.1 we showed that $f_{ij} = b_{ij}$ for strongly active arcs), $D(\mathbf{f}^p, \mathbf{v}^p)$ does not contain arc (i,j), but contains reverse arc $(j,i)_r$. The reduced cost of arc $(j,i)_r$ is $-c^v_{(i,j)} = -c_{ij} - v_i + v_j > 0$.

By considering arc $(j,i)^k$, we can similarly show that (j,i) and $(i,j)_r$ in $D(\mathbf{f}^p, \mathbf{v}^p)$ have positive reduced cost. Therefore, all the arcs in $D(\mathbf{f}^p, \mathbf{v}^p)$ have positive reduced cost.

We now show that at the conclusion of subroutine *phase*(p) the dual-ascent algorithm sends flow on the shortest path between nodes s and t in $D(\mathbf{f}^p, \mathbf{v}^p)$. We establish this result with the help of Theorem 4.6.1, which shows the dual-ascent algorithm solves the shortest

path problem. Therefore, the following (dual-ascent) algorithm finds the shortest (directed) path from node s to t in a digraph with no negative-cost arcs.

Shortest path algorithm

1. Initially set $T = \{t\}$ and set $d_i = 0$ for all nodes in the digraph.
2. Find the arc (i,j) directed into T with the smallest value of $c_{ij} + d_i - d_j$. Set $d_k = d_k + (c_{ij} + d_i - d_j)$ for all nodes k in T. Add node i to T and set **next**$[i] = j$.
3. If $s \in T$, stop. Otherwise, repeat Step 2.

Backtracking, using the **next** array, we obtain the shortest path. Note, because the algorithm increases d_j only for nodes in T, $d_i = 0$ for nodes not in T. Thus, in Step 2 the algorithm finds the arc (i,j) with the smallest value of $c_{ij} - d_j$ and adds node i to T.

Let us interpret this algorithm in a way that will make the connection between the dual-ascent algorithm and the successive shortest path algorithm more transparent. The shortest path algorithm maintains a dicut $\delta^-(T)$. For each arc (i,j) in the dicut $\delta^-(T)$, the maximum increase in the node label d_j is $c_{ij} - d_j$. That is, the value of d_j at which $c_{ij} + d_i - d_j = 0$. The algorithm considers all arcs across the dicut, (i) selects the arc (i,j) with the minimum value of $c_{ij} - d_j$, (ii) increases the node potentials by $c_{ij} - d_j$ for all nodes in T, and (iii) adds node i to T.

Some thought shows that this algorithm is similar to a "reverse" implementation of Dijkstra's algorithm. That is, it starts by setting $T = \{t\}$. Then it finds the least cost arc (i,j) directed into T, adds node i to T, and sets **next**$[i] = j$. It continues in this fashion until node s is in T.

Suppose we use this algorithm to find the shortest directed path from node s to node t in the residual network. Then in each iteration, to select the node i to add to T the algorithm considers the value of $c^v_{(i,j)} - d_j$ for a forward arc (i,j) directed into T, and considers the value of $c^v_{(i,j)_r} - d_j$ for a reverse arc $(i,j)_r$ directed into T. Note that

$$\begin{aligned} c^v_{(i,j)} - d_j &= c_{ij} + v_i - v_j - d_j = c_{ij} + v_i - (v_j + d_j) \\ c^v_{(i,j)_r} - d_j &= -c_{ji} - v_j + v_i - d_j = -c_{ji} + v_i - (v_j + d_j) \end{aligned} \tag{4.26}$$

The subroutine *phase* of the dual-ascent algorithm starts with $T = \{t\}$. It adds all nodes i to T, for which $(i,j)^k$ is active with $f_{ij} < b_{ij}$ or $(j,i)^k$ is weakly active with $f_{ji} > 0$ and sets **next**$[i][k] = j$ (ignore the sign for the moment). To show that the dual-ascent algorithm performs identical steps to the shortest path algorithm on $D(\mathbf{f}^p, \mathbf{v}^p)$, we show that in subroutine *phase*(p),

1. Node potentials for any $i \notin T$ remain unchanged.

2. Once a node is added to T, it remains in T until the algorithm calls the procedure *increment-flow*(k) (i.e., increases the flow by a unit).

3. The algorithm adds nodes to T in the same order as the shortest path algorithm.

First, by definition, an ascent step changes the node potentials only for nodes in T. Therefore, the first statement is true. Second, since this is a single commodity problem, no complementary slackness violations take place in the dual-ascent algorithm. Consequently the subroutine *phase* terminates after calling *increment-flow*(k).

Suppose instead of increasing the node label d_j, the shortest path algorithm increases the node potential v_j (i.e., $v_j = v_j + d_j$). Then, from equation (4.26), in each iteration the shortest path algorithm will consider the value of $c_{ij} + v_i - v_j$ for a forward arc (i,j) directed into T and consider the value of $-c_{ij} + v_i - v_j$ for a reverse arc $(i,j)_r$ directed into T.

The *find-ascent-set* procedure of the dual-ascent algorithm adds a node i to T if either $(i,j)^k$ is active with $f_{ij} < b_{ij}$, or $(j,i)^k$ is weakly active with $f_{ji} > 0$. The former case corresponds to adding i to T because $c_{ij} + v_i - v_j = 0$ for arc (i,j) in the residual network, and the latter case corresponds to adding i to T because $-c_{ji} + v_i - v_j = 0$ for arc $(i,j)_r$ in the residual network.

Now, if T is an ascent set, consider the five possible cases (see page 112). In each case $j \in T, i \notin T$. Note that the first case is the only one for which there is a reverse arc directed into T.

1. $(j,i)^k$ is strongly active: If $(j,i)^k$ is strongly active it implies that $f_{ji} = b_{ji}$. The maximum amount the dual-ascent algorithm increases v_j before adding i to T is $\min(u_{ij}, v_i - v_j)$, which is u_{ij}. From equation (4.12), $u_{ij} = -c_{ji} + v_i - v_j$. Since $f_{ji} = b_{ji}$ the residual network contains the reverse arc $(i,j)_r$. (Note, we need not consider the arc (i,j).[16])

2. $(j,i)^k$ is weakly active: In this case $f_{ji} = 0$; otherwise the algorithm would have added node i to T. Because this is a single commodity problem, in the dual-ascent algorithm we do not need to stop an ascent step when $v_j = v_i$. Instead the maximum increase is $2(v_i - v_j)$. Since $(j,i)^k$ is weakly active $c_{ij} = v_i - v_j$. Therefore the maximum increase $2(v_i - v_j) = c_{ij} + v_i - v_j$.

3. $(j,i)^k$ is inactive and $v_j < v_i$: The maximum increase for v_j in this scenario is $s_{ij} + 2(v_i - v_j)$. This corresponds to considering arc (i,j) in the residual network. Substituting $s_{ij} = c_{ij} + v_j - v_i$, we find the maximum increase is $c_{ij} + v_i - v_j$.

[16] If both $(i,j)_r$ and (i,j) are contained in $D(\mathbf{f}^p, \mathbf{v}^p)$, then only $(i,j)_r$ needs to be considered in the shortest path algorithm. Why? The reduced cost of arc (i,j), $c^v_{(i,j)}$, is equal to $c_{ij} + v_i - v_j$, and the reduced cost of arc $(i,j)_r$, $c^v_{(i,j)_r}$, is $-c_{ji} + v_i - v_j$. Because all edge costs are nonnegative $-c_{ji} \leq c_{ij}$, and so $c^v_{(i,j)_r} \leq c^v_{(i,j)}$.

4. $(i,j)^k$ is inactive and $v_j \geq v_i$: The maximum increase for v_j in this scenario is s_{ij}. Note $s_{ij} = c_{ij} + v_i - v_j$.

5. $(i,j)^k$ is active with $f_{ij} = b_{ij}$: There is no limit on the amount of increase in the node potential. Observe that because $f_{ij} = b_{ij}$ and $f_{ij} = 0$, arc (i,j) and arc $(i,j)_r$ are not in the residual network.

We have thus shown that if arc (i,j) is directed into T in the residual network, the maximum amount the dual-ascent algorithm will increase the node potential v_j before adding node i to T is $c_{ij} + v_i - v_j$. Similarly, if arc $(i,j)_r$ is directed into T in the residual network the maximum amount the dual-ascent algorithm will increase the node potential v_j before adding node i to T is $-c_{ij} + v_i - v_j$. Therefore the steps of the dual-ascent algorithm in subroutine *phase*(p) are identical to the shortest path algorithm on the residual network $D(\mathbf{f}^p, \mathbf{v}^p)$.

The preceding discussion shows that the dual-ascent algorithm can be considered as a generalization of Dijkstra's algorithm. Suurbaale [Suu74] describes an algorithm for the q-arc-disjoint path problem and the q-node-disjoint path problem (the node-disjoint path problem can be formulated as a min-cost flow problem by using the node splitting procedure) where arc replication is not permitted. His algorithm essentially interprets the steps of the successive shortest path algorithm on the original network. Our discussion also shows that the dual-ascent algorithm can also be considered as a generalization of Suurbaale's algorithm.

The foregoing discussion suggests that one might be able to derive the successive shortest path algorithm by following a Lagrangian duality approach. This is indeed the case. For a detailed discussion and derivation of the successive shortest path method as a dual-ascent method see Bertsekas [Ber91].

4.7 Improvements

The dual-ascent algorithm provides a heuristic solution to the NDC problem. In this section we discuss two improvements to the algorithm. When the dual-ascent algorithm terminates, it provides a feasible primal solution $(\mathbf{x}, \mathbf{f})$ and a feasible dual solution and an associated lower bound. The first improvement increases the cost of the dual solution, i.e., the lower bound. The second improvement decreases the cost of the primal solution, i.e., the cost of the heuristic solution.

At the conclusion of the dual-ascent procedure, there might still be a basic direction of ascent that improves the dual objective (i.e., lower bound). In this case we can improve the lower bound by following a steepest basic ascent procedure. We have previously shown (on page 103) how to find the steepest basic ascent direction. Although finding a steepest

basic ascent direction is computationally expensive, it might be worthwhile pursuing this approach because (i) it has the potential to increase the lower bound, and (ii) we expect only a few iterations of improvement.

The primal solution $\mathbf{x}$ might contain more edges than necessary to satisfy the connectivity requirements. We can improve the primal solution solution by dropping edges if deleting them does not create an infeasible primal solution. There are many different ways to consider dropping the edges from the network. For instance, we could maintain an ordered list of the edges in the network as they became tight, and delete them in LIFO fashion. (Note that we must exercise care during the course of the algorithm to ensure that only tight edges are on the list. Consequently, if an edge on the list has positive slack we must delete it from the list). Another approach drops the edges in greedy fashion. This heuristic would consider the edges for dropping in order of decreasing cost. Another variation of both these heuristics uses the auxiliary network (instead of the primal feasible solution $\mathbf{x}$) as the initial primal feasible network.

This concludes our discussion of the dual-ascent algorithm for the NDC and the DNDC problem (note that both the improvements can be applied to the DNDC problem). To conclude this chapter, we compare our dual-ascent algorithm with Goemans et al.'s algorithm for the NDC problem, and then summarize the contributions of our discussion.

4.8 Comparison with Goemans et al.'s Algorithm

In this section we describe the Goemans et al.'s algorithm for the NDC problem with edge-connectivity requirements. Then, we compare the dual-ascent algorithm for the NDC problem with this algorithm. The Goemans et al.'s algorithm was developed in a sequence of papers (see [WGMV93], [GGW93], and [GGP+94]). For convenience, we refer to the algorithm as the GW algorithm (since Goemans and Williamson are the common co-authors in the three papers [WGMV93], [GGW93], and [GGP+94]). Our presentation mainly follows the description of these results in Williamson's Ph.D. thesis [Wil93].

The GW algorithm applies to the class of network design problems that can be formulated as follows:

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{4.27a}$$

$$\text{subject to:} \quad \sum_{\{i,j\} \in \delta(S)} x_{ij} \geq f(S) \quad \text{for all } S \subset N; \tag{4.27b}$$

$$x_{ij} \in \{0,1\} \quad \text{for all } \{i,j\} \in E. \tag{4.27c}$$

The function $f(S)$ satisfies the following "proper" conditions:

- [Symmetry] $f(S) = f(N - S)$ for all $S \subset N$; and
- [Maximality] If A and B are disjoint, then $f(A \cup B) \leq \max\{f(A), f(B)\}$.

Also, $f(N) = 0$ for every proper function f.

Notice that with the choice $f(S) = \text{econ(S)}$ of the proper function, Formulation (4.27) models all NDC problems with edge-connectivity requirements (in the model that does not permit edge replication). Actually, proper functions model more general problems like the minimum cost matching and the minimum cost T-join problem.[17] However, Formulation (4.27) cannot model node-connectivity requirements and the other generalizations of the NDC that we described in Sections 4.5.1 and 4.5.2.

4.8.1 Description and Analysis of the GW Algorithm

The GW algorithm is a fairly complicated algorithm. We will explain it in three parts. First, we give a high-level description of the algorithm. Next, we describe in some detail one of its key subroutines. Then, we show that the algorithm is a dual-ascent algorithm that has a worst case ratio of $2\mathcal{H}(f_{\max})$, where $\mathcal{H}(f_{\max}) = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{f_{\max}}$. It is instructive to pursue this analysis because it highlights the differences in the underlying design methodology of the two algorithms.

Let us set some notation that we shall need in describing the algorithm. Let $f_{\max}$ denote the maximum value of $f(S)$ over all sets $S \subset N$. Suppose $\mathbf{X}$ denotes the set of edges in the integer primal solution (not necessarily feasible). Let $\Delta(S) = f(S) - |\delta_{\mathbf{X}}(S)|$ denote the deficiency of cut $\delta(S)$. That is, given a solution $\mathbf{X}$, $\Delta(S)$ denotes the minimum number of edges, in the cut $\delta(S)$, that need to be added to $\mathbf{X}$ to satisfy constraint (4.27b).

The GW algorithm works in stages. In the first stage, it considers all sets S with deficiency $f_{\max}$. It then uses a subroutine called *approx-uncrossable* to find a solution that has at least one edge in each such cut $\delta(S)$. It adds the edges in the solution obtained by *approx-uncrossable* to the primal solution $\mathbf{X}$. Let $\mathbf{X}_p$ denote the set of edges in the primal solution $\mathbf{X}$ at the end of stage p. Then in stage p, the GW algorithm considers the current solution $\mathbf{X}_{p-1}$ and identifies all maximally deficient cuts, that is, cuts $\delta(S)$ with the maximum value of $\Delta_p(S) = f(S) - |\delta_{\mathbf{X}_{p-1}}(S)|$ (we use the subscript p in $\Delta_p(S)$ to denote stage p). It then calls *approx-uncrossable* which finds a solution $\mathbf{X}'$ that (i) does not contain edges in the current solution $\mathbf{X}_{p-1}$, and (ii) has at least one edge in each maximally deficient cut $\delta(S)$. It adds the edges in $\mathbf{X}'$ to $\mathbf{X}_{p-1}$, the solution at the end of stage $p-1$, to obtain $\mathbf{X}_p$, the solution at the end of stage p (i.e., $\mathbf{X}_p = \mathbf{X}_{p-1} \cup \mathbf{X}'$). Notice that at the end of

[17] Given a graph $G = (N, E)$ and a subset of nodes T, a set of edges $E' \subseteq E$ is a T-join if, in the subgraph $G' = (N, E')$, the degree of a node i is odd if and only if $i \in T$. The minimum cost T-join is to find a T-join at minimum cost.

a stage, the GW algorithm decreases the deficiency of the maximally deficient cuts by at least 1. Shortly, in the discussion of *approx-uncrossable*, we will show that the algorithm decreases the deficiency of at least one maximally deficient cut by exactly one. Therefore, at the end of stage 1 the maximum deficiency is $f_{\max} - 1$ and, in general, at the conclusion of stage p, the maximum deficiency is $f_{\max} - p$. Consequently, at the conclusion of stage $f_{\max}$, the solution $\mathbf{X}_{f_{\max}}$ satisfies all the connectivity requirements.

We now describe the key subroutine of the GW algorithm, *approx-uncrossable*, in more detail. In stage p, define

$$h_p(S) = \begin{cases} 1 & \text{if } \Delta_p(S) = f_{\max} - p + 1; \\ 0 & \text{otherwise.} \end{cases}$$

In stage p, *approx-uncrossable* considers the following network design problem on the graph $G_p = (N_p, E_p)$, where $E_p = E - \mathbf{X}_{p-1}$.

Network design problem for stage p:

$$\text{Minimize} \quad \sum_{\{i,j\} \in E_p} c_{ij} x_{ij} \tag{4.28a}$$

$$\text{subject to:} \quad \sum_{\{i,j\} \in \delta_{E_p}(S)} x_{ij} \geq h_p(S) \quad \text{for all } S \subset N; \tag{4.28b}$$

$$x_{ij} \in \{0,1\} \quad \text{for all } \{i,j\} \in E_p. \tag{4.28c}$$

Approx-uncrossable is a dual-ascent algorithm. In order to describe *approx-uncrossable*, consider the dual to the LP relaxation of the network design problem for stage p.

Dual to network design problem for stage p:

$$\text{Maximize} \quad \sum_{S \subset N} h_p(S) w_S \tag{4.29a}$$

$$\text{subject to:} \quad \sum_{S:\{i,j\} \in \delta(S)} w_S \leq c_{ij} \quad \text{for all } \{i,j\} \in E_p \tag{4.29b}$$

$$w_S \geq 0 \quad \text{for all } S \subset N. \tag{4.29c}$$

Let us now outline a dual-ascent algorithm for the network design problem for stage p (i.e., model (4.28)). For convenience, we drop the subscript p in the following discussion (i.e., we refer to $h_p(S)$ as $h(S)$, E_p as E, and $\delta_{E_p}(S)$ as $\delta(S)$). Consider the complementary

slackness conditions for the LP relaxation of Formulation (4.28) and its dual.

$$x_{ij} > 0 \Longrightarrow \sum_{S:\{i,j\}\in\delta(S)} w_S = c_{ij}; \tag{4.30a}$$

and

$$w_S > 0 \Longrightarrow \sum_{\{i,j\}\in\delta(S)} x_{ij} = h(S). \tag{4.30b}$$

By design, the dual-ascent algorithm maintains primal complementary slackness (4.30a) and relaxes dual complementary slackness (4.30b). We refer to $c_{ij} - \sum_{S:\{i,j\}\in\delta(S)} w_S$ as the slack s_{ij}. As before, if the slack of an edge is zero, we say the edge is *tight*. We define the auxiliary network by the tight edges. Initially, set $w_S = 0$ for all $S \subset N$. Consider the auxiliary network. Suppose we identify a set S with $h(S) = 1$ that has no tight edge in the cut $\delta(S)$. Then, by increasing w_S, we improve the dual objective function. Note that as we increase w_S, the slack for all edges in $\delta(S)$ decreases until some edge $\{i,j\}$ becomes tight. At this point, the algorithm adds that edge $\{i,j\}$ to the auxiliary network. We refer to a set S with $h(S) = 1$ that has no tight edge as an *ascent set.*[18]

A generic dual-ascent algorithm considers the auxiliary network and performs the following steps:

1. Find an ascent set S.
2. Increase w_S until an edge becomes tight. If $\delta(S)$ contains no edge stop. The problem is infeasible.
3. Repeat Steps 1 and 2 until every set S with $h(S) = 1$ has a tight edge in $\delta(S)$.

At the conclusion of these steps, the auxiliary network is feasible. However, a solution with fewer edges might be feasible. Therefore, the dual-ascent algorithm performs a LIFO drop heuristic.

The GW algorithm is a specialization of this generic dual algorithm. It considers minimal ascent sets. That is, an ascent set that contains no subset that is also an ascent set. It then *simultaneously* increases w_S for all minimal ascent sets until some edge $\{i,j\}$ becomes tight.

Williamson, Goemans, Mihail and Vazirani [WGMV93] show that if the function $h(S)$ satisfies a certain technical property (called uncrossable), the algorithm's solution is no more than twice the cost of the optimal solution. We state this result without proof.

Theorem 4.8.1 *Approx-uncrossable is a 2-approximation algorithm. In particular, the integer primal solution* $\mathbf{X}'$ *produced by approx-uncrossable is at most twice the cost of the*

[18]Note this definition of an ascent set is different from the definition of an ascent set earlier in this chapter. This definition is similar to the definition of an ascent set in Chapter 3.

dual feasible solution produced by approx-uncrossable. That is,

$$\sum_{\{i,j\}\in \mathbf{X}'} c_{ij} \leq 2 \sum_{S} w_S. \tag{4.31}$$

Recall, the two concepts—minimal ascent sets and the LIFO dropping rule—that we presented in Chapter 3 in the context of the branching problem. Increasing the dual variables w_S simultaneously for all minimal ascent sets results in a dual-ascent algorithm with a provable worst-case performance guarantee.

Using the worst-case performance ratio of approx-uncrossable, let us now show that the GW algorithm is a $2\mathcal{H}(f_{\max})$-approximation algorithm.

First, observe that we cannot delete the last edge selected by *approx-uncrossable* in stage p and maintain feasibility. This result implies the existence of some set S with $\Delta_p(S) = f_{\max} - p + 1$ and $|\delta_{\mathbf{X}'}(S)| = 1$. Because $\mathbf{X}' \subset E - \mathbf{X}_{p-1}$, $\Delta_{p+1}(S) = f_{\max} - p + 1 - 1$. That is, the algorithm decreases the deficiency of at least one maximally deficient cut by exactly 1 unit.

Consider the solution to Formulation (4.29) in stage p. We show how to convert this dual solution to a solution to the dual of Formulation (4.27); that is, a dual solution to the original problem. Formulation (4.27) has the following linear programming dual.

Dual to Formulation (4.27):

$$\text{Maximize} \quad Z_D = \sum_{S \subset N} f(S) w_S - \sum_{\{i,j\}\in E} u_{ij} \tag{4.32a}$$

$$\text{subject to:} \quad \sum_{S:\{i,j\}\in\delta(S)} w_S \leq c_{ij} + u_{ij} \quad \text{for all } \{i,j\} \in E; \tag{4.32b}$$

$$w_S \geq 0 \quad \text{for all } S \subset N; \tag{4.32c}$$

$$u_{ij} \geq 0 \quad \text{for all } \{i,j\} \in E. \tag{4.32d}$$

Given the dual variables constructed by *approx-uncrossable* in stage p, define $u_{ij} = \sum_{S:\{i,j\}\in\delta(S)} w_S$ for all $\{i,j\}$ in $\mathbf{X}_{p-1}$, and $u_{ij} = 0$ otherwise. We claim that the vector $(\mathbf{w}, \mathbf{u})$ is feasible to Formulation (4.32). To see this, note that the construction of $\mathbf{w}$ by *approx-uncrossable*, implies that $\sum_{S:\{i,j\}\in\delta(S)} w_S \leq c_{ij}$ for $\{i,j\} \in E_p$. Thus, the solution satisfies constraint (4.32b) for $\{i,j\} \notin \mathbf{X}_{p-1}$. For $\{i,j\} \in \mathbf{X}_{p-1}$, the definition of u_{ij} ensures that the solution satisfies constraint (4.32b).

We now show that the GW algorithm has a worst case ratio of $2\mathcal{H}(f_{\max})$.

Theorem 4.8.2 *The GW algorithm has a worst case ratio of $2\mathcal{H}(f_{\max})$.*

Proof: Let Z_D^* denote the optimal value of Formulation (4.32). From arguments preceding

this theorem, we know that in stage p

$$Z_D^* \geq \sum_S f(S)w_S - \sum_{\{i,j\}\in E} u_{ij}$$

Observe that

$$\sum_{\{i,j\}\in E} u_{ij} = \sum_{\{i,j\}\in \mathbf{X}_{p-1}} \sum_{S:\{i,j\}\in\delta(S)} w_S = \sum_S \sum_{\{i,j\}\in\delta_{\mathbf{X}_{p-1}}(S)} w_S = \sum_S |\delta_{\mathbf{X}_{p-1}}(S)|w_S.$$

Therefore,

$$Z_D^* \geq \sum_S (f(S) - |\delta_{\mathbf{X}_{p-1}}(S)|)w_S.$$

Now in stage p, $w_S > 0$ only if the deficiency of S $(f(S) - |\delta_{\mathbf{X}_{p-1}}(S)|)$ is $f_{\max} - p + 1$. Thus,

$$Z_D^* \geq (f_{\max} - p + 1)\sum_S w_S.$$

Now using Theorem 4.8.1 (i.e., equation (4.31)) and summing over all stages, we obtain

$$\sum_{\{i,j\}\in \mathbf{X}_{f_{\max}}} c_{ij} \leq 2\sum_{p=1}^{f_{\max}} \frac{1}{f_{\max} - p + 1} Z_D^* = 2\mathcal{H}(f_{\max})Z_D^*.$$

■

4.8.2 Comparison

We can now compare the dual-ascent algorithm for the NDC problem to the GW algorithm.

We developed the dual-ascent algorithm for the NDC problem from the flow-based formulation (the dual of the flow-based formulation). The GW algorithm is based upon a cutset formulation that models the edge-connectivity version of the NDC problem (and extensions of these problems). Even though these formulations are equivalent as LPs, they lead to different dual-ascent algorithms. As shown in Sections 4.4, and 4.5, our dual-ascent algorithm easily generalizes to node-connectivity problems, DNDC problems, problems with combined connectivity requirements, and problems with node costs, whereas the GW algorithm does not. On the other hand, the GW algorithm is applicable to many problems that cannot be modeled with the flow-based formulation.

The dropping heuristics of the dual-ascent algorithm for the NDC problem and the

GW algorithm are different. As its last step, the dual-ascent algorithm for the NDC problem drops edges from the auxiliary network (described in Section 4.7). The GW algorithm drops edges at the conclusion of each stage. In the GW algorithm, if an edge is in the primal solution at the conclusion of a stage, it remains in the primal solution. Consequently, although it might be possible to drop an edge from the primal solution at the conclusion of the GW algorithm while maintaining feasibility, the edge remains in the primal solution.

The GW algorithm and the dual-ascent algorithm for the NDC problem define ascent sets, and perform dual-ascent steps on ascent sets, in quite different ways (they are working in different spaces). The GW algorithm considers minimal ascent sets and simultaneously increases the dual variables for all minimal ascent sets. Our dual-ascent algorithm for the NDC problem does not do this. It considers ascent sets sequentially. (Future research might consider performing ascent steps for the ascent sets simultaneously.)

The dual-ascent algorithm for the NDC problem solves the q-edge-disjoint path problem and the q-node-disjoint path problem optimally. The GW algorithm does not solve the q-edge-disjoint path problem. In fact, it can achieve its worst-case ratio, i.e., provide a solution that is $2\mathcal{H}(q)$ times the optimal solution, for the q-edge-disjoint path problem (see [GGP+94]).

One motivation for dual-ascent methods is as a lower bounding mechanism for optimization problems. (We are interested in providing lower bounds in addition to heuristic solutions for the NDC problem.) For this reason, let us compare the lower bound provided by the dual-ascent algorithm for the NDC with the best lower bound obtained by the GW algorithm. Just prior to Theorem 4.8.2, we showed how to convert the dual solution of the network design problem for stage p to a dual solution for the NDC problem. Let us compare the best lower bound over all stages given by the GW algorithm to the lower bound given by the dual-ascent algorithm for the NDC problem. As an example, consider the 3-edge-disjoint path problem shown in Figure 4-10. At the conclusion of stage 1, $\mathbf{X_1} = \{\{s,a\},\{a,b\},\{b,c\},\{c,d\},\{d,t\}\}$, $w_{\{t\}} = 1, w_{\{s\}} = 1, w_{\{t,f,d\}} = 1, w_{\{s,e,a\}} = 1, w_{\{t,f,d,c\}} = 0.5$ and $w_{\{s,e,a,b\}} = 0.5$. The dual solution to the NDC problem constructed from this solution has value 15 (since this is stage 1 all $\mathbf{u}$ variables are zero). At the conclusion of stage 2, $\mathbf{X_2} = \{\{s,e\},\{e,f\},\{f,t\}\}$, and $w_{\{t\}} = 1, w_{\{s\}} = 1, w_{\{t,f\}} = 2, w_{\{s,e\}} = 2, w_{\{t,f,d,c\}} = 1$ and $w_{\{s,e,a,b\}} = 1$. To convert this dual solution to the overall dual, we set $u_{sa} = u_{dt} = 3$, $u_{bc} = 2$. Therefore, we obtain an overall dual solution with a value of 16. At the conclusion of stage 3, $\mathbf{X_3} = \{\{s,d\},\{a,t\}\}$, and $w_{\{t\}} = w_{\{s\}} = 8$. To convert this dual solution to an overall dual solution, we set $u_{sa} = u_{se} = u_{dt} = u_{ft} = 8$ and thus obtain an overall dual solution with value 16. The best dual solution obtained by the GW algorithm is 16 units, whereas the optimal dual solution, obtained by our dual-ascent algorithm, is 25 units. Therefore, the dual-ascent algorithm for the NDC problem can provide better

lower bounds than the GW algorithm.

On the other hand we note that the GW algorithm provides a guaranteed worst-case bound (and was developed for this purpose) for all NDC problems with edge-connectivity requirements. The dual-ascent algorithm for the NDC problem has no worst-case guarantees (and we would not be surprised if for some examples, it can perform arbitrarily badly).

4.9 Concluding Remarks

In this chapter we compared a cutset formulation (with an exponential number of constraints) and a compact flow formulation for the NDC problem. We showed that the projection of the feasible space of the LP relaxation of the flow formulation is the feasible space of the LP relaxation of the cut formulation. We then showed how to decrease the number of commodities in the flow model for several NDC problems.

Our main contribution is a very flexible dual-ascent algorithm, based upon the flow formulation, for the NDC problem. This algorithm easily generalizes to the DNDC problem, problems with node costs, and problems with combined node- and edge-connectivity requirements between a pair of nodes.

We showed that the dual-ascent algorithm solves problems that are formulated with a single commodity in the flow formulation for the NDC optimally. The q-edge-disjoint path problem and the q-node-disjoint path problem are special cases. As a result, we also showed that (for nonnegative-cost vectors) the LP relaxation of the cutset formulation and the LP relaxation of the flow formulation give integer optimal solutions (i.e., the $\mathbf{x}$ variables are integer). We also showed that when applied to the q-edge-disjoint path problem, the dual-ascent algorithm mimics the successive shortest path algorithm for minimum cost flows.

We then compared the dual-ascent algorithm to Goemans et al.'s algorithm (GW algorithm), highlighting their similarities and differences. We showed that the for the q-edge-disjoint path problem, the dual-ascent algorithm provides both a better feasible solution and a lower bound than the GW algorithm. We believe that the dual-ascent algorithm for the NDC problem is particularly attractive because it is designed to generate both good lower bounds (dual solutions) and upper bounds (integer primal solutions).

In order to evaluate any algorithmic procedure, it is important to conduct empirical testing. We did not test the dual-ascent algorithm for two reasons. First, we found that it was difficult to gather real-world data, or characteristics of real-world data, for the NDC problem. Moreover, currently the most crucial problems for telcos appear to be network design problems with low connectivity requirements. In Chapter 6 we consider a specialization of the dual-ascent approach for the NDLC problem. As part of that development, we studied the computational performance of the dual-ascent approach for the NDLC problem.

In our future research, we intend to empirically test the dual-ascent algorithm for the NDC and DNDC problem.

Two possible directions for extending the research in this chapter appear to be particularly appealing. As described in Chapter 1, network planners need to both solve the topological design problem and make decisions on facilities to install in the network based on traffic demand estimates. One research avenue is to study whether it is possible to integrate these two decisions into a single step, and devise a dual-ascent methodology for the problem.

The integer multicommodity flow problem, which has many applications in transportation, is closely related to the NDC problem. In the integer multicommodity flow problem, we are given commodities and their associated flow requirements. We are also given capacity constraints on the total flow over all commodities that may flow on an edge (or arc). The problem imposes a cost of d_{ij}^k for sending a unit of flow of commodity k on edge $\{i,j\}$ (arc (i,j)). We wish to find a minimum cost routing of the flows, with the added restriction that we send only an integer amount of a flow on a path. The dual-ascent algorithm we described in this chapter can be adapted to provide feasible solutions and lower bounds for integer multicommodity flow problems as well. In his Ph.D. thesis, Balakrishnan [Bal84] describes a dual-ascent procedure for the capacitated fixed-charge network design problem. (The capacitated fixed charge network design problem is similar to the uncapacitated fixed charge network design problem. It has additional capacity constraints on the total flow over all commodities that may flow on each edge (or arc).) He describes an algorithm for problems with unit demands. By setting the fixed charge cost to zero, his algorithm can be used to solve integer multicommodity flow problems with unit demands. Using the Lagrangian duality approach we have described, it is very easy to generalize his algorithm to integer multicommodity flows with nonunit demands. The dual-ascent framework described in this chapter, along with the research results of Balakrishnan, has established a strong footing for pursuing research on developing dual-ascent procedures for integer multicommodity flow problems.

Chapter 5

Improved Formulation: Unitary Network Design Problem with Connectivity Requirements

In this chapter we consider models for the unitary network design problem with connectivity requirements (NDC). Recall that for the unitary NDC problem all nodes with a connectivity requirement (a node i has a connectivity requirement if $r_{ij} \geq 1$ for some node j) must be connected.

Motivated by a result due to Nash-Williams, we show how to improve upon the undirected flow model (4.2) for the unitary NDC problem by using a flow directing procedure. We also develop an equivalent directed cut model. These models improve upon the well-known cutset model (4.1). By how much? In order to at least partially answer this question and to compare the new models with the cutset model, we project out the flow variables from the improved flow model. By doing so, we derive three classes of valid inequalities—partition, odd-hole and combinatorial design inequalities—for the cutset model (these inequalities are generalizations of known inequalities for the Steiner tree problem). Each of these three classes of valid inequalities is new for the unitary NDC problem. For special case of the survivable network design problem (SND), the partition inequality is also known to be facet defining. (Recall that the SND problem is a unitary NDC problem with prescribed node connectivity parameters r_v for node v and either edge- or node-connectivity requirements defined by $r_{ij} = \min\{r_i, r_j\}$.)

The development in this chapter is motivated by a desire to develop better linear programming relaxations for network design problems. Considerable accumulated experience in the optimization literature has demonstrated the value of developing good linear programming relaxations of combinatorial optimization problems. In Chapter 6 we use the improved

flow model to develop a linear programming based heuristic for the network design problem with low connectivity requirements (NDLC). (Recall that the NDLC problem is a special case of the SND problem with $r_v \in (0, 1, 2)$.) Computational experience with this algorithm (it solves problems with 300 nodes and 3000 edges to within 4 percent of optimality) shows that the linear programming relaxation of the improved flow formulation provides a good approximation to the mixed integer programming formulation of this model.

Throughout most of this chapter, we consider the edge-connectivity version of the unitary NDC problem. At the end of the chapter, we examine node-connectivity requirements.

5.1 Formulations for the Unitary NDC Problem

For convenience, to begin our discussion we restate the cutset formulation and the undirected flow formulation for the unitary NDC problem defined on the undirected graph $G = (N, E)$.

Cutset formulation for the unitary NDC problem:

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{5.1a}$$

$$\text{subject to:} \quad \sum_{\{i,j\} \in \delta(S)} x_{ij} \geq \text{econ}(S) \quad \text{for all node sets } S \text{ with } \phi \neq S \neq N; \tag{5.1b}$$

$$x_{ij} \leq b_{ij} \quad \text{for all } \{i,j\} \in E; \tag{5.1c}$$

$$x_{ij} \geq 0 \quad \text{and integer, for all } \{i,j\} \in E. \tag{5.1d}$$

Recall that $\text{econ}(S) = \max\{r_{ij} | j \in S; i \in N \backslash S\}$ and that b_{ij} is the largest number of times that the design is permitted to replicate edge $\{i,j\}$ ($b_{ij} = 1$ for applications without edge replication). We also define the *maximum connectivity requirement* of a node i as $r_i = \text{econ}(\{i\})$.[1]

Undirected flow formulation for the unitary NDC problem:

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{5.2a}$$

$$\text{subject to:} \quad \sum_{j \in N} f_{ji}^k - \sum_{l \in N} f_{il}^k = \begin{cases} -q_k & \text{if } i = O(k); \\ q_k & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } i \in N \text{ and } k \in K \tag{5.2b}$$

[1] For the NDC problem, r_{st} need not be equal to $\min(r_s, r_t)$. For the SND problem, the maximum connectivity requirement of a node equals the connectivity requirement of a node.

$$\left.\begin{matrix} f_{ij}^k \\ f_{ji}^k \end{matrix}\right\} \leq x_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{5.2c}$$

$$f_{ij}^k, f_{ji}^k \geq 0 \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{5.2d}$$

$$x_{ij} \leq b_{ij} \qquad \text{for all } \{i,j\} \in E; \tag{5.2e}$$

$$x_{ij} \geq 0 \qquad \text{and integer, for all } \{i,j\} \in E. \tag{5.2f}$$

In this undirected flow formulation, K denotes the set of all commodities; $O(k)$, $D(k)$, and q_k, denote the origin, destination, and flow requirement for commodity $k \in K$. In this formulation, for each node pair $\{s,t\}$, we create a commodity from one of these nodes to the other with a flow requirement of r_{st} (Chapter 4 and our discussion later in this chapter show how to use fewer commodities).

Our discussion of the minimum spanning tree problem showed that directing the model improves the formulation. This result raises a natural question: is it possible to direct the NDC problem and improve its formulation as well? In this section we first show how to direct unitary NDC problems for situations when the connectivity requirements are all even or one. We then generalize this result, developing an improved model for any unitary NDC problem (i.e., even those with odd connectivity requirements). In Chapter 7, we develop an improved model for all NDC problems (i.e., nonunitary problems as well). At the end of this section, we compare the improved model with another model introduced by Balakrishnan, Magnanti and Mirchandani [BMM94a]. For ease of exposition, for the rest of this section we assume that the model does not permit edge replication.

5.1.1 Directing the Unitary NDC Problem

The following result due to Nash-Williams [NW60] provides a key ingredient for transforming the undirected formulation to a directed one.

Theorem 5.1.1 (Nash-Williams) *Suppose G is an undirected graph with r_{xy} edge-disjoint paths connecting each pair x and y of its nodes. Then it is possible to direct the graph (i.e., orient its edges) so that the resulting digraph contains $\lfloor r_{xy}/2 \rfloor$ arc-disjoint paths from node x to node y and $\lfloor r_{xy}/2 \rfloor$ arc-disjoint paths from node y to node x.*

Consider any unitary NDC problem whose connectivity requirements r_{st} are even or one. We can view any feasible integer solution to this problem as follows: it is connected and contains several 2-edge-connected components. If we contract the 2-edge-connected components, the solution becomes a tree. The edges on the tree are the *bridge* edges in the feasible solution before we contracted the 2-edge-connected components; that is, those whose removal disconnects the graph defined by that solution.

The Nash-Williams theorem permits us to direct the edges of each 2-edge-connected component so that for any pair of nodes i and j with $r_{ij} \geq 2$ (by assumption these requirements must be even), the network contains $r_{ij}/2$ directed arc-disjoint paths from node i to node j, and $r_{ij}/2$ directed arc-disjoint paths from node j to node i. Once oriented, each 2-edge-connected component contains a directed path between every pair of its nodes. Therefore, if nodes i and j belong to the same 2-edge-connected component and $r_{ij} = 2$, the oriented network contains a directed path from node i to node j and a directed path from node j to node i. To direct the bridges, consider the tree obtained by contracting each 2-edge-connected component of the solution. Select any one of the nodes created by the contraction as a root node and direct the tree away from this node.

Figure 5-1 illustrates this directing procedure. In this example a, b, c and g are 2-edge-connected components. Between every pair of nodes s and t in these components $r_{st} = 2$. We orient the edges of each component (see Figure 5-1b) so that it contains a directed path between every pair of nodes in each 2-edge-connected component. Next, we select the node created by contracting component b as the root node and direct the tree edges (i.e., the bridges of the solution) away from node b. Figure 5-1c shows the graph at the conclusion of the directing procedure.

These observations permit us to formulate the unitary NDC problem as follows. Let y_{ij} be 1 if edge $\{i,j\}$ is oriented from node i to node j in the directing procedure applied to the optimal solution (i.e., the oriented network contains arc (i,j)) and be 0 otherwise.

Directed cut formulation for the unitary NDC problem ($r_{st} \in \{0,1,\text{even}\}$):

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{5.3a}$$

$$\text{subject to:} \quad \sum_{(i,j) \in \delta^-(S)} y_{ij} \geq \frac{\text{econ}(S)}{2} \quad \text{if econ}(S) \geq 2\text{; for all } S, \tag{5.3b}$$

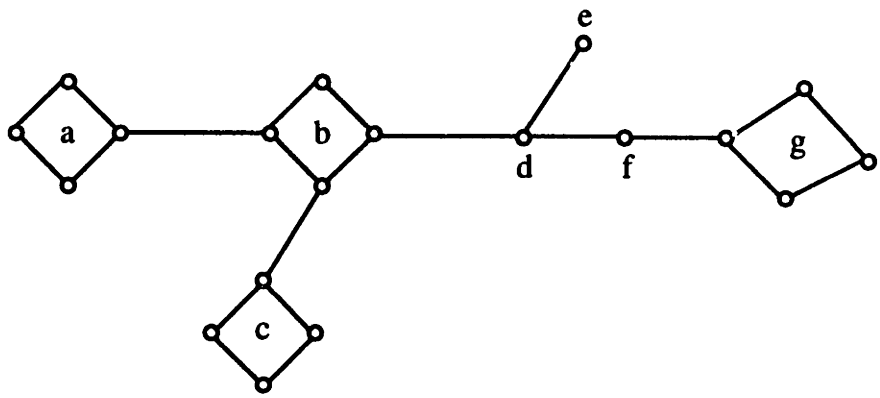
$$\sum_{(i,j) \in \delta^-(S)} y_{ij} \geq 1 \quad \text{if econ}(S) = 1\text{; for all } S\text{, root} \notin S \tag{5.3c}$$

$$y_{ij} + y_{ji} \leq x_{ij} \quad \text{for all } \{i,j\} \in E \tag{5.3d}$$

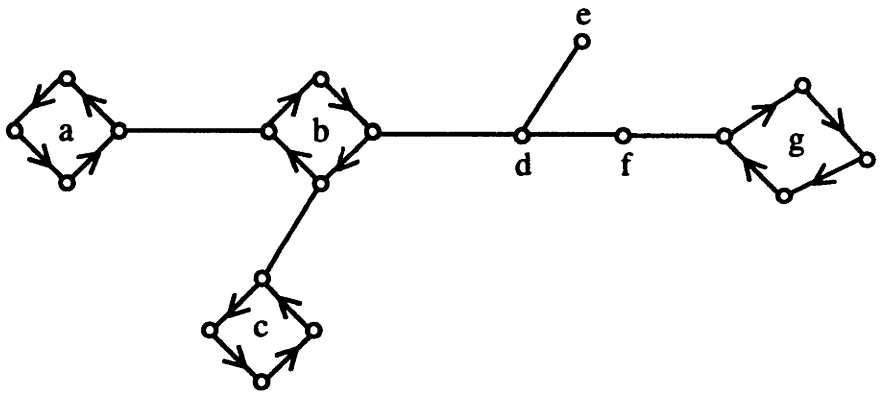
$$x_{ij} \leq 1 \quad \text{for all } \{i,j\} \in E \tag{5.3e}$$

$$y_{ij}, y_{ji}, x_{ij} \geq 0 \quad \text{and integer, for all } \{i,j\} \in E. \tag{5.3f}$$

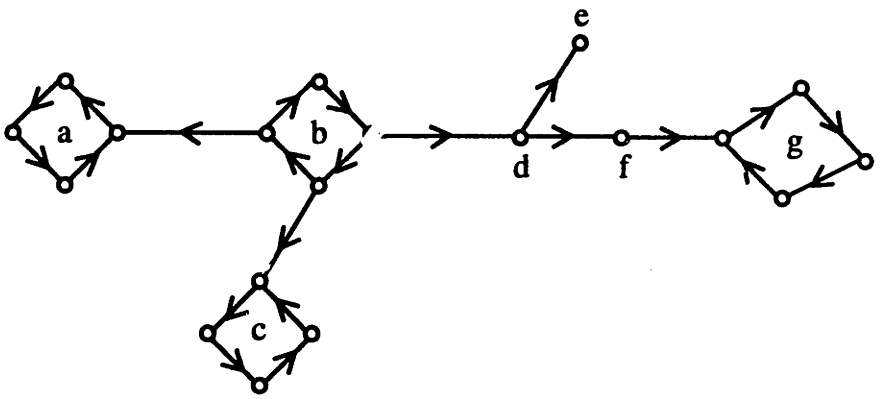
Since $\text{econ}(S) \equiv \max\{r_{ij} | j \in S; i \in N \backslash S\}$, constraint (5.3b) ensures that for every pair of nodes s and t with $r_{st} \geq 2$, every s–t dicut contains at least $r_{st}/2$ arcs and every t–s dicut contains at least $r_{st}/2$ arcs. Menger's theorem ensures that the oriented network contains at least $r_{st}/2$ arc-disjoint paths from node s to node t and $r_{st}/2$ arc-disjoint paths from node t to node s. Similarly, constraints (5.3c) and (5.3b) ensure that the oriented network



(a)



(b)



(c)

Figure 5-1: Directing the bridges of a feasible solution to the unitary NDC problem. (a) Feasible solution. (b) Direct the edges of each 2-edge-connected component. (c) Direct the bridges away from component b.

contains a directed path from the root to every other node. Constraint (5.3d) ensures that the oriented network contains at most one of the arcs (i,j) and (j,i).

Proposition 5.1.2 *The directed cut model is a valid formulation for the unitary NDC problem in the sense that* $(\mathbf{x},\mathbf{y})$ *is a feasible solution to this model if and only if* $\mathbf{x}$ *is an incidence vector of a feasible NDC design (that is,* $\mathbf{x}$ *is a feasible solution to the cutset model (5.1)).*

Proof: Suppose $\mathbf{x}$ is an incidence vector of a feasible NDC design. The argument preceding Formulation (5.3) shows how to construct an integer vector $\mathbf{y}$ so that $(\mathbf{x},\mathbf{y})$ is a valid solution for the directed cut model.

To establish the converse, suppose $(\mathbf{x},\mathbf{y})$ is a feasible solution to the directed cut model. Let Q be any node set with $\text{econ}(Q) = \text{econ}(N\backslash Q) \geq 2$. Combining inequality (5.3b) for $S = Q$ and $S = N\backslash Q$ and the inequalities (5.3d) summed over all $\{i,j\} \in \delta(Q)$ gives

$$\sum_{\{i,j\}\in\delta(Q)} x_{ij} \geq \sum_{(i,j)\in\delta^+(Q)} y_{ij} + \sum_{(j,i)\in\delta^-(Q)} y_{ji} \geq \text{econ}(Q).$$

If Q is any node set with $\text{econ}(Q) = \text{econ}(N\backslash Q) = 1$, assume without loss of generality that the root node is not in Q. Then the inequality (5.3c) with $S = Q$ and the inequality (5.3d) summed over $\{i,j\} \in \delta(Q)$ implies that

$$\sum_{\{i,j\}\in\delta(Q)} x_{ij} \geq 1.$$

Therefore, $\mathbf{x}$ is a feasible solution to the cutset formulation (5.1). ■

To see that the the linear programming relaxation of the directed cut formulation is stronger than linear programming relaxation of the cutset formulation, consider the NDLC example shown in Figure 5-2. In this example, each edge has unit cost. Nodes a, b and c have a connectivity requirement of 2. Nodes d, e and f have a connectivity requirement of 1. The optimal LP solution to the cutset formulation is $x_{ab} = x_{ac} = 1$ and $x_{bc} = x_{bd} = x_{cd} = x_{de} = x_{df} = x_{ef} = 0.5$. The optimal solution has value 5. The optimal solution to LP relaxation of the directed cut model has integer values for the edge variables. The solution is $y_{ab} = y_{bd} = y_{dc} = y_{ca} = y_{de} = y_{df} = 1$ with $x_{ab} = x_{ac} = x_{bd} = x_{cd} = x_{de} = x_{df} = 1$. The optimal LP solution has value 6. By reformulating the problem and solving the linear programming relaxation, we have obtained the optimal solution.

Some special cases of the directing procedure have appeared previously in the literature. Using the approach we have described, Goemans [Goe90] developed a directed formulation for the SND problem whose connectivity requirements are all even or one.

In the next section we show how to generalize the results in this section, developing

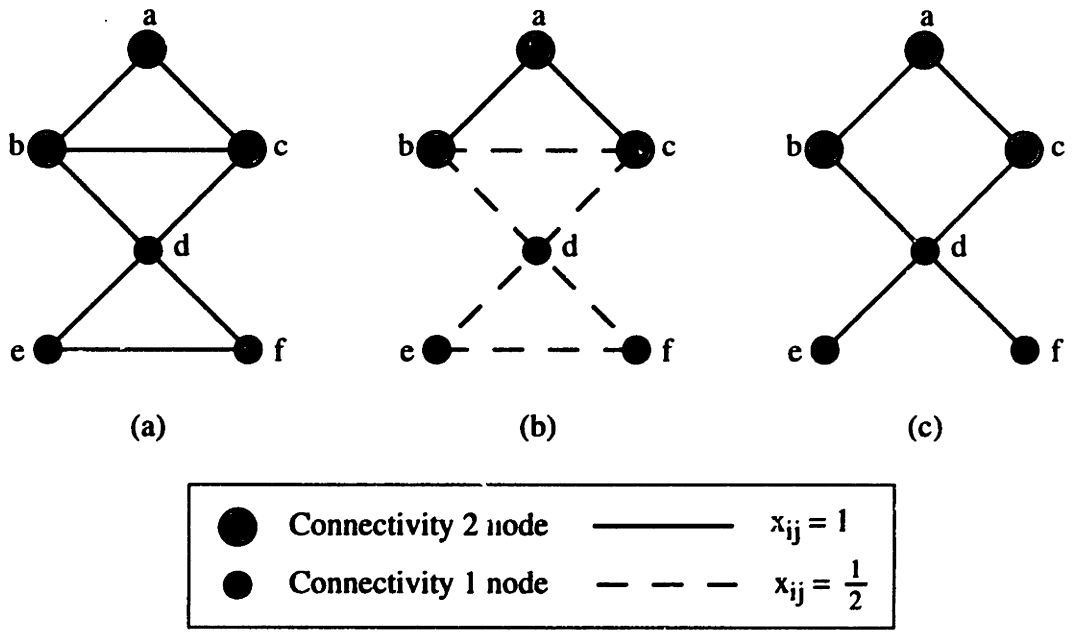


Figure 5-2: The directed cut formulation is stronger than the cutset formulation. (a) Graph with all edge costs equal to 1. (b) Optimal LP solution to the cutset formulation. (c) Optimal LP solution to the directed cut formulation.

extensions that apply to all unitary NDC problems.

5.1.2 Generalizing the Directing Procedure

Consider the directed cut formulation. Observe that the directed cut model (5.3) is not valid for unitary NDC problems with odd connectivity requirements.[2] Suppose, however, that we relax the integrality constraints imposed upon the y_{ij} variables, and interpret y_{ij} as the capacity on the flow from node i to node j. We will show that this formulation is a valid mixed integer program for any unitary NDC problem. Consider any feasible solution to an unitary NDC problem. By our earlier arguments, the solution is a connected graph consisting of 2-edge-connected components and bridges. Suppose (i) we select $y_{ij} = y_{ji} = 1/2$ for each edge on the 2-edge-connected components, and (ii) direct the bridges away from the component that contains the root node, setting y_{ij} to 1 if edge $\{i, j\}$ is oriented from node i to node j, and 0 otherwise. The resulting solution $(\mathbf{x}, \mathbf{y})$ is feasible in the directed cut formulation if we relax the integrality condition on $\mathbf{y}$. Therefore, the following directed cut formulation is valid for all unitary NDC problems.

[2]As an example, consider an SND problem defined on K_4, the complete graph on four nodes, assuming each node has a connectivity requirement of 3. The optimal solution for this problem is K_4. For any node i in K_4, there is no way to direct the edges so that both $\delta^+(i)$ and $\delta^-(i)$ are at least 1.5.

Directed cut formulation for the unitary NDC problem:

$$\text{Minimize} \quad \sum_{\{i,j\}\in E} c_{ij}x_{ij} \tag{5.4a}$$

$$\text{subject to:} \quad \sum_{(i,j)\in\delta^-(S)} y_{ij} \geq \frac{\text{econ}(S)}{2} \quad \text{if econ}(S) \geq 2\text{; for all } S, \tag{5.4b}$$

$$\sum_{(i,j)\in\delta^-(S)} y_{ij} \geq 1 \quad \text{if econ}(S) = 1\text{; for all } S, \text{ root} \notin S \tag{5.4c}$$

$$y_{ij} + y_{ji} \leq x_{ij} \quad \text{for all } \{i,j\} \in E \tag{5.4d}$$

$$x_{ij} \leq 1 \quad \text{for all } \{i,j\} \in E \tag{5.4e}$$

$$y_{ij}, y_{ji} \geq 0 \quad \text{for all } \{i,j\} \in E \tag{5.4f}$$

$$x_{ij} \geq 0 \quad \text{and integer, for all } \{i,j\} \in E. \tag{5.4g}$$

Proposition 5.1.3 *The directed cut model (5.4) is a valid formulation for the unitary NDC problem in the sense that* $(\mathbf{x},\mathbf{y})$ *is a feasible solution to this model if and only if* $\mathbf{x}$ *is an incidence vector of a feasible NDC design.*

Proof: Similar to Proposition 5.1.2. ■

Flow Formulation

The max-flow min-cut theorem permits us to formulate an improved flow model, with multiple commodities, that is equivalent to the directed cut model (5.4).

Improved undirected flow formulation for the unitary NDC problem:

$$\text{Minimize} \quad \sum_{\{i,j\}\in E} c_{ij}x_{ij} \tag{5.5a}$$

$$\text{subject to:} \quad \sum_{j\in N} f^k_{ji} - \sum_{l\in N} f^k_{il} = \begin{cases} -q_k & \text{if } i = O(k); \\ q_k & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } i \in N \text{ and } k \in K \tag{5.5b}$$

$$f^k_{ij} + f^h_{ji} \leq x_{ij} \quad \text{for all } \{i,j\} \in E \text{ and } k,h \in K \tag{5.5c}$$

$$f^k_{ij}, f^k_{ji} \geq 0 \quad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{5.5d}$$

$$x_{ij} \leq 1 \quad \text{for all } \{i,j\} \in E \tag{5.5e}$$

$$x_{ij} \geq 0 \quad \text{and integer, for all } \{i,j\} \in E. \tag{5.5f}$$

Applying the max-flow min-cut theorem directly would, in general, create a model with $O(n^2)$ flow variables since to model the constraints (5.4b), we would define a commodity for every pair of nodes $\{s,t\}$ with $r_{st} \geq 2$.

To obtain a more compact flow formulation, we first find a maximum spanning tree of the requirement graph (from now on we will refer to this tree as the *requirement spanning tree*). In Section 4.1.2, we noted that if $\{i, j\}$ is any nontree edge of the requirement graph and P_{ij} is the (unique) path between nodes i and j in the requirement spanning tree, then $r_{ij} \leq \min_{\{s,t\} \in P_{ij}} r_{st}$. We concluded that any solution that satisfies the connectivity requirements of the requirement spanning tree, satisfies all the connectivity requirements because, it contains at least $\min_{\{s,t\} \in P_{ij}} r_{st}$ paths between nodes i and j.[3]

To use the requirement spanning tree to formulate the flow model, we select any node i whose maximum connectivity requirement is greater than or equal to 2 as the root node. (If the maximum connectivity requirement of all nodes is 1, and so the problem is a Steiner tree problem, we arbitrarily select any one of the nodes as the root node.) We create the commodities as follows.

Commodity selection procedure for Formulation (5.5)

1. Find the requirement spanning tree.

2. Delete all edges with $r_{st} = 0$ from the requirement spanning tree. The resulting tree is connected because, by assumption, the NDC problem is unitary.

3. For each edge $\{s, t\}$ of the requirement spanning tree with $r_{st} \geq 2$, create two commodities: one with origin node s and destination node t, and the other with origin node t and destination node s; each of these commodities has a flow requirement of $r_{st}/2$.

4. Contract each edge $\{s, t\}$ with $r_{st} \geq 2$ in the requirement spanning tree, creating a contracted requirement spanning tree T in which $r_{ij} = 1$ for all edges $\{i, j\}$. We distinguish nodes created by the contraction from the original nodes, by calling them *components*. We denote a component by any one the nodes it contains in the original requirement spanning tree (e.g., if we create a component by contracting nodes s and t, then we denote the component s). Select a component i in T as the root node (if T does not contain any components, then select any node as the root node arbitrarily). Create a commodity for every node j in T other than the root node, with node i as its origin (in the original graph), and node/component j as its destination (in the original graph), with a requirement of 1.

Figure 5-3 illustrates this procedure. Figure 5-3a shows the requirement spanning tree of a unitary NDC problem. Figure 5-3b shows the requirement spanning tree after we

[3]This result implies the requirements of nontree edges are irrelevant, and so without loss of generality we may assume $r_{ij} = \min_{\{s,t\} \in P_{ij}} r_{st}$.

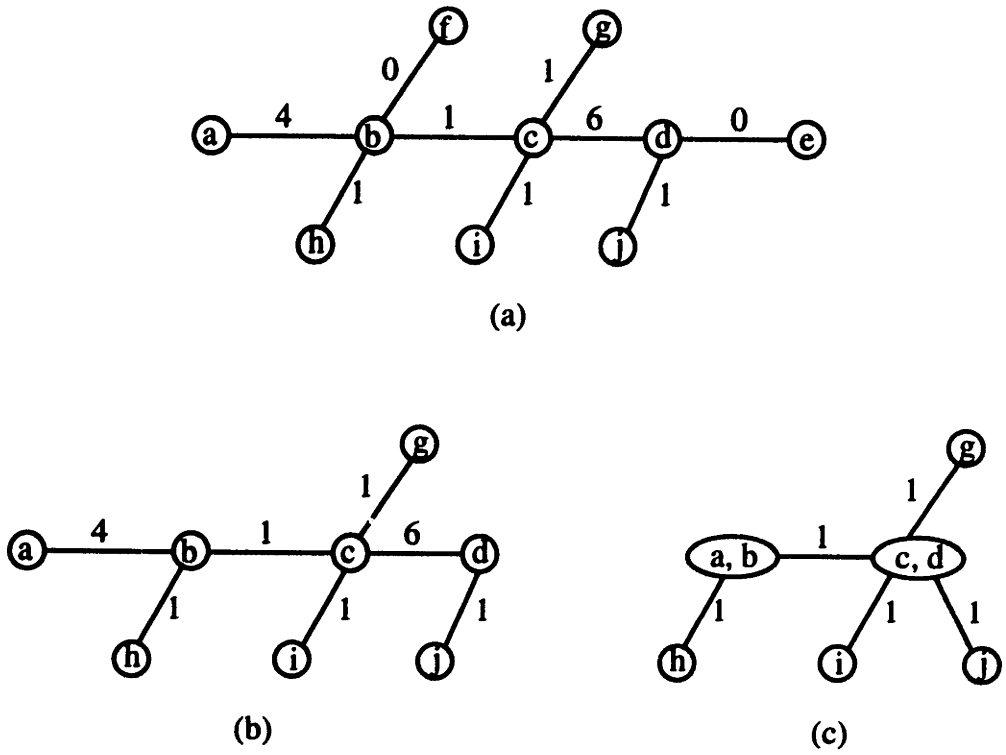


Figure 5-3: Commodity selection procedure for the unitary NDC problem. (a) Requirement spanning tree. (b) Tree obtained by deleting edges $\{s,t\}$ with $r_{st} = 0$. (c) Tree obtained by contracting edges $\{s,t\}$ with $r_{st} \geq 2$.

Commodity Origin	Commodity Destination	Commodity Requirement
a	b	2
b	a	2
c	d	3
d	c	3
a	c	1
a	g	1
a	h	1
a	i	1
a	j	1

Table 5.1: Commodities in Formulation (5.5) for example in Figure 5-3.

have deleted edges $\{s,t\}$ with $r_{st} = 0$. Notice that because the problem is a unitary NDC problem the graph in Figure 5-3b is connected. Otherwise, it would be a forest. Edges $\{a,b\}$ and $\{c,d\}$ are the only edges with requirement at least 2 on this tree. Therefore, we can orient every feasible solution so that it contains at least 2 directed paths from node b to node a, at least 2 directed paths from node a to node b, at least 3 directed paths from node c to node d, and at least 3 directed paths from node d to node c. Figure 5-3c shows the requirement spanning tree after we have contracted all edges $\{s,t\}$ with $r_{st} \geq 2$. Since node a is the root node, the directing procedure implies that we can orient every feasible solution so that it contains a directed path from component $\{a,b\}$ (i.e., from both nodes a and b) to component $\{c,d\}$ (i.e., to both nodes c and d), and to nodes g, h, i and j. (Component $\{a,b\}$ might or might not be in the same 2-edge connected component as any one of the nodes/component $\{c,d\}$, g, h, i and j. The directing procedure implies that in both cases, the network contains a directed path from $\{a,b\}$ to these nodes.)

Table 5.1 identifies the commodities obtained by applying the directing procedure to the example in Figure 5-3. Edges $\{a,b\}$ and $\{c,d\}$ are the only edges with a requirement of at least 2 on this tree. Therefore, we create four commodities—those shown in the first four rows of Table 5.1. Figure 5-3c shows the contracted requirement spanning tree (i.e., after contracting edges with $r_{st} \geq 2$). Let node a denote the component $\{a,b\}$, and node c denote component $\{c,d\}$. We select node a as the root. The tree in Figure 5-3c contains 6 nodes. Therefore, we create 5 commodities, each with origin node a, and destination nodes c, g, h, i and j.

The following useful property is a consequence of the commodity selection procedure.

Property 5.1.4 *For any node set S,*

1. *If $\mathit{econ}(S) \geq 2$, then the improved flow formulation contains a commodity k whose flow requirement is $\mathit{econ}(S)/2$, origin is in $N \backslash S$, and destination is in S.*

2. *If* $econ(S) = 1$ *and root* $\notin S$, *then the improved flow model contains a commodity whose flow requirement is 1, origin is the root node, and destination is in* S.

3. *If* $econ(S) = 1$ *and root* $\in S$, *then no commodity in the improved flow model has its origin in* $N \backslash S$, *and destination in* S.

Proof: This result follows from the commodity selection procedure and the fact that for any edge $\{s,t\}$ in the requirement spanning tree $r_{st} = \max\{r_{ij} | j \in S; i \in N \backslash S\} \equiv \text{econ}(S)$. ■

We now establish the validity of the improved undirected flow formulation for the unitary NDC problem by showing that the improved undirected flow formulation and the directed cut formulation are equivalent.

Lemma 5.1.5 *The improved undirected flow formulation (5.5) and the directed cut formulation (5.4) are equivalent, both as linear programs and mixed integer programs.*

Proof: We assume that we select the same root node in both formulations. First, consider any feasible solution $(\mathbf{x}^*, \mathbf{y}^*)$ to the directed cut formulation. If we interpret y^*_{ij} as a capacity imposed upon the flow from node i to node j, the max-flow min-cut theorem implies that we can (i) send $r_{st}/2$ units of flow between any pair of nodes s and t in the requirement spanning tree, with $r_{st} \geq 2$, and (ii) send one unit of flow from the root component in the contracted requirement spanning tree to any other node/component in the contracted requirement spanning tree. Furthermore, the constraint $y^*_{ij} + y^*_{ji} \leq x^*_{ij}$ implies that we can fulfill conditions (i) and (ii) while ensuring that for each edge $\{i,j\}$, the sum of the maximum flow sent (on the edge $\{i,j\}$) from node i to node j, and the maximum flow sent from node j to node i does not exceed x_{ij}.[4] These arguments show that we can find flow variables $\mathbf{f}^*$ so that $(\mathbf{x}^*, \mathbf{f}^*)$ is feasible in the flow formulation.

Suppose $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$ is a feasible solution to the improved flow formulation. For each edge $\{i,j\}$, set $\bar{y}_{ij} = \max_{k \in K} \bar{f}^k_{ij}$ and $\bar{y}_{ji} = \max_{k \in K} \bar{f}^k_{ji}$. We claim the solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is feasible in the directed cut formulation. Whenever edge $\{s,t\}$ is in the requirement spanning tree and $r_{st} \geq 2$, the improved undirected flow formulation sends $r_{st}/2$ units of flow from node s to node t and $r_{st}/2$ units of flow from node t to node s. Consequently, if edge $\{s,t\}$ is in the requirement spanning tree and $r_{st} \geq 2$, the capacity of every s–t dicut and every t–s dicut is at least $r_{st}/2$ (for the solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$). For any node set S, the requirement spanning tree contains an edge $\{s,t\}$ in $\delta(S)$ with $\text{econ}(S) = r_{st}$. Therefore, whenever $\text{econ}(S) \geq 2$, the capacity of the dicut $\delta^-(S)$ is at least $\text{econ}(S)/2$. The improved undirected flow formulation sends 1 unit of flow from the root component to every node/component in the contracted

[4] If $r_{st} \in \{0, 1, \text{even}\}$, Nash-Williams' theorem permits us to send flow in only direction on an edge. In general, this is not the case.

requirement spanning tree. Therefore, the capacity of every dicut $\delta^-(S)$ with root $\notin S$ and $\text{econ}(S) = 1$ is at least 1. The constraint $\bar{f}_{ij}^k + \bar{f}_{ji}^h \leq \bar{x}_{ij}$ implies that for every edge, $\bar{y}_{ij} + \bar{y}_{ji} \leq \bar{x}_{ij}$. Consequently, $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is feasible for the directed cut model, and thus the improved undirected flow formulation and the directed cut formulation are equivalent. ■

Before concluding this section, we note the improved models (5.4) and (5.5) are stronger, as linear programs, than the cutset (5.1) and the undirected flow (5.2) model only if the requirement spanning tree contains an edge $\{s,t\}$ with $r_{st} = 1$. To see this result, observe that if no pair of nodes i and j has a connectivity requirement of 1, then for all node sets S, $\text{econ}(S) \neq 1$. But then, if $\bar{\mathbf{x}}$ is any feasible solution to the cutset formulation, the vector $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, with $\bar{y}_{ij} = \bar{y}_{ji} = \bar{x}_{ij}/2$, is feasible in the directed cut formulation. As we have shown before, if $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is any feasible solution to the directed cut formulation, then $\bar{\mathbf{x}}$ is feasible in the cutset formulation. Therefore, in this case, the two models are equivalent.

5.1.3 Comparison with the Enhanced Cutset Model

Balakrishnan, Magnanti and Mirchandani [BMM94a] introduced the following enhanced cutset model (our terminology) for the edge-connectivity version of the NDC problem: this model resembles the directed cut formulation (5.4).

Enhanced cutset formulation for the NDC problem:

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{5.6a}$$

$$\text{subject to:} \quad \sum_{\{i,j\} \in \delta(S)} y_{ij} \geq \frac{\text{econ}(S)}{2} \quad \text{if } \text{econ}(S) \geq 2; \tag{5.6b}$$

$$\sum_{\{i,j\} \in \delta(S)} z_{ij} \geq \frac{\text{econ}(S)}{2} \quad \text{if } \text{econ}(S) \geq 2; \tag{5.6c}$$

$$\sum_{\{i,j\} \in \delta(S)} y_{ij} \geq 1 \quad \text{if } \text{econ}(S) = 1; \tag{5.6d}$$

$$y_{ij} + z_{ij} = x_{ij} \quad \text{for all } \{i,j\} \in E \tag{5.6e}$$

$$x_{ij} \leq 1 \quad \text{for all } \{i,j\} \in E \tag{5.6f}$$

$$x_{ij} \geq 0 \quad \text{and integer, for all } \{i,j\} \in E. \tag{5.6g}$$

$$y_{ij}, z_{ij} \geq 0 \quad \text{for all } \{i,j\} \in E. \tag{5.6h}$$

This formulation introduces additional edge variables y_{ij} and z_{ij}. It "splits" the connectivity requirements of all sets with $\text{econ}(S) \geq 2$, and ensures that the y_{ij} and z_{ij} variables satisfy these connectivity requirements (the model permits fractional values for y_{ij} and z_{ij}). In addition, the y_{ij} variables satisfy the connectivity requirements of those node sets S with

$\text{econ}(S)$ equal to 1. It is easy to see that this formulation is valid (since $x_{ij} = y_{ij} + z_{ij}$, the x_{ij} variables satisfy the connectivity constraints).

If the problem contains no node set S with $\text{econ}(S) = 1$, the enhanced cutset formulation does not strengthen the cutset formulation (5.1). The x_{ij} variables of the enhanced cutset formulation are feasible for the cutset formulation, and we can convert a feasible solution to the cutset formulation to a feasible solution to the enhanced cutset formulation by setting $y_{ij} = z_{ij} = x_{ij}/2$ (with no change in the x_{ij} variables). If the problem contains no node sets with $\text{econ}(S) \geq 2$, then the z_{ij} variables can be set to zero. Therefore, $y_{ij} = x_{ij}$ for all edges, and the formulation is identical to the cutset model. Consequently, the enhanced cutset formulation does not strengthen the cutset model if $\text{econ}(S)$ equals zero or one for all node sets S.

The enhanced cutset formulation is stronger than the cutset formulation when the problem contains some node sets S with $\text{econ}(S) = 1$ and some node sets S with $\text{econ}(S) \geq 2$ (see Figure 5-4). In this SND example, each edge has unit cost. Nodes a, b and c have a connectivity requirement of 2. Nodes d, e, f, g, h and i have a connectivity requirement of 1. The optimal solution to the cutset formulation, which sets all edge variables to 0.5, has an objective value of 6. The LP relaxation of the enhanced cutset formulation has the solution $y_{ad} = y_{db} = y_{bf} = y_{fc} = y_{ce} = y_{ea} = y_{ag} = y_{gb} = y_{bi} = y_{ic} = y_{ch} = y_{ha} = z_{ad} = z_{db} = z_{bf} = z_{fc} = z_{ce} = z_{ea} = 0.5$ (Figure 5-4c shows the $\mathbf{x}$ variables), with an objective value of 9 (which is the objective value of the optimal integer solution).

For the SND problem (and in general the unitary NDC problem), the improved undirected flow formulation can be stronger than the enhanced cutset formulation. Figure 5-2 provides an example. The optimal solution of the enhanced cutset formulation is $y_{ab} = y_{ac} = y_{bd} = y_{cd} = y_{de} = y_{df} = y_{ef} = 0.5$, $z_{ab} = z_{bc} = z_{ac} = 0.5$, and $x_{ab} = x_{ac} = 1$, $x_{bc} = x_{bd} = x_{cd} = x_{de} = x_{df} = x_{ef} = 0.5$, and the objective value is 5. In contrast, the improved undirected flow formulation solves the problem optimally.

It is an open question whether for some instance of the unitary NDC problem, the enhanced cutset formulation is stronger than the improved undirected flow formulation (5.5). We conjecture that this is not the case, and so the improved undirected flow formulation (5.5) is stronger than the enhanced cutset formulation (5.6).

Summary

In this section we first showed how to direct unitary NDC problems when the connectivity requirements r_{st} are even or one. We then showed how to generalize this directing procedure to all unitary NDC problems. We used this directing procedure to develop two improved models, a flow-based model and a directed cut model, for the unitary NDC problem. Finally, we compared these improved models with the enhanced cutset model of Balakrishnan, Mag-

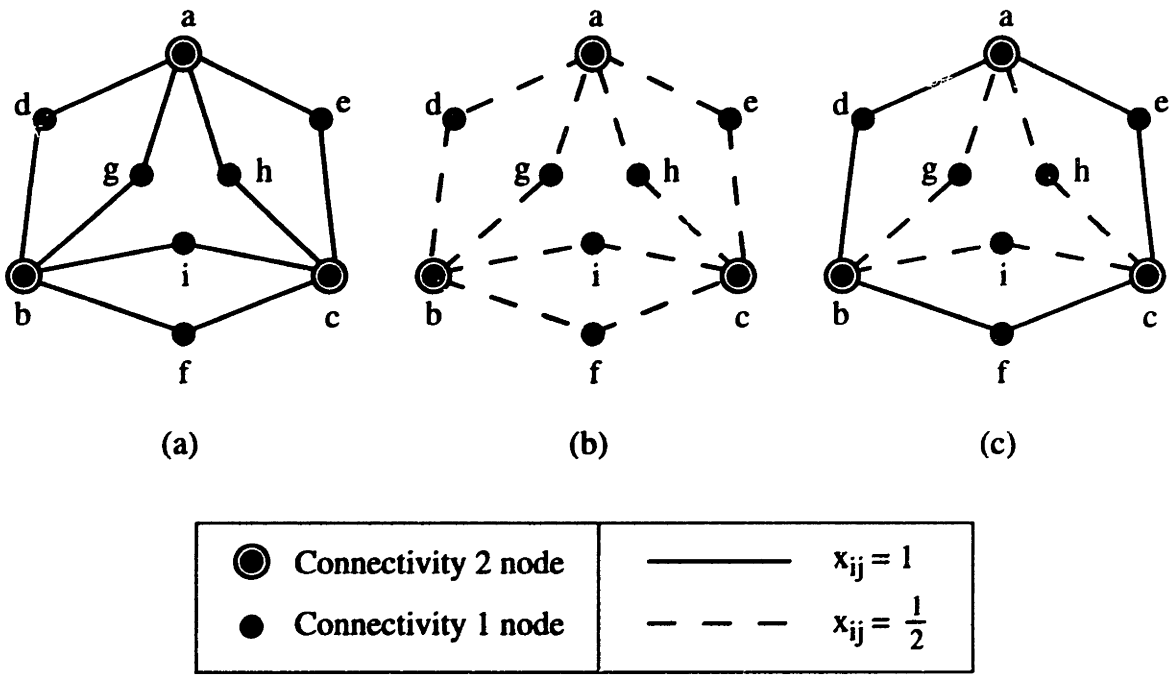


Figure 5-4: The enhanced cutset formulation is stronger than the cutset formulation. (a) Graph with all edge costs equal to 1. (b) Optimal LP solution to the cutset formulation. (c) Optimal LP solution to the enhanced cutset formulation.

nanti and Mirchandani [BMM94a], and showed that the improved models can be stronger than the enhanced cutset model.

To conclude this section, we note that a simple modification of the formulations we have considered permits us to model situations that allow edge replication: we just replace the constraint $x_{ij} \leq 1$ by the constraint $x_{ij} \leq b_{ij}$ throughout our discussion.

5.2 Projecting from the Improved Formulation

In the previous section we described an improved flow-based model (5.5) for the unitary NDC problem. In this section we investigate the relationship between this formulation and the cutset model for the unitary NDC problem (5.1). More specifically, we project out the flow variables from the improved undirected flow formulation (5.5) to obtain several valid inequalities for the cutset model.

5.2.1 The Projection Theorem

To compare the improved flow formulation and the cutset formulation, we would like to project out the flow variables from the improved flow formulation so that the resulting models have the same set of variables. One approach is to use Fourier-Motzkin elimination:

project out a single flow variable at a time. A more elegant method, proposed by Balas and Pulleyblank [BP83], is based upon a theorem of the alternatives.

Theorem 5.2.1 (Balas-Pulleyblank) *The projection of the set* $P = \{(\mathbf{x}, \mathbf{f}) \in \mathcal{R}^{n+m} | \mathbf{Ax} + \mathbf{Bf} \geq \mathbf{d}\}$ *onto the space of the* x *variables is*

$$Proj_{\mathbf{x}}(P) = \{\mathbf{x} \in \mathcal{R}^n | (\mathbf{g}^j)^T \mathbf{Ax} \geq (\mathbf{g}^j)^T \mathbf{d}, \quad for\, j = 1, 2, \ldots, J\},$$

which is defined by a finite set of generators $\{\mathbf{g}^j | j = 1, \ldots, J\}$ *of the cone* $C = \{\mathbf{g} | \mathbf{B}^T \mathbf{g} = \mathbf{0}; \mathbf{g} \geq 0\}$.

Proof: We first show that $\text{Proj}_{\mathbf{x}}(P) \subseteq \{\mathbf{x} \in \mathcal{R}^n | (\mathbf{g}^j)^T \mathbf{Ax} \geq (\mathbf{g}^j)^T \mathbf{d}, \text{ for } j = 1, 2, \ldots, J\}$. Let $\bar{\mathbf{x}} \in \text{Proj}_{\mathbf{x}}(P)$. By definition, for some vector $\bar{\mathbf{f}} \in \mathcal{R}^m$, $(\bar{\mathbf{x}}, \bar{\mathbf{f}}) \in P$. Therefore,

$$\mathbf{A}\bar{\mathbf{x}} + \mathbf{B}\bar{\mathbf{f}} \geq \mathbf{d}.$$

Consider any element $\mathbf{g} \in C$. Multiplying both sides of this equation by $\mathbf{g}^T$, we obtain

$$\mathbf{g}^T \mathbf{A}\bar{\mathbf{x}} + \mathbf{g}^T \mathbf{B}\bar{\mathbf{f}} \geq \mathbf{g}^T \mathbf{d}.$$

Noting that for any member of the cone C, $\mathbf{B}^T \mathbf{g} = \mathbf{0}$, we see that

$$\mathbf{g}^T \mathbf{A}\bar{\mathbf{x}} \geq \mathbf{g}^T \mathbf{d}.$$

Since this inequality is valid for all members of the cone C, it is also true for the generators of the cone. Thus,

$$\bar{\mathbf{x}} \in \{\mathbf{x} \in \mathcal{R}^n | (\mathbf{g}^j)^T \mathbf{Ax} \geq (\mathbf{g}^j)^T \mathbf{d}, \text{ for } j = 1, 2, \ldots, J\}.$$

Now we show that $\{\mathbf{x} \in \mathcal{R}^n | (\mathbf{g}^j)^T \mathbf{Ax} \geq (\mathbf{g}^j)^T \mathbf{d}, \text{ for } j = 1, 2, \ldots, J\} \subseteq \text{Proj}_{\mathbf{x}}(P)$. Let $\hat{\mathbf{x}} \in \{\mathbf{x} \in \mathcal{R}^n | (\mathbf{g}^j)^T \mathbf{Ax} \geq (\mathbf{g}^j)^T \mathbf{d}, \text{ for } j = 1, 2, \ldots, J\}$. Then no element $\mathbf{g} \in C$ satisfies the condition

$$\mathbf{B}^T \mathbf{g} = 0, \quad \mathbf{g} \geq \mathbf{0}, \quad \mathbf{g}^T (\mathbf{d} - \mathbf{A}\hat{\mathbf{x}}) > 0.$$

By Farkas' lemma, this result implies some vector $\hat{\mathbf{f}} \in \mathcal{R}^m$ satisfies the condition

$$\mathbf{B}\hat{\mathbf{f}} \geq (\mathbf{d} - \mathbf{A}\hat{\mathbf{x}}).$$

But then $\hat{\mathbf{x}} \in \text{Proj}_{\mathbf{x}}(P)$. ■

We can similarly establish the following result.

Corollary 5.2.2 *The projection of the set* $P = \{(\mathbf{x},\mathbf{f}) \in \mathcal{R}^{n+m} | \mathbf{Ax} + \mathbf{Bf} \geq \mathbf{d}, \mathbf{x} \in \mathbf{X}\}$, *where* $\mathbf{X}$ *is an arbitrary subset of* $\mathcal{R}^n$, *onto the space of the* x *variables is*

$$Proj_{\mathbf{x}}(P) = \{\mathbf{x} \in \mathbf{X} | (\mathbf{g}^j)^T \mathbf{Ax} \geq (\mathbf{g}^j)^T \mathbf{d}, \quad for\, j = 1, 2, \ldots, J\},$$

which is defined by a finite set of generators $\{\mathbf{g}^j | j = 1, \ldots, J\}$ *of the cone* $C = \{\mathbf{g} | \mathbf{B}^T\mathbf{g} = \mathbf{0}; \mathbf{g} \geq 0\}$.

Proof: Similar to Theorem 5.2.1. ■

The cone C in the statement of Theorem 5.2.1 is just the linear programming dual to the feasibility problem obtained by deleting the $\mathbf{x}$ variables and setting the righthand side to zero. Consequently, there are many ways to express Theorem 5.2.1. For the improved undirected formulation, it is more convenient to consider the following form of Theorem 5.2.1.

Corollary 5.2.3 *The projection of the set* $P = \{(\mathbf{x},\mathbf{f}) \in \mathcal{R}^{n+m} | \mathbf{Ax} + \mathbf{Bf} \geq \mathbf{d}; \mathbf{A'x} + \mathbf{B'f} = \mathbf{d'}; \mathbf{x} \in \mathbf{X}; \mathbf{f} \geq \mathbf{0}\}$ *onto the space of the* x *variables is*

$$Proj_{\mathbf{x}}(P) = \{\mathbf{x} \in \mathbf{X} | (\mathbf{g}_u^j)^T \mathbf{Ax} + (\mathbf{g}_v^j)^T \mathbf{A'x} \geq (\mathbf{g}_u^j)^T \mathbf{d} + (\mathbf{g}_v^j)^T \mathbf{d'}, \quad for\, j = 1, 2, \ldots, J\},$$

which is defined by a finite set of generators $\{(\mathbf{g}_u^j, \mathbf{g}_v^j) | j = 1, \ldots, J\}$ *of the cone* $C = \{(\mathbf{u},\mathbf{v}) | \mathbf{B}^T\mathbf{u} + \mathbf{B'}^T\mathbf{v} \geq \mathbf{0}; \mathbf{u} \geq 0; \mathbf{v}$ *unrestricted*$\}$.

If we can identify a set of finite generators of the cone C, then we obtain the projection of the set P. The Balas-Pulleyblank theorem has the additional advantage that every member $(\mathbf{u},\mathbf{v})$ of the cone C defines a valid inequality $(\mathbf{u}^T A + \mathbf{v}^T A')\mathbf{x} \geq \mathbf{u}^T\mathbf{d} + \mathbf{v}^T\mathbf{d'})$ for $\text{Proj}_{\mathbf{x}}(P)$. As a consequence, even if we cannot characterize the generators of the cone, we can still use the cone to obtain valid inequalities for $\text{Proj}_{\mathbf{x}}(P)$.

5.2.2 Projection Cone of the Improved Undirected Flow Formulation (5.5)

As noted earlier, the cone C in the statement of Theorem 5.2.1 is the LP dual to the feasibility problem obtained by deleting the $\mathbf{x}$ variables and setting the righthand side to zero. For the improved flow formulation (5.5), we need to consider the following projection cone:

$$\left.\begin{array}{l} \sum_{h \in K} u_{ij}^{kh} + v_i^k - v_j^k \\ \sum_{h \in K} u_{ij}^{hk} + v_j^k - v_i^k \end{array}\right\} \geq 0 \quad \forall \{i,j\} \in E, \ \forall k \in K \tag{5.7a}$$

$$u_{ij}^{kh} \geq 0 \quad \forall \{i,j\} \in E, \ \forall k, h \in K. \tag{5.7b}$$

In these inequalities, v_i^k is the dual variable corresponding to the flow balance equation at node i for commodity k, and u_{ij}^{kh} is the dual variable corresponding to the forcing constraint

$f_{ij}^k + f_{ji}^h \leq x_{ij}$. Note that for any edge $\{i,j\}$ and any pair of commodities k and h, the model contains two forcing constraints $f_{ij}^k + f_{ji}^h \leq x_{ij}$ and $f_{ij}^h + f_{ji}^k \leq x_{ij}$. We identify the dual variable u_{ij}^{kh} with the constraint $f_{ij}^k + f_{ji}^h \leq x_{ij}$ and the dual variable u_{ij}^{hk} with the constraint $f_{ij}^h + f_{ji}^k \leq x_{ij}$. By convention, the dual variables obtained by reversing the indices, i.e. u_{ij}^{kh} and u_{ji}^{hk}, are the same.

Since one flow balance equation for each commodity is redundant, we can set, for each commodity k, $v_{O(k)}^k$ to value zero. Using Corollary 5.2.3 for any member of this cone, we obtain a valid inequality of the form

$$\sum_{\{i,j\}\in E}(\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh})x_{ij} \geq \sum_{k\in K} q_k v_{D(k)}^k. \tag{5.8}$$

In this expression, q_k is the number of units of commodity k sent from commodity k's origin to its destination. We refer to the coefficient of x_{ij} in this inequality as π_{ij} and the righthand side coefficient as π_0.

Given some choice of the variables v_i^k for all the nodes i and commodities k, there are a number of choices for u_{ij}^{hk}. How do we determine the best such choice? Since the coefficient π_{ij} of x_{ij} in the inequality $\pi \mathbf{x} \geq \pi_0$ is $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh}$, we would like $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh}$ to be as small as possible for each edge. The following theorem describes the choice of u_{ij}^{hk} that minimizes $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh}$. We give a constructive proof that also shows how to determine the u_{ij}^{hk} values.

Theorem 5.2.4 *Suppose we are given values for v_i^k for all nodes i and all commodities k. For any edge $\{i,j\}$, let $t_{ij}^k = \max(0, v_j^k - v_i^k)$ and $t_{ji}^k = \max(0, v_i^k - v_j^k)$. Define $t_{ij} = \sum_{k\in K} t_{ij}^k$ and $t_{ji} = \sum_{k\in K} t_{ji}^k$. Then $\max(t_{ij}, t_{ji})$ is the minimum feasible value of $\sum_{k\in K}\sum_{h\in k} u_{ij}^{kh}$ in the inequalities (5.7a).*

Proof: We will establish this result for each edge $\{i,j\}$. Let us first show that $\max(t_{ij}, t_{ji})$ is a lower bound on the value of $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh}$ in any feasible solution to the inequalities (5.7a). Let I be the set of all commodities with $t_{ij}^k > 0$ and J be the set of commodities with $t_{ji}^k > 0$. For any edge $\{i,j\}$ with $v_j^k - v_i^k > 0$, equation (5.7a) implies $\sum_{h\in K} u_{ij}^{kh} \geq v_j^k - v_i^k$. Summing over all commodities in I gives $\sum_{k\in I}\sum_{h\in K} u_{ij}^{kh} \geq \sum_{k\in I}(v_j^k - v_i^k) = t_{ij}$. Similarly, by considering the commodities in J, we obtain $\sum_{k\in J}\sum_{h\in K} u_{ij}^{hk} \geq \sum_{k\in J}(v_i^k - v_j^k) = t_{ji}$. But then the inequalities $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh} \geq \sum_{k\in I}\sum_{h\in K} u_{ij}^{kh}$ and $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh} \geq \sum_{k\in J}\sum_{h\in K} u_{ij}^{hk}$ imply that $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh} \geq \max(t_{ij}, t_{ji})$.

We next prescribe values for the variables u_{ij}^{kh} that achieve the lower bound $\max(t_{ij}, t_{ji})$. Initially, each $u_{ij}^{kh} = 0$ and $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh} = 0$. Select a commodity l from I and a commodity m from J. Set $u_{ij}^{lm} = \min\{t_{ij}^l, t_{ji}^m\}$. If $t_{ij}^l \geq t_{ji}^m$, delete m from J, and if $t_{ij}^l \leq t_{ji}^m$, delete l from I. Set $t_{ij}^l = t_{ij}^k - u_{ij}^{lm}$ and $t_{ji}^m = t_{ji}^m - u_{ij}^{lm}$. Repeat this procedure until one of the two sets I and J, say J, is empty. Note that at this point $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh} = \min(t_{ij}, t_{ji})$ and

the **u** and **v** variables satisfy inequality (5.7a) for every commodity we have deleted from I and J. For the remaining commodities $l \in I$, let m be any commodity in K and set $u_{ij}^{lm} = t_{ij}^{l}$. Thus, $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh} = \min(t_{ij}, t_{ji}) + (\max(t_{ij}, t_{ji}) - \min(t_{ij}, t_{ji})) = \max(t_{ij}, t_{ji})$. By construction, this choice of u_{ij}^{hk} satisfies the equations of the cone. ∎

With the aid of Theorem 5.2.4, we will now derive three classes of valid inequalities—partition inequalities, odd-hole inequalities, and combinatorial design inequalities. For ease of exposition, we will consider the NDLC problem. Then we will generalize the results to the unitary NDC problem.

5.3 Partition Inequalities

For the NDLC problem, partition inequalities (also called multicut inequalities by some authors) have the following form. Partition the node set N into disjoint node sets $N_0, N_1, \ldots, N_p$ satisfying the property that each node set has at least one node i with a connectivity requirement $r_i > 0$. A partition inequality is an inequality of the form

$$\frac{1}{2}\sum_{k=0}^{k=p}\sum_{\delta(N_k)} x_{ij} \geq \begin{cases} p+1 & \text{if at least two sets have a node with} \\ & \text{a connectivity requirement of two;} \\ p & \text{otherwise.} \end{cases}$$

This inequality implies that (i) if no set or exactly one set in the partition has a node with a connectivity requirement of two, then the network contains at least p edges between the $p+1$ sets, and (ii) if two or more sets have a node with a connectivity requirement of two, then the network contains at least $p+1$ edges between these sets. Chopra [Cho92b] shows that the partition inequalities describe the dominant of the spanning tree polytope. Chopra and Rao [CR94] and Grötschel and Monma [GM90] show that under appropriate conditions, partition inequalities are facet defining for the Steiner tree problem and for the NDLC problem.

Consider the five node example shown in Figure 5-5. Nodes 0 and 1 have a connectivity requirement of two; nodes 2, 3 and 4 have a connectivity requirement of 1. The improved undirected formulation (with node 0 as the root node) for this network has four commodities (1, 2, 3 and 4) with the origin as the root node and node 1, 2, 3 and 4, respectively, as the destination node. It also has a commodity (commodity 5) with node 1 as the origin and node 0 as the destination. We will refer to commodity 5 as commodity 0 to simplify our discussion. Therefore, for each commodity i, node i is the destination. Consider the partition inequality with each node defining a set in the partition (i.e., $\frac{1}{2}\sum_i\sum_{j\neq i} x_{ij} \geq 5$).

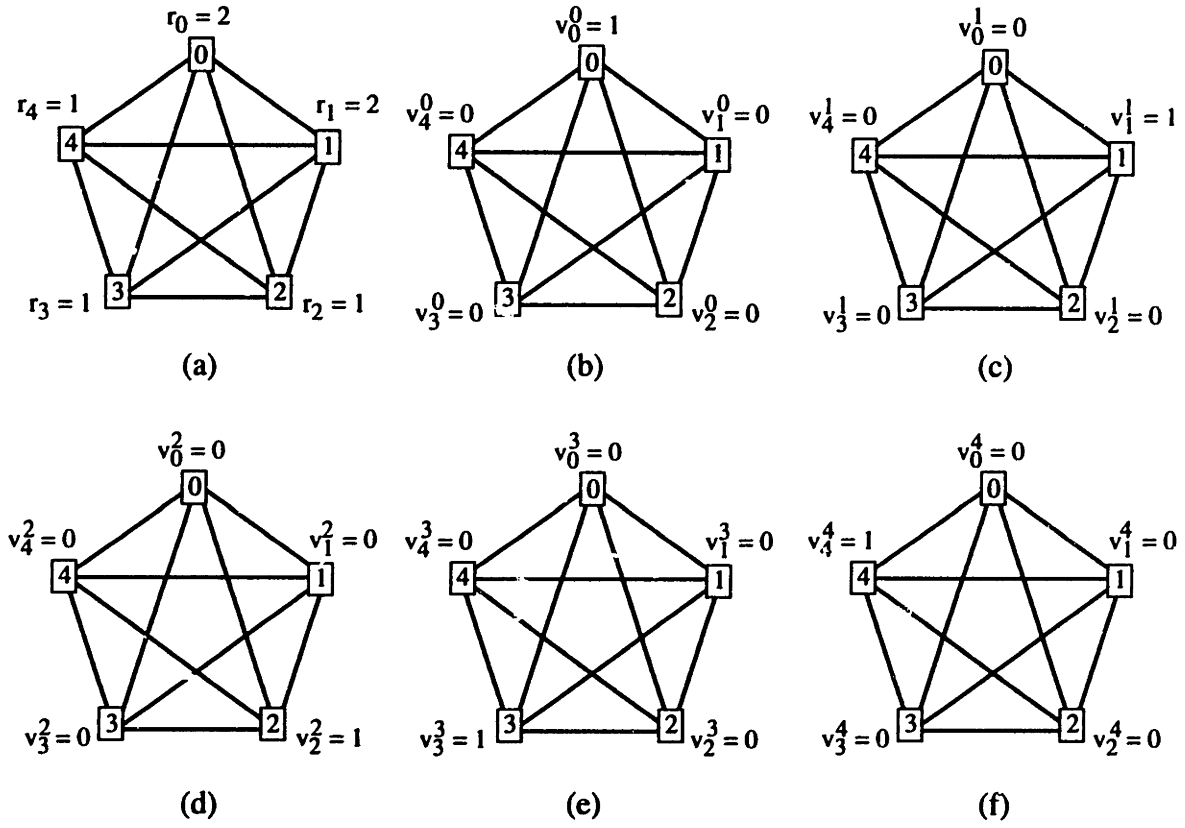


Figure 5-5: Projecting partition inequalities for a five node example. (a) Connectivity requirements (b) Value of the v_i^0 variables (c) Value of the v_i^1 variables (d) Value of the v_i^2 variables (e) Value of the v_i^3 variables (f) Value of the v_i^4 variables.

Each edge variable[5] in the partition inequality has a coefficient of 1. Consequently, in projecting out these variables to obtain this inequality, we need $\sum_{k\in K}\sum_{h\in K} u_{ij}^{kh} = 1$. Since the righthand side has value 5, we also need $\sum_{k\in K} v_{D(k)}^k = 5$ ($q_k = 1$ for all commodities in the NDLC problem).

In the example of Figure 5-5, for each commodity $k = 0,\ldots,4$, let $v_i^k = 0$ if $i \neq k$ and $v_k^k = 1$. Consider any edge $\{i,j\}$. Note that the v_i^k and v_j^k values are the same for all commodities, except for (i) commodity i, with $v_i^i = 1$ and $v_j^i = 0$, and (ii) commodity j, with $v_i^j = 0$ and $v_j^j = 1$. Thus, for each edge $\{i,j\}$, $\sum_{k\in K} \max(0, v_j^k - v_i^k) = \sum_{k\in K} \max(0, v_i^k - v_j^k) = 1$. Therefore, in accordance with Theorem 5.2.4 we can set $u_{ij}^{hk} = 1$ if $h = j$ and $k = i$, and $u_{ij}^{hk} = 0$ otherwise. This choice of variables in the projection cone imply the partition $\frac{1}{2}\sum_i\sum_{j\neq i} x_{ij} \geq 5$.

In general, consider a partition $N_0,\ldots,N_p$. Without loss of generality assume that

[5]Note that the inequality counts each edge twice in the double summation.

the root node is a node in N_0. Suppose $\text{econ}(N_i) = 2$. Recall, from Property 5.1.4, if $\text{econ}(N_i) = 2$ the improved flow formulation must contain two commodities: one with origin $n_i \in N_i$ and destination some node $m_i \notin N_i$, and one with destination $n_i \in N_i$ and origin $m_i \notin N_i$, both with a flow requirement of $\text{econ}(N_i)/2 = 1$. Let i denote the commodity with destination node $n_i \in N_i$ (and origin $m_i \notin N_i$) and a requirement of 1. Similarly, if $\text{econ}(N_i) = 1$ and $i \neq 0$, the improved flow formulation must contain a commodity i with destination a node $n_i \in N_i$ (the origin would be the root node) and a requirement of 1.

For commodities $0, \ldots, p$, we set

$$v_i^k = \begin{cases} 1 & \text{if } i \in N_k, \\ 0 & \text{otherwise,} \end{cases}$$

and set $v_i^k = 0$ otherwise.

With this choice of values for the v_i^k variables, we ensure that for all edges $\{i, j\}$ across the partition $\sum_{k \in K} \max(0, v_j^k - v_i^k) = \sum_{k \in K} \max(0, v_i^k - v_j^k) = 1$ and for edges $\{i, j\}$ not in the partition $\sum_{k \in K} \max(0, v_j^k - v_i^k) = \sum_{k \in K} \max(0, v_i^k - v_j^k) = 0$. Thus, by choosing the values of u_{ij}^{kh} as indicated by Theorem 5.2.4, we find that $\pi_{ij} = \sum_{k \in K} \sum_{h \in K} u_{ij}^{kh}$ is 1 if i and j are in different sets of the partition and $\pi_{ij} = 0$ otherwise. Also, $\pi_0 = \sum_{k \in K} v_{D(k)}^k = p + 1$, since $v_{D(k)}^k = 1$ for commodities $0, \ldots, p$ and $v_{D(k)}^k = 0$ otherwise.

If all the nodes with a connectivity requirement of two lie in one of the sets (i.e., N_0) of the partition or the problem has no node with a connectivity requirement of two (in this case the problem is a Steiner tree problem), by Property 5.1.4, there is no commodity in the improved flow formulation with destination in N_0. Consequently, we select p commodities with destinations in $N_1, \ldots, N_p$ and origin in N_0 (i.e., the root node). With the same choice of v_i^k for $k = 1, \ldots, p$ (i.e., $v_i^k = 1$ if $i \in N_k$ and 0 otherwise), for all edges $\{i, j\}$ across the partition, $\max(\sum_{k \in K} \max(0, v_j^k - v_i^k), \sum_{k \in K} \max(0, v_i^k - v_j^k)) = 1$; and for edges $\{i, j\}$ not in the partition, $\sum_{k \in K} \max(0, v_j^k - v_i^k) = \sum_{k \in K} \max(0, v_i^k - v_j^k) = 0$. Thus, in accordance with Theorem 5.2.4 the edges across the partition have coefficient 1; all others have coefficient 0. Since we have only p commodities, the resulting righthand side is $\pi_0 = \sum_k v_{D(k)}^k = p$.

5.3.1 Partition Inequalities for the Unitary NDC Problem

Since the projection cone obtained from the improved flow model for the NDLC problem and the unitary NDC problem are identical, any member of the cone that provides a valid inequality for the NDLC problem also provides a valid inequality for the unitary NDC problem.

Consider a partition $N_0, \ldots, N_p$. Without loss of generality assume that the root node is a node in N_0. Consider any set N_i of the partition. If $\text{econ}(N_i) \geq 2$, Property 5.1.4 implies the improved flow formulation must contain two commodities: one with origin

$n_i \in N_i$ and destination some node $m_i \notin N_i$, and one with destination $n_i \in N_i$ and origin $m_i \notin N_i$, both with a flow requirement of $\text{econ}(N_i)/2$. Let i denote the commodity with destination node $n_i \in N_i$ (and origin $m_i \notin N_i$) and a requirement of $\text{econ}(N_i)/2$. Similarly, if $\text{econ}(N_i) = 1$ and $i \neq 0$, the improved flow formulation must contain a commodity i with destination a node $n_i \in N_i$ (the origin would be the root node) and a requirement of 1. For commodities $0, \ldots, p$, we set

$$v_i^k = \begin{cases} 1 & \text{if } i \in N_k, \\ 0 & \text{otherwise,} \end{cases}$$

and set $v_i^k = 0$ otherwise.

We have not changed anything so far. These values for v_i^k variables are the same as for the NDLC problem. Therefore, for all edges in the partition, $\pi_{ij} = 1$ if i and j are in different sets of the partition and $\pi_{ij} = 0$ otherwise. For commodities $k = 0, \ldots, p$, $q_k = 1$ if $\text{econ}(N_k) = 1$, and $q_k = \text{econ}(N_k)/2$ if $\text{econ}(N_k) \geq 2$. Substituting these values, we obtain the following valid inequality:

$$\frac{1}{2} \sum_{k=0}^{k=p} \sum_{\delta(N_k)} x_{ij} \geq \begin{cases} p & \text{if } I_2 = \phi; \\ \frac{1}{2} \sum_{i \in I_2} \text{econ}(N_i) + |I_1| & \text{otherwise.} \end{cases} \tag{5.9}$$

In this inequality, $I_1 = \{i : \text{econ}(N_i) = 1\}$ and $I_2 = \{i : i \in \text{econ}(N_i) \geq 2\}$.

But since, in the mixed integer program, the variables x_{ij} are either 0 or 1 we can round up the righthand side in this inequality and still maintain feasibility. Thus, the following inequalities are valid:

$$\frac{1}{2} \sum_{k=0}^{k=p} \sum_{\delta(N_k)} x_{ij} \geq \begin{cases} p & \text{if } I_2 = \phi \\ \lceil \frac{1}{2} \sum_{i \in I_2} \text{econ}(N_i) \rceil + |I_1| & \text{otherwise.} \end{cases} \tag{5.10}$$

Grötschel, Monma and Stoer [GMS92c, Sto92] call inequalities (5.10) partition inequalities and show that under appropriate conditions they are facet defining for the SND problem. Our derivation shows that these inequalities are valid for the more general unitary NDC problem. If the number of sets in the partition inequality with odd connectivity requirement greater than one is odd, then the improved undirected formulation implies a weaker form of the partition inequality that we refer to as weak partition inequalities (i.e., inequalities (5.9)). Otherwise, the formulation implies the partition inequality. Note that the weak partition inequalities are stronger than the cutset formulation (as long as $I_1 \neq \phi$).

Since the flow model implies the weak partition inequalities, and does not always imply the partition inequalities, we might like to characterize, in a certain sense, how much stronger a model that contains the partition inequalities is compared to a model that con-

tains the weak partition inequalities.

To compare two classes of valid inequalities, we use the following notion previously introduced by Goemans [Goe93]. Let $\mathcal{X}_1$ and $\mathcal{X}_2$ be two classes of valid inequalities. Then, the *relative strength* of the class of the valid inequalities $\mathcal{X}_1$ to the class of the valid inequalities $\mathcal{X}_2$ is defined as

$$\max\left\{\frac{\{\min \mathbf{cx} : \mathbf{x} \in \mathcal{R}_+^{|E|};\ \mathbf{x} \in \mathcal{X}_1;\ \mathbf{x} \in \mathcal{X}_2\}}{\{\min \mathbf{cx} : \mathbf{x} \in \mathcal{R}_+^{|E|};\ \mathbf{x} \in \mathcal{X}_2\}}\right\}.$$

The relative strength measures how much, in the best case, the objective function of an LP that contains the class of valid inequalities $\mathcal{X}_2$ improves by adding to it the class of valid inequalities $\mathcal{X}_1$.

We have been able to characterize the relative strength of the partition inequalities with respect to the weak partition inequalities when (unlimited) edge replication is permitted.

Theorem 5.3.1 *The relative strength of the class of partition inequalities $\mathcal{X}_1$ with respect to the class of weak partition inequalities $\mathcal{X}_2$ is $\frac{10}{9}$.*

Proof: We will show that by multiplying any feasible solution (including any optimal solution) to the linear program $\{\min \mathbf{cx} : \mathbf{x} \in \mathcal{R}_+^{|E|};\ \mathbf{x} \in \mathcal{X}_2\}$ (we refer to this linear program as LP2) by $\frac{10}{9}$ gives a feasible solution to the linear program $\{\min \mathbf{cx} : \mathbf{x} \in \mathcal{R}_+^{|E|};\ \mathbf{x} \in \mathcal{X}_1;\ \mathbf{x} \in \mathcal{X}_2\}$ (we refer to this linear program as LP1). This result implies that the optimal value to LP1 is at most $\frac{10}{9}$ times the optimal value to LP2. Note that the weak partition inequalities are implied by the partition inequalities. Consequently, we can delete $\mathbf{x} \in \mathcal{X}_2$ from LP1.

LP1 and LP2 differ when the righthand side of the weak partition inequalities is fractional. If we show the maximum ratio between the righthand side of any partition inequality and its corresponding weak partition inequality is $\frac{10}{9}$, we have shown that the relative strength of the partition inequalities with respect to the weak partition inequalities is $\frac{10}{9}$. (Since multiplying any solution that satisfies the weak partition inequalities by $\frac{10}{9}$ gives a solution that satisfies the partition inequalities.)

The righthand sides of the weak partition inequalities and the partition inequalities differ by at most 0.5. This happens when the cardinality of the set $\{i : \text{econ}(N_i)$ odd; and $\text{econ}(N_i) \geq 3\}$ is odd (i.e., for an odd number of sets, $\text{econ}(N_i)$ is (i) odd, and (ii) greater than or equal to 3). Noting that the partition contains at least two sets with the highest value of $\text{econ}(N_i)$, we find that the maximum ratio is obtained by considering a partition with three sets, each with connectivity 3. The righthand side for the weak partition inequality is 4.5 and the righthand side for the partition inequality is 5. The ratio is $\frac{10}{9}$. ■

5.4 Odd-Hole Inequalities

Chopra and Rao [CR94] introduced odd hole inequalities for the Steiner tree problem. These inequalities have the following form. Consider a graph $G_p = (N_p, E_p)$ with $N_p = (T_p \cup Z_p)$, $T_p = \{a_0, a_1, \ldots, a_p\}$ the set of nodes with nonzero connectivity requirements, and $Z_p = \{b_0, b_1, \ldots, b_p\}$ the set of nodes with zero connectivity requirements. The edge set E_p consists of edges of the form $\{a_i, b_i\}$, $\{a_i, b_{i-1}\}$ and $\{b_i, b_{i-1}\}$ for $i = 0, \ldots, p$. Note that we define $b_{-1} = b_p$, $a_{-1} = a_p$, $b_{p+1} = b_0$ and $a_{p+1} = a_0$. An odd-hole inequality is an inequality of the form

$$\sum_{\{i,j\} \in E_p} x_{ij} \geq 2p.$$

When p is even, the graph G_p is called an odd-hole.[6] Chopra and Rao [CR94] show that if p is even and $p \geq 2$, then an odd-hole inequality defines a facet for the Steiner tree polytope on the graph G_p.

We extend the definition of odd-hole inequalities to obtain the following valid inequalities for the NDLC problem.

$$\sum_{\{i,j\} \in E_p} x_{ij} \geq \begin{cases} 2(p+1) & \text{if at least two nodes have a} \\ & \text{connectivity requirement of two;} \\ 2p & \text{otherwise.} \end{cases}$$

To prove that these inequalities are valid, we now show how to project these inequalities from the improved undirected formulation.

Consider the odd-hole shown in Figure 5-6. Nodes a_0 and a_1 have a connectivity requirement of 2, and a_2, a_3 and a_4 have a connectivity requirement of 1. The improved undirected formulation, with root node a_0, has five commodities. Commodities 1, 2, 3 and 4 have a_0 as the origin and a_1, a_2, a_3 and a_4 as the destination nodes. Commodity 5, which we will also refer to as commodity 0, has a_1 as the origin and a_0 as the destination. Note, with this notation, each commodity i has destination a_i.

In the example of Figure 5-6, for all commodities $k = 0, \ldots, 4$, let $v_i^k = 0$ if $i \neq a_k, b_k$ or b_{k-1}. Let $v_{a_k}^k = 2$ and $v_{b_k}^k = v_{b_{k-1}}^k = 1$. Notice that with this choice of values

$$\begin{aligned} v_{a_i}^i - v_{b_i}^i &= v_{b_i}^{i+1} - v_{a_i}^{i+1} = 1 \\ v_{a_i}^i - v_{b_{i-1}}^i &= v_{b_{i-1}}^{i-1} - v_{a_i}^{i-1} = 1 \\ v_{b_{i-1}}^{i-1} - v_{b_i}^{i-1} &= v_{b_i}^{i+1} - v_{b_{i-1}}^{i+1} = 1. \end{aligned}$$

All other $v_i^k - v_j^k = 0$. Thus, for each edge $\{i, j\}$, $\sum_{k \in K} \max(0, v_j^k - v_i^k) = \sum_{k \in K} \max(0, v_i^k - v_j^k) = 1$. Therefore in accordance with Theorem 5.2.4 we can set $u_{a_k b_k}^{(k+1)k}$, $u_{a_k b_{k-1}}^{(k-1)k}$ and

[6] The odd in odd-hole refers to the number of nodes with nonzero connectivity requirements.

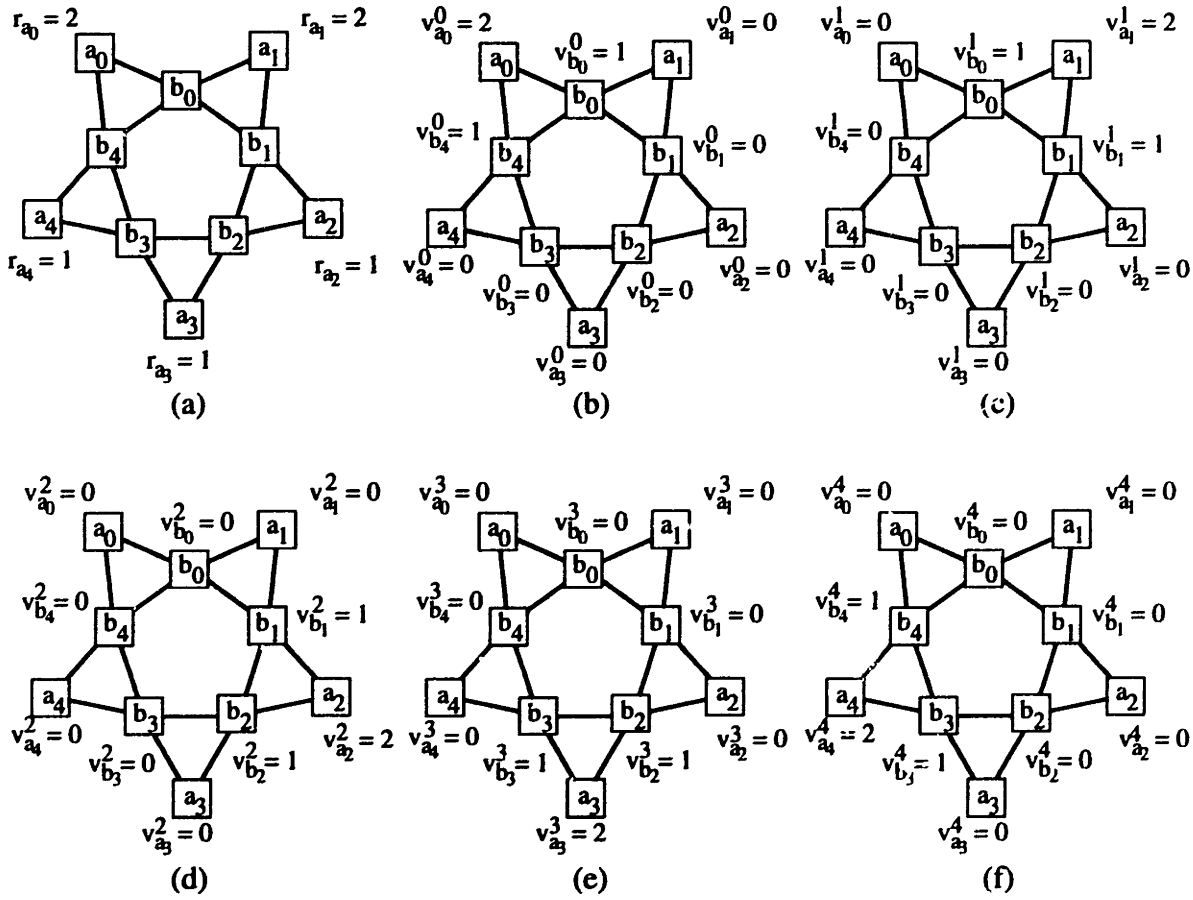


Figure 5-6: Projecting odd-hole inequalities for a ten node example. (a) Connectivity requirements (b) Value of the v_i^0 variables (c) Value of the v_i^1 variables (d) Value of the v_i^2 variables (e) Value of the v_i^3 variables (f) Value of the v_i^4 variables.

$u_{b_k b_{k-1}}^{(k+1)(k-1)}$ to 1 for $k = 0, \ldots, 4$, and $u_{ij}^{hk} = 0$ otherwise. Since $\sum_{k \in K} v_{a_k}^k = 10$, this choice of variables defines the odd-hole inequality $\sum_{\{i,j\} \in E_4} x_{ij} \geq 10$.

In general, let node a_0 be the root node. For each node a_l, select a commodity l with $D(l) = a_l$. For commodities $k \in \{0, \ldots, p\}$, set

$$v_i^k = \begin{cases} 2 & \text{if } i = a_k; \\ 1 & \text{if } i = b_k \text{ or } i = b_{k-1}; \\ 0 & \text{otherwise,} \end{cases}$$

and set all other v_i^k values to zero.

With this choice of the v_i^k variables, we once again ensure that $\sum_{k \in K} \max(0, v_j^k - v_i^k) = \sum_{k \in K} \max(0, v_i^k - v_j^k) = 1$ for any edge $\{i, j\}$. Thus, by choosing the values of u_{ij}^{kh} as indicated in Theorem 5.2.4, we find that $\pi_{ij} = \sum_{k \in K} \sum_{h \in K} u_{ij}^{kh}$ is 1 for each edge $\{i, j\}$;

and $\pi_0 = \sum_{k \in K} v^k_{D(k)} = 2(p+1)$, since $v^k_{D(k)} = 2$ for commodities $0, \ldots, p$, and $v^k_{D(k)} = 0$ otherwise.

Suppose the graph G_p has more edges than E_p. What inequality is implied? Retain the same choice of v^k_i variables and consider an edge $\{i,j\} \notin E_p$. This edge can be of three types—$\{a_i, b_j\}$, $\{b_i, b_j\}$ and $\{a_i, a_j\}$. For edges of type $\{a_i, a_j\}$,

$$v^j_{a_j} - v^j_{a_i} = v^i_{a_i} - v^i_{a_j} = 2.$$

For edges of type $\{a_i, b_j\} \notin E_p$, $j \neq i$ and $j \neq i-1$,

$$v^j_{b_j} - v^j_{a_i} = v^{j+1}_{b_j} - v^{j+1}_{a_i} = 1$$

and

$$v^i_{a_i} - v^i_{b_j} = 2.$$

Finally, for edges of type $\{b_i, b_j\} \notin E_p$, $j \neq i-1$ and $j \neq i+1$,

$$v^j_{b_j} - v^j_{b_i} = v^{j+1}_{b_j} - v^{j+1}_{b_i} = v^i_{b_i} - v^i_{b_j} = v^{i+1}_{b_i} - v^{i+1}_{b_j} = 1.$$

Thus, for any edge $\{i,j\} \notin E_p$,

$$\sum_{k \in K} \max(0, v^k_i - v^k_j) = \sum_{k \in K} \max(0, v^k_j - v^k_i) = 2.$$

Selecting u^{kh}_{ij} values as prescribed by Theorem 5.2.4, shows that $\pi_{ij} = \sum_{k \in K} \sum_{h \in K} u^{kh}_{ij}$ is 2 for each edge $\{i,j\} \notin E_p$. The projected inequality

$$\sum_{\{i,j\} \in E_p} x_{ij} + \sum_{\{i,j\} \in E \setminus E_p} 2x_{ij} \geq 2(p+1)$$

is called a lifted odd-hole inequality and is also a facet-defining inequality for the Steiner tree problem (see Chopra and Rao [CR94]).

If the problem has no node with a connectivity requirement of two, it does not contain any commodity with destination the root node. Therefore we select p commodities, one for each node $a_1, \ldots, a_p$, each with origin a_0 (the root node). With the same choice of v^k_i for commodities $1, \ldots, p$, we obtain $\max(\sum_{k \in K} \max(0, v^k_j - v^k_i), \sum_{k \in K} \max(0, v^k_i - v^k_j)) = 1$ for all edges $\{i,j\} \in E_p$ and $\max(\sum_{k \in K} \max(0, v^k_j - v^k_i), \sum_{k \in K} \max(0, v^k_i - v^k_j)) = 2$ for edges $\{i,j\} \in E \setminus E_p$. Since we have only p commodities, the resulting righthand side is $\pi_0 = \sum_{k \in K} v^k_{D(k)} = 2p$.

Both the odd-hole inequality and its lifted version are new valid inequalities for the NDLC problem.

By examining the cone element that generates the odd-hole inequality, we can obtain valid odd-hole inequalities for the unitary NDC problem. In the next section, we study a generalization of odd-hole inequalities called combinatorial design inequalities. Therefore, to avoid repetition, we will not derive odd-hole inequalities for the unitary NDC problem.

5.5 Combinatorial Design Inequalities

Goemans [Goe94b] introduced a new class of facet defining valid inequalities called *combinatorial design inequalities* for Steiner tree problems. He showed that under appropriate conditions combinatorial design inequalities are facet defining for the Steiner tree problem. He derives the combinatorial design inequalities by projecting from a node weighted (undirected) extended formulation for the Steiner tree problem. Goemans and Myung [GM93] show the LP relaxation of the node weighted model is equivalent to the improved undirected flow formulation for the Steiner tree problem. Therefore we can expect the combinatorial design inequalities to be in the projection of the improved flow model. Since we are not aware of any similar node weighted (undirected) formulation for the unitary NDC problem, we show how to project out the combinatorial design inequalities from the improved undirected flow formulation (5.5). As a result, we generalize the combinatorial design inequality to the unitary NDC problem, and obtain a new class of valid inequalities for this problem.

The description of the combinatorial design inequality is fairly involved. Let $T_p = \{a_0, \ldots, a_p\}$ be the set of nodes with nonzero connectivity requirements. $Z_q = N_p - T_p = \{b_0, \ldots, b_q\}$ is the set of nodes with zero connectivity requirements. Associate with each node a_i of T_p a subset T_{a_i} containing elements of Z_q. Based on these subsets, we also define sets Z_{b_i} associated with each node b_i in Z_q. Z_{b_i} contains those elements of T_p whose associated subset contains the node b_i. For example, consider the odd hole with $p = 4$ (see Figure 5-6). $T_4 = \{a_0, a_1, a_2, a_3, a_4\}$ and $Z_4 = \{b_0, b_1, b_2, b_3, b_4\}$. Here $T_{a_0} = \{b_0, b_4\}$, $T_{a_1} = \{b_0, b_1\}$, $T_{a_2} = \{b_1, b_2\}$, $T_{a_3} = \{b_2, b_3\}$, $T_{a_4} = \{b_3, b_4\}$. Thus, the sets associated with the elements in Z_4 are $Z_{b_0} = \{a_0, a_1\}$, $Z_{b_1} = \{a_1, a_2\}$, $Z_{b_2} = \{a_2, a_3\}$, $Z_{b_3} = \{a_3, a_4\}$, $Z_{b_4} = \{a_4, a_0\}$. (The odd-hole inequality is a special case of the combinatorial design inequality).

Define the $q \times p$ matrix $\mathbf{D} = [d_{ij}]$ with $d_{ij} = 1$ if $a_j \in Z_{b_i}$ and $d_{ij} = 0$ otherwise. The following two conditions are imposed on $\mathbf{D}$: (i) $\text{rank}(\mathbf{D}) = p$, and (ii) the unit vector e belongs to the cone generated by the columns of $\mathbf{D}$; i.e., $\mathbf{Dy} = e$ for some vector $\mathbf{y} \in \mathcal{R}^q_+$. For any fixed $d > 0$, if we set $\beta_j = dy_j$, we see that

$$\sum_{a_j \in Z_{b_i}} \beta_j = d \qquad \text{for all } i = 1, \ldots, q. \tag{5.11}$$

If we select d so that the greatest common divisor of $\beta_1, \ldots, \beta_p$, and d is 1, the coefficients of x_{ij} in the following combinatorial design inequalities will be integer and as small as possible. For every edge $\{s, t\}$, define

$$d_{st} = \begin{cases} \sum_{k:a_k \in (Z_{b_i} \cap Z_{b_j})} \beta_k & \{s,t\} = \{b_i, b_j\} \text{ where } b_i, b_j \in Z_p; \\ \beta_j & \{s,t\} = \{a_j, b_i\} \text{ with } b_i \in T_{a_j}; \\ 0 & \text{otherwise.} \end{cases}$$

For the Steiner tree problem, the inequality

$$\sum_{\{i,j\}\in E}(d-d_{ij})x_{ij} \geq dp,$$

is a combinatorial design inequality. The odd-hole inequality is a special case of the combinatorial design inequality when $d=2$, $\beta_j=1$ for $j=0,\ldots,p$, and $|T_p|=|Z_q|$.

We extend the definition of the combinatorial design inequality to obtain the following valid inequalities for the NDLC problem.

$$\sum_{\{i,j\}\in E}(d-d_{ij})x_{ij} \geq \begin{cases} d(p+1) & \text{if at least two nodes have a} \\ & \text{connectivity requirement of two;} \\ dp & \text{otherwise.} \end{cases} \tag{5.12}$$

Let us now project the combinatorial design inequality from the improved undirected formulation. In the improved undirected formulation, let node a_0 be the root node. For each node a_l, select a commodity l with destination a_l and set

$$v_i^k = \begin{cases} d & \text{if } i=a_k; \\ \beta_k & \text{if } i\in T_{a_k}; \\ 0 & \text{otherwise.} \end{cases}$$

For all other commodities k set v_i^k to zero.

Consider an edge $\{s,t\}=\{a_j,b_i\}$ with $b_i\in T_{a_j}$. When a_j is the destination node, $v_{a_j}^j=d$ and $v_{b_i}^j=\beta_j$ if $i\in T_{a_j}$. Consider any node $a_k\in Z_{b_i}\backslash\{a_j\}$. $v_{a_k}^l\neq 0$ only if a_k is a destination node for commodity l, i.e., $l=k$ and, in this case, $v_{b_i}^k=\beta_k$ and $v_{a_j}^k=0$. Note that, $v_{b_i}^k-v_{a_j}^k=\beta_k$. Thus,[7]

$$\sum_{a_k\in\{Z_{b_i}\backslash\{a_j\}\}} v_{b_i}^k-v_{a_j}^k = \sum_{a_k\in\{Z_{b_i}\backslash\{a_j\}\}}\beta_k \overset{(5.11)}{=} d-\beta_j = v_{a_j}^j-v_{b_i}^j.$$

If $a_k\notin Z_{b_i}$, then $v_{a_j}^k=v_{b_i}^k=0$. Therefore, $\sum_{k\in K}\max(0,v_{a_j}^k-v_{b_i}^k)=\sum_{k\in K}\max(0,v_{b_i}^k-v_{a_j}^k)=d-\beta_j$. By Theorem 5.2.4, in the projected inequality $d-\beta_j$ is the coefficient of an edge $\{s,t\}=\{a_j,b_i\}$ with $b_i\in T_{a_j}$.

Consider an edge $\{s,t\}=\{b_i,b_j\}$. For any commodity k with destination a_k, $v_{b_i}^k$ and $v_{b_j}^k$ differ only if exactly one of b_i and b_j belongs to T_{a_k}. If b_i belongs to T_{a_k} then $v_{b_i}^k=\beta_k$ and $v_{b_j}^k=0$. Thus $v_{b_i}^k-v_{b_j}^k=\beta_k$. Summing over all sets T_{a_k} that contain node b_i but not node b_j, we find that

$$\begin{aligned}\sum_{k:b_j\in T_{a_k};b_i\notin T_{a_k}} v_{b_i}^k-v_{b_j}^k &= \sum_{k:b_j\in T_{a_k}:b_i\notin T_{a_k}}\beta_k \\ &\overset{(5.11)}{=} d-\sum_{k:b_j\in T_{a_k};b_i\in T_{a_k}}\beta_k.\end{aligned}$$

[7]The notation $\overset{(5.11)}{=}$ means that the equality follows from expression (5.11).

Similarly, summing up over all sets T_{a_k} that contain node b_j but not node b_i, we find that

$$\sum_{k:b_i\in T_{a_k};b_j\notin T_{a_k}} v^k_{b_j} - v^k_{b_i} = \sum_{k:b_i\in T_{a_k};b_j\notin T_{a_k}} \beta_k \overset{(5.11)}{=} d - \sum_{k:b_i\in T_{a_k};b_j\in T_{a_k}} \beta_k.$$

Therefore, $\sum_{k\in K}\max(0, v^k_{b_j} - v^k_{b_i}) = \sum_{k\in K}\max(0, v^k_{b_i} - v^k_{b_j}) = d - \sum_{k:b_j\in T_{a_k};b_i\in T_{a_k}}\beta_k$. By Theorem 5.2.4, $d - \sum_{k:b_j\in T_{a_k};b_i\in T_{a_k}}\beta_k$ is the coefficient of an edge $\{s,t\} = \{b_i, b_j\}$ in the projected inequality.

All the other edges are either of the form $\{s,t\} = \{a_i, a_j\}$ or $\{s,t\} = \{a_j, b_i\}$ with $b_i \notin T_{a_j}$. For any edge of the form $\{a_i, a_j\}$, $v^j_{a_j} - v^j_{a_i} = v^i_{a_i} - v^i_{a_j} = d$ for commodities i and j and $v^k_{a_i} = v^k_{a_j} = 0$ otherwise. Consider an edge of the form $\{s,t\} = \{a_j, b_i\}$ with $b_i \notin T_{a_j}$. For all commodities k with $a_k \in Z_{b_i}$, i.e. a_k is the destination, $v^k_{b_i} - v^k_{a_i} = \beta_k$. For commodity j, a_j is the destination and $v^j_{a_j} - v^j_{b_i} = d$. For all other commodities k, $v^k_{a_j} = v^k_{b_i} = 0$. Therefore, for both edges of the form $\{s,t\} = \{a_i, a_j\}$ and $\{s,t\} = \{a_j, b_i\}$ with $b_i \notin T_{a_j}$, we find that $\sum_{k\in K}\max(0, v^k_s - v^k_t) = \sum_{k\in K}\max(0, v^k_s - v^k_t) = d$. By Theorem 5.2.4, d is the coefficient of these edges in the projected inequality.

The righthand side of the projected inequality is $\pi_0 = \sum_k v^k_{D(k)} = d(p+1)$ since $v^k_{D(k)} = d$ for commodities $0,\ldots,p$, and $v^k_{D(k)} = 0$ otherwise.

If the problem has exactly one node or no nodes with a connectivity requirement of 2, with the same choice of v^k_i variables we obtain the same coefficients for x_{ij} and a righthand side of dp (there will be no commodity with destination the root node a_0).

5.5.1 Combinatorial Design Inequalities for the Unitary NDC Problem

To conclude this discussion, we examine the valid inequality for the unitary NDC problem that is implied by the cone element that generates the combinatorial design inequality for the NDLC problem.

For each node a_k, $k = 0,\ldots,p$, we select a commodity k with destination a_k as follows. If the maximum connectivity requirement r_{a_k} of node a_k is greater than or equal to 2, then we select a commodity k with destination node a_k and flow requirement $r_{a_k}/2$. If the maximum connectivity requirement r_{a_k} of node a_k is equal to 1, then we select a commodity k with destination node a_k and flow requirement 1. Retain the same choice of variables v^k_i. The projected inequality is

$$\sum_{\{i,j\}\in E}(d - d_{ij})x_{ij} \geq \begin{cases} dp & \text{if } L_2 = \phi, \\ d(\frac{1}{2}\sum_{l\in L_2} r_l + |L_1|) & \text{otherwise.} \end{cases} \tag{5.13}$$

In this inequality $L_1 = \{i : i \in T_p, r_i = 1\}$ and $L_2 = \{i : i \in T_p, r_i \geq 2\}$ (r_i denotes the maximum connectivity requirement of a node).

Once again, noting that the lefthand side should be integer if the x variables are integer, we can round up the righthand side, giving the inequality

$$\sum_{\{i,j\}\in E}(d-d_{ij})x_{ij} \geq \begin{cases} dp & \text{if } L_2 = \phi, \\ (\lceil \frac{d}{2} \sum_{l\in L_2} r_l \rceil + d|L_1|) & \text{otherwise.} \end{cases} \quad (5.14)$$

We refer to inequalities (5.13) and (5.14) as weak combinatorial design and combinatorial design inequalities. Noting that $d \geq 1$, it is easy to prove a result similar to Theorem 5.3.1—namely, if edge replication is permitted, the value of the linear program determined by the weak combinatorial design inequalities is at least $\frac{10}{9}$ths the value of the linear program determined by the combinatorial design inequalities.

5.6 Node-Connectivity Requirements

So far, we have restricted our attention to edge-connectivity requirements. We now briefly show how to improve the flow formulation for node-connectivity problems.

Because node connectivity implies edge connectivity, all the valid inequalities derived for the edge-connectivity version of the unitary NDC problem are valid for the node-connectivity version. That is, the partition, odd-hole and combinatorial design inequalities are valid for the node-connectivity version of the unitary NDC problem. Stoer [Sto92] shows that under certain conditions, the partition inequalities are facet defining for the node-connectivity version of the SND problem.

We now show a simple way to strengthen the flow-based formulation for the unitary NDC problem with node-connectivity requirements. This method creates additional commodities derived from the directing procedure. To distinguish between the flow variables that impose node-connectivity requirements and the flow variables that direct the problem, we denote the former by **f** and the latter by **g**. We also let K^f denote the set of commodities that impose node-connectivity requirements and K^g denote the set of commodities that direct the problem. We can state the improved undirected flow formulation for the unitary NDC problem with node-connectivity requirements as follows.

Improved undirected flow formulation for the unitary NDC problem with node-connectivity requirements

$$\text{Minimize} \quad \sum_{\{i,j\}\in E} c_{ij}x_{ij} \quad (5.15a)$$

$$\text{subject to:} \quad \sum_{j\in N} f^k_{ji} - \sum_{l\in N} f^k_{il} = \begin{cases} -r_k & \text{if } i = O(k); \\ q_k & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } i \in N \;\&\; k \in K^f \quad (5.15b)$$

$$\left.\begin{matrix} f_{ij}^k \\ f_{ji}^k \end{matrix}\right\} \leq x_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K^f \tag{5.15c}$$

$$\sum_{l \in N} f_{il}^k \leq 1 \qquad \text{for all } k \in K^f \text{ \& all } i \neq O(k),\ D(k) \tag{5.15d}$$

$$\sum_{j \in N} g_{ji}^k - \sum_{l \in N} g_{il}^k = \begin{cases} -q_k & \text{if } i = O(k); \\ q_k & \text{if } i = D(k); \text{ for all } i \in N \text{ \& } k \in K^g \\ 0 & \text{otherwise;} \end{cases} \tag{5.15e}$$

$$g_{ij}^k + g_{ji}^h \leq x_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and } k,h \in K^g \tag{5.15f}$$

$$f_{ij}^k, f_{ji}^k \geq 0 \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K^f \tag{5.15g}$$

$$g_{ij}^k, g_{ji}^k \geq 0 \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K^g \tag{5.15h}$$

$$x_{ij} \in \{0,1\} \qquad \text{for all } \{i,j\} \in E. \tag{5.15i}$$

The set of commodities in K^f have the interpretation described in Section 4.1.2 (see page 92). The set of commodities in K^g have the same interpretation as for the unitary NDC problem with edge-connectivity requirements. We note, as in Section 5.1.2, this formulation is stronger than the undirected flow formulation for the unitary NDC problem with node-connectivity requirements only when some edge $\{s,t\}$ on the requirement spanning tree (obtained by considering the node-connectivity requirements as edge-connectivity requirements) has $r_{st} = 1$.

5.6.1 Combined Edge- and Node-Connectivity Requirements in the Unitary NDC Problem

In Chapter 4 we described a model for combined edge- and node-connectivity requirements (see Section 4.5.1). We now describe a similar model for combined edge- and node-connectivity requirements in the unitary NDC problem.

In this model, we define the connectivity requirements as follows. Between each pair of nodes $\{s,t\}$, we are given an edge-connectivity requirement r_{st}^e and a node-connectivity requirement r_{st}^n. We require that $r_{st}^e \geq r_{st}^n$ for each edge $\{s,t\}$. Design a network that contains, for every pair of nodes i and j, r_{ij}^e edge-disjoint paths and r_{ij}^n node-disjoint paths.

For the SND problem, we describe this model as follows. Each node has an edge-connectivity requirement r_v^e and a node-connectivity requirement r_v^n. We require that $r_v^e \geq r_v^n$ for each node. Design a network that contains, for every pair of nodes i and j, $\min(r_i^e, r_j^e)$ edge-disjoint paths and $\min(r_i^n, r_j^n)$ node-disjoint paths.

Formulation (5.15) can easily model these requirements. We model the edge-connectivity requirements with the commodities K^e and the node-connectivity requirements with the commodities K^n.

5.7 Concluding Remarks

In this chapter we described an improved flow formulation for the unitary NDC problem. When solved as a linear program, the optimal value of this formulation provides a tighter lower bound on the cost of an optimal design than does the linear programming relaxation of the standard cutset formulation. By projecting out the flow variables from the improved model and rounding up the righthand side of the projected inequality, we derived three classes of valid inequalities (partition, odd-hole, and combinatorial design) for the unitary NDC problem (in the space of the edge selection variables). These inequalities are generalizations of known inequalities for the Steiner tree problem. Using the notion of relative strength of a class of valid inequalities with respect to another class of valid inequalities, we showed that the relative strength of the partition inequalities with respect to the weak partition inequalities is 10/9, and the relative strength of the combinatorial design inequalities with respect to the weak combinatorial design inequalities is 10/9.

Grötschel, Monma and Stoer [GMS95] have previously examined the directed cut formulation (5.3) as applied to the SND problem when the connectivity requirements of the nodes are restricted to be even or one. They called all inequalities in the space of the edge selection variables x_{ij} that can be projected from the directed cut formulation *Prodon inequalities*. Because of the equivalence of the improved undirected flow formulation (5.5) and the directed cut formulation (5.4), Prodon inequalities are equivalent to the inequalities that can be projected out from the improved undirected flow formulation. Grötschel, Monma and Stoer (5.3) also showed how to project out the partition inequalities using the directed cut formulation for the SND problem, but did not investigate other types of inequalities that might be obtainable by projection. If all the connectivity requirements are even, they showed that other than the cut constraints, no Prodon inequality is facet defining. This result corroborates the observation we made earlier that the improved undirected flow model is stronger than the cutset model only if $r_{st} = 1$ for some edge $\{s, t\}$ of the requirement spanning tree.

Although we have shown that the flow formulation implies three classes of valid inequalities (some in weaker form), some classes of facet defining inequalities are not obtainable by projecting from the flow formulation.

Consider the SND problem shown in Figure 5-7. In this problem, nodes a, b, and c have a connectivity requirement of 2. Figure 5-7 shows the underlying graph and the cost of the edges. One optimal integer solution to the problem has the edges $x_{ad} = x_{db} = x_{bc} = x_{ca} = 1$ and has cost 8. In the improved flow model assume commodity 1 is from node a to b, commodity 2 is from node a to c, commodity 3 is from node b to a and commodity 4 is from node c to a. The optimal LP solution to the improved undirected flow model (5.5) is $f^{a1}_{ad} = f^{a1}_{db} = f^{c1}_{id} = f^{c1}_{dc} = f^{a2}_{bd} = f^{a2}_{da} = f^{c2}_{cd} = f^{c2}_{da} = 0.5$, $f^{a1}_{ab} = f^{a1}_{ac} = f^{a1}_{cb} = f^{c1}_{ab} = f^{c1}_{ac} =$

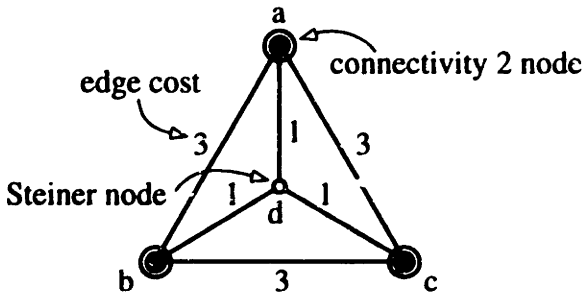


Figure 5-7: The flow formulation does not always give integer solutions. The LP solution is 7.5 and the optimal solution to the problem has cost 8.

$f_{bc}^{c1} = f_{ba}^{a2} = f_{ca}^{a2} = f_{bc}^{a2} = f_{ca}^{c2} = f_{cb}^{c2} = f_{ba}^{c2} = 0.25$, $x_{ad} = x_{bd} = x_{cd} = 1$ and $x_{ab} = x_{bc} = x_{ca} = 0.5$ with cost 7.5. Grötschel, Monma and Stoer [GMS92b] have introduced a class of inequalities for the SND problem, called r-cover inequalities, that eliminates such fractional solutions in the cutset model.

It is unlikely that any polynomial size flow formulation implies the r-cover inequalities because, in a certain sense, the r-cover inequalities are complements of the blossom inequalities of the matching polytope. Therefore, if we can project out the r-cover inequalities from a polynomial size flow-based formulation, then we will have a polynomial size extended formulation for the matching polytope. However, Yanakakis [Yan88] showed that the matching polytope has no polynomial size extended formulation unless $\mathcal{P} = \mathcal{NP}$ (i.e., unless all $\mathcal{NP}$-hard problems can be solved in polynomial time).

We also described an improved flow model for the node-connectivity version of the unitary NDC problem that implies the partition, odd-hole, and combinatorial design inequalities. However, we were unable to project out of the improved flow model some classes of inequalities that Stoer [Sto92] has described for the node-connectivity version of the SND problem. Is it possible to strengthen the improved flow model for the unitary NDC problem to obtain these and other valid inequalities? Notice that Formulation (5.15) does not include any direct interaction between the flow variables that impose the node-connectivity requirements and the flow variables that arise from the directing procedure. This observation leads to the following questions. Is there a way to link the two sets of variables? Does linking the variables improve the formulation? Answering these questions might lead to better flow-based models for the unitary NDC problem with node-connectivity requirements.

Grötschel, Monma and Stoer [GMS92a, GMS92c] found the cut and partition inequalities to be the most useful in polyhedral computations for the SND problem. These results suggest that the improved flow model might be a good (strong) formulation for the SND problem. In the next chapter we use the improved flow model to devise a new dual-ascent algorithm for the NDLC problem and test this assertion empirically on problems with up

to 300 nodes and 3000 edges. The results of this experimentation suggest that the linear programming relaxation of the improved flow formulation provides a good approximation to this mixed integer program model.

Chapter 6

Low-Connectivity Requirements

In this chapter we consider the network design problem with low connectivity requirements (NDLC). Network providers (e.g., the RBOCs) frequently need to connect a set of customer nodes (a node might represent an office building of a large corporation). To provide adequate service, the network often needs to connect some of these customers (e.g., those generating high revenues) to each other by two physically diverse paths. Some nodes are optional: they need not be contained in the network, but can be used if they lower the cost of the network design. Because network failures tend to be rare, many network providers (e.g., the telcos) believe it is important to protect against single link and single node failures. Therefore, the NDLC problem is significantly more important to these corporations than the other survivability problems we consider.

In this chapter we consider the edge-connectivity version of the NDLC problem. This restriction is reasonable for two reasons. First, at this time protecting against single link failures in a network seems to be of greater importance to telcos than protecting against single node failures in a network [TEM93a]. Second, recall (see Section 2.2.2) that for the NDLC problem, if the costs satisfy the triangle inequality and the underlying graph is complete, then the cost of the optimal solution of the edge-connectivity model is the same as the node-connectivity model. In telecommunication settings, typically the costs associated with the links are Euclidean (and so satisfy the triangle inequality when the underlying graph is complete). Therefore, in many instances solving the edge-connectivity version of the NDLC problem is sufficient.

Using an undirected flow formulation, in Chapter 4 we described a dual-ascent procedure for the general network design problem with connectivity requirements (NDC). This dual-ascent algorithm is a potential solution technique for the NDLC problem because the problem is a special type of NDC problem. In this chapter we provide an alternative dual-ascent procedure (for the NDLC problem) that exploits the improved flow formulation (model (5.5)) described in Chapter 5.

In computational experiments reported in this chapter, this dual-ascent algorithm has solved problems with up to 300 nodes and 3000 edges to within 4 percent of optimality.

6.1 Directed Flow Formulation

Formally, we can state the NDLC problem as follows. Given a graph $G = (N, E)$ and a connectivity requirement $r_i \in (0, 1, 2)$ for each node, design a network at minimum cost that contains $r_{st} = \min(r_s, r_t)$ edge-disjoint paths between nodes s and t. We first consider the NDLC problem without edge replication. Later, we show how to apply a modified version of the dual-ascent algorithm for situations with edge replication. In this chapter, we let T_1 denote the nodes with connectivity requirement 1, T_2 the nodes with connectivity requirement 2, and T $(= T_1 \cup T_2)$ the nodes with nonzero connectivity requirements.

Before developing the dual-ascent algorithm, we first review some of the material presented in Chapter 5, now in the context of the NDLC problem.

A feasible integer solution to the NDLC problem consists of a single 2-edge-connected component that contains T_2, with trees attached to it. When applied to the 2-edge-connected network, the Nash-Williams theorem says that we can orient its edges so that the resulting digraph is strongly connected. In the literature, this result is referred to as Robbins' theorem [Rob39].

Theorem 6.1.1 (Robbins) *An undirected graph is 2-edge connected if and only if*[1] *we can orient its edges so that the resulting directed graph is strongly connected.*

Therefore, we can orient the edges in the main 2-edge connected component, i.e., the one that contains T_2, so that it becomes a strongly connected digraph. We can also orient every tree away from the 2-edge connected component. As a result, the oriented network contains a directed path from every node with a connectivity requirement of 2 to every other node with a connectivity requirement. Figure 6-1 illustrates this directing procedure.

Consequently, we formulate the NDLC problem as follows.

Improved undirected flow formulation for the NDLC problem:

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{6.1a}$$

$$\text{subject to:} \quad \sum_{j \in N} f_{ji}^k - \sum_{l \in N} f_{il}^k = \begin{cases} -1 & \text{if } i = O(k); \\ 1 & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } i \in N \text{ and } k \in K \tag{6.1b}$$

[1] For the only if part, we impose a tacit assumption that the directed graph does not contain both arcs (i, j) and (j, i).

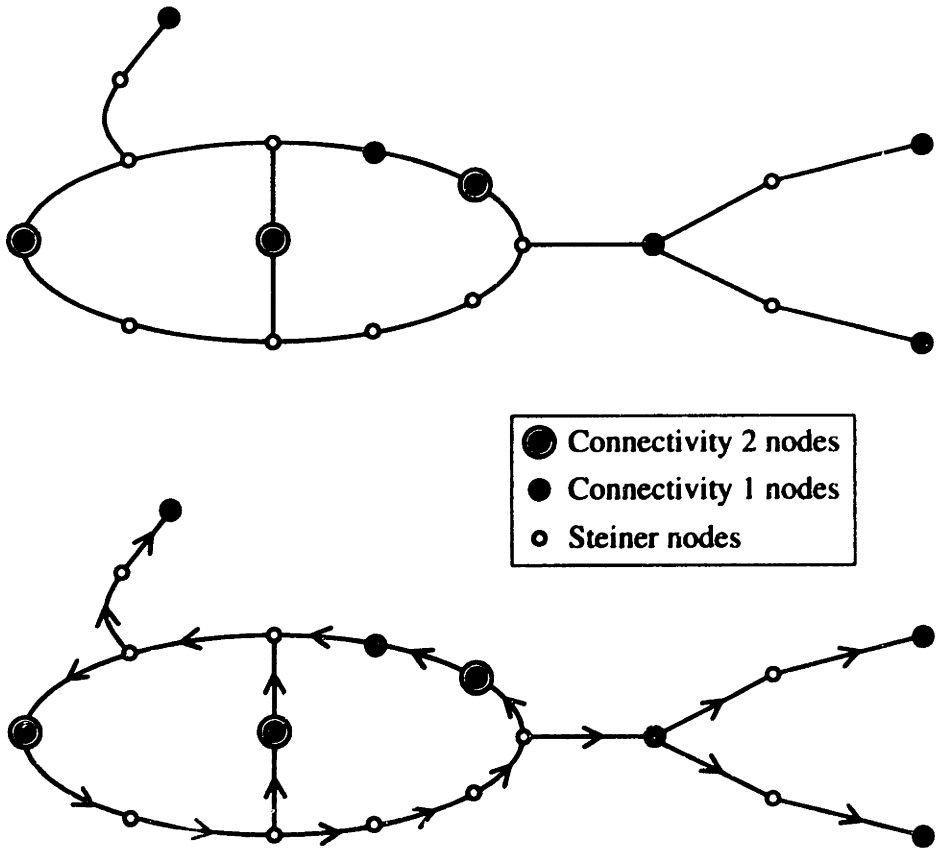


Figure 6-1: Directing procedure applied to the NDLC problem.

$$f_{ij}^k + f_{ji}^h \leq x_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and } k,h \in K \qquad (6.1c)$$

$$x_{ij} \leq 1 \qquad \text{for all } \{i,j\} \in E \qquad (6.1d)$$

$$f_{ij}^k, f_{ji}^k \geq 0 \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \qquad (6.1e)$$

$$x_{ij} \geq 0 \qquad \text{and integer for all } \{i,j\} \in E. \qquad 6.1f)$$

To define the commodities in this model (see Section 5.1.2), we select any node with a connectivity requirement of 2 as a root node. We then define two types of commodities. (i) Any node i from T, other than the root node, defines a commodity with the origin as the root node and with node i as the destination, (ii) Any node i (other than the root) in T_2, defines a commodity with node i as the origin and the root node as the destination.

Alternatively, we can formulate the problem on a digraph. To describe this formulation, we first convert the undirected network to a directed one $D = (N, A)$ by replacing each undirected edge $\{i,j\}$ by two directed arcs (i,j) and (j,i), each with the same cost as the undirected edge $\{i,j\}$. We say that arcs (i,j) and (j,i) are *twins*. Let y_{ij} be 1 if arc (i,j) is in the design and be 0 otherwise. The following model prescribes the directed flow formulation.

Directed flow formulation for the NDLC problem:

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}y_{ij} \tag{6.2a}$$

$$\text{subject to:} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k = \begin{cases} -1 & \text{if } i = O(k); \\ 1 & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } i \in N \text{ and } k \in K \tag{6.2b}$$

$$f_{ij}^k \leq y_{ij} \qquad \text{for all } (i,j) \in A \text{ and } k \in K \tag{6.2c}$$

$$y_{ij} + y_{ji} \leq 1 \qquad \text{for all } \{i,j\} \in E \tag{6.2d}$$

$$f_{ij}^k, f_{ji}^k \geq 0 \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{6.2e}$$

$$y_{ij} \geq 0 \qquad \text{and integer for all } \{i,j\} \in A. \tag{6.2f}$$

As is the case for the minimum spanning tree problem (see Section 3.1), the improved undirected flow formulation and the directed flow formulation are equivalent models of the NDLC problem (as mixed integer programs or as relaxed linear programs).[2]

Like the NDC problem, the flow-based formulation for the NDLC problem can be quite large; it contains as many as $\mathcal{O}(|N||E|)$ constraints and variables. As a result, solving even its linear programming relaxation is quite challenging (for some computational experiments with this LP relaxation see Section 6.6). Therefore, we develop a dual-ascent approach for the problem.

6.2 Dual-Ascent Procedure

In developing a dual-ascent procedure for the NDLC problem, we draw on many ideas presented in Section 3.4 for the Steiner branching problem.

We will apply the dual-ascent algorithm to the following dual of the linear programming relaxation of the directed flow formulation for the NDLC problem.

Dual of directed flow formulation for the NDLC problem:

$$\text{Maximize} \quad \sum_{k\in K} v_{D(k)}^k - \sum_{\{i,j\}\in E} u_{ij} \tag{6.3a}$$

$$\text{subject to} \quad v_j^k - v_i^k \leq w_{ij}^k \qquad \text{for all } (i,j) \in A \text{ and } k \in K \tag{6.3b}$$

$$\sum_{k\in K} w_{ij}^k - u_{ij} \leq c_{ij} \qquad \text{for all } (i,j) \in A \tag{6.3c}$$

$$u_{ij} \geq 0 \qquad \text{for all } \{i,j\} \in E \tag{6.3d}$$

$$w_{ij}^k \geq 0 \qquad \text{for all } (i,j) \in A \text{ and } k \in K. \tag{6.3e}$$

[2]Goemans and Myung [GM93] first showed this result in the context of the Steiner tree problem.

In this dual linear program, v_j^k is the dual variable for the flow balance constraint (6.2b) at node j for commodity k, w_{ij}^k is the dual variable for constraint (6.2c), and u_{ij} is the dual variable for constraint (6.2d). Since for each commodity k, one flow balance constraint in the primal problem (6.2) is redundant, we can choose an arbitrary value for one dual variable v_j^k. We set $v_{O(k)}^k = 0$ for each commodity $k \in K$. In this dual, note that the variable u_{ij} is associated with the edge $\{i,j\}$ (so $u_{ij} = u_{ji}$), but the variable w_{ij}^k is associated with the arc (i,j).

For given w_{ij}^k and u_{ij} values, the dual problem separates by commodity. For commodity k, the problem $\{\max v_{D(k)}^k$, subject to $v_j^k - v_i^k \leq w_{ij}^k$, for all $(i,j) \in A\}$ is the dual to a directed shortest path problem between the nodes $O(k)$ and $D(k)$ on the digraph D with arc costs w_{ij}^k. As a result, if we prescribe feasible values for the $\mathbf{w}$ and $\mathbf{u}$ variables, it is easy to calculate the shortest path lengths for each subproblem and thereby obtain a dual solution for that choice of $\mathbf{w}$ and $\mathbf{u}$.

To maintain dual feasibility, we need to ensure the solution satisfies constraints (6.3c), (6.3d) and (6.3e). Let s_{ij} be the slack in constraint (6.3c) (i.e., $s_{ij} = c_{ij} + u_{ij} - \sum_{k \in K} w_{ij}^k$). As before, if $s_{ij} = 0$, we say that the arc is *tight*, and the *auxiliary network* is defined by the tight arcs. We denote the set of tight arcs by Z, and the auxiliary network by $D_Z = (N, Z)$.

6.2.1 Preliminaries: A Generalization of Wong's Algorithm

We first discuss the dual-ascent algorithm with the values of the u_{ij} variables fixed at value 0. We refer to this part of the dual-ascent algorithm as Stage 1. Keeping u_{ij} fixed at value 0 is tantamount to assuming that the formulation does not contain constraint (6.2d). In that case, Formulation (6.2) is structurally very similar to the directed flow formulation for the Steiner branching problem (model (3.7) on page 66); Formulation (6.2) contains additional commodities with the root node as their destination.

This observation suggests a simple dual-ascent strategy to obtain a network that satisfies all the constraints except (6.2d). For any one of the commodities $k \in K$, identify and perform a basic dual-ascent step on a loose $O(k)$-$D(k)$ dicut. (Recall, a loose $O(k)$-$D(k)$ dicut is an $O(k)$-$D(k)$ dicut that does not contain any tight arcs, and a basic dual-ascent step increases the w_{ij}^k values for all arcs in a loose dicut until the slack on one of the arcs becomes zero. By Menger's theorem, if the auxiliary network does not contain a directed path from $O(k)$ to $D(k)$, then the digraph contains a loose $O(k)$-$D(k)$ dicut.) Continue in this fashion until for every commodity k, every $O(k)$-$D(k)$ dicut contains a tight arc. Each step of this strategy improves the dual objective value, and at termination the auxiliary network contains a path from the root node to every node with a connectivity requirement and a path from every node with a connectivity requirement of 2 to the root node.

We consider two dual-ascent strategies for Stage 1. Recall (from Chapter 3), if $\delta^-(S)$ is

a loose $O(k)$-$D(k)$ dicut for some commodity k, we refer to S as an *ascent set.* A *minimal ascent set* is an ascent set that contains no subset of nodes that is an ascent set.

The first dual-ascent strategy is similar to the dual-ascent algorithm for the minimum cost branching problem and the dual-ascent algorithm for the Steiner branching problem. In this approach we identify and perform dual-ascent steps on minimal ascent sets in the auxiliary network until the digraph contains no more ascent sets.

In the second strategy, we initially consider only minimal ascent sets that do not contain the root node. This approach is equivalent to performing Wong's algorithm for the Steiner branching problem on the digraph with the terminal node set $T\backslash\{\text{root node}\}$ (and with the root as the root node). At the conclusion of this procedure, we obtain a network that contains a directed path from the root node to every node with a connectivity requirement. We next perform dual-ascent steps so that the the auxiliary network contains a directed path from every node with a connectivity requirement of 2 to the root node. That is, we ensure that the transpose of the auxiliary network contains a Steiner branching with terminal node set $T_2\backslash\{\text{root node}\}$. We say a set of nodes S is a *transpose minimal ascent set* if S is a minimal ascent set in D^T, the transpose[3] of the digraph D. That is, $\delta^+(S)$ (the arcs directed out of S in the digraph D) contains no tight arc and no subset of S contains an outgoing tight arc. We now perform a basic dual-ascent step on the loose dicut $\delta^-(N\backslash S)$ defined by the *complement* of S.[4] In the dual-ascent algorithm, we perform basic dual-ascent steps on the complement of transpose minimal ascent sets until the digraph contains no more transpose minimal ascent sets.

There is considerable flexibility in implementing either of the two strategies. In Section 6.3 we will discuss several implementation approaches.

6.2.2 Twinless Strongly Connected Components

In this section we characterize the auxiliary network at the end of Stage 1 and use this characterization to develop a second stage for a dual-ascent algorithm. At the conclusion of Stage 1, if we use either of the two dual-ascent strategies outlined in the previous section, the auxiliary network contains a directed path from the root to every node with a connectivity requirement and from every node with a connectivity requirement of 2 to the root node. As a result, the nodes of T_2 belong to the same strongly connected component of the auxiliary network. Does the auxiliary network always satisfy the conditions of Robbins' theorem? (Remember, typically if the auxiliary network violates some primal constraint, it is possible to ascend on the dual.)

[3] D^T, the transpose of D, is the digraph $D^T = (N, A^T)$ defined by the arc set $A^T = \{(i,j) : (j,i) \in A\}$.

[4] Observe that the set $N\backslash S$ might not be a minimal ascent set. This illustrates the difference between the two strategies.

Figure 6-2: Twin paths.

As the example in Figure 6-2 shows, the answer to this question is no. In this example, nodes a and b have a connectivity requirement of 2. The auxiliary network has a path from node a to node b and a path from node b to node a. However, if we replace the directed arcs by their undirected counterparts, (i.e., replace (i,j), (j,i), or both by a single edge $\{i,j\}$), the network is not 2-edge connected. We refer to the pair of paths from node a to node b and from node b to node a as *twin paths*. In general, we say that two paths P and P' are *twin paths* if P is a path from node i to node j, and P' is a path from node j to node i that uses the reversal of each arc on the path P.

A digraph $D = (N, A)$ is a *Twinless Strongly Connected Digraph* (TSCD) if for some subset A' of A, the digraph (N, A') is strongly connected and does not contain any arc and its twin. A *Twinless Strongly Connected Component* (TSCC) of a digraph is a maximal twinless strongly connected subdigraph of D. Figure 6-3 identifies four TSCCs (that contain 3 or more nodes) in a digraph.

Robbins' theorem (Theorem 6.1.1) says that a graph is 2-edge connected if and only if some orientation of it is strongly connected. The statement of the theorem has an implicit assumption, namely that the digraph contains no arc (i,j) and its twin. Robbins' theorem implies that if the auxiliary network satisfies the connectivity requirements for the problem, then *it must have a TSCC that contains the root node and all other nodes that have a connectivity requirement of 2.* We will try to find violations to this characterization in the auxiliary network and use the violations to improve the dual objective function.

We first derive some structural properties of TSCCs in a strongly connected digraph D. We will use these properties to design a dual-ascent procedure and efficient implementations of the procedure. For ease of exposition, we introduce some additional notation. The *TSCC induced subdigraph of D* is the digraph D^* obtained by contracting each TSCC in D to a single node. We replace parallel arcs that the contraction creates by a single arc. For any digraph $D = (N, A)$, the *associated undirected graph* $G = (N, E)$ is a graph with edges $E = \{\{i,j\} : (i,j) \in A \text{ and/or } (j,i) \in A\}$. If (i,j) belongs to A, we refer to the edge $\{i,j\}$ in E as an *image* of this arc.

We first prove a useful property concerning the structure of directed paths between TSCCs in a strongly connected digraph.

Theorem 6.2.1 (Twin-arc) *Let $D = (N, A)$ be any strongly connected digraph and let D^* be the TSCC induced subgraph of D. The associated undirected graph of D^* is a tree. Moreover, every edge in the associated tree is the image of a pair of twin arcs (and no other*

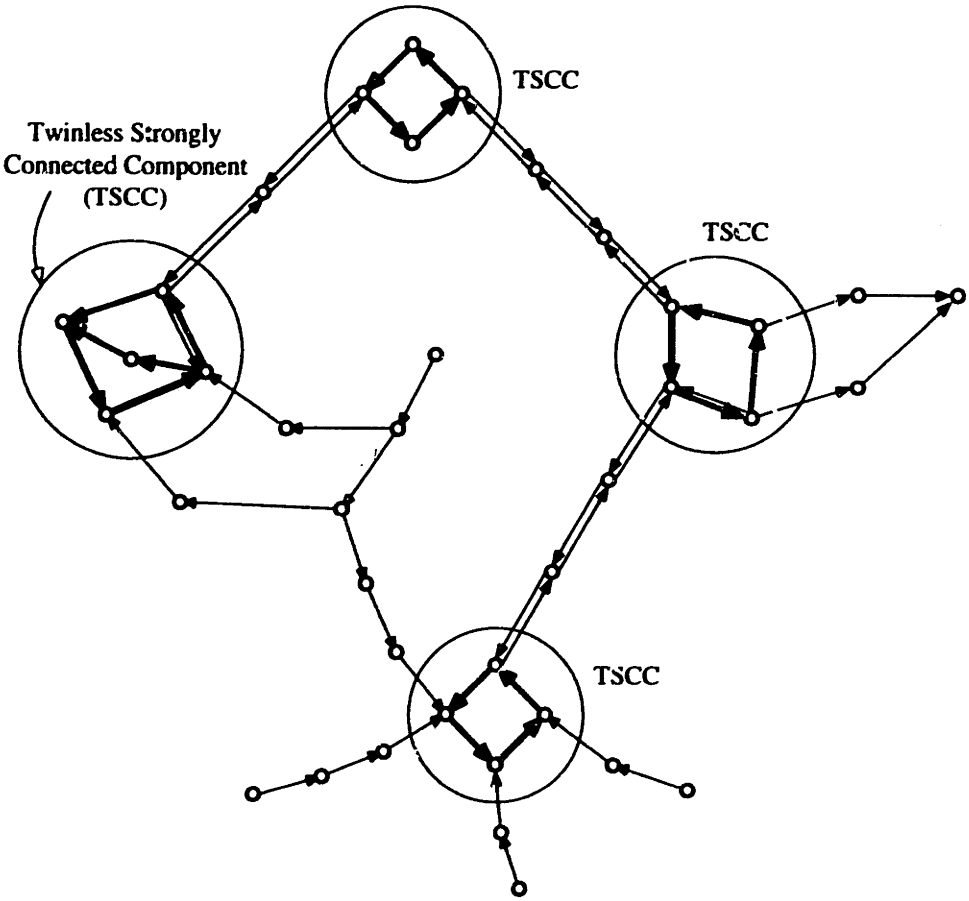


Figure 6-3: Twinless Strongly Connected Components. Bold arcs show twinless arcs that form a strongly connected component.

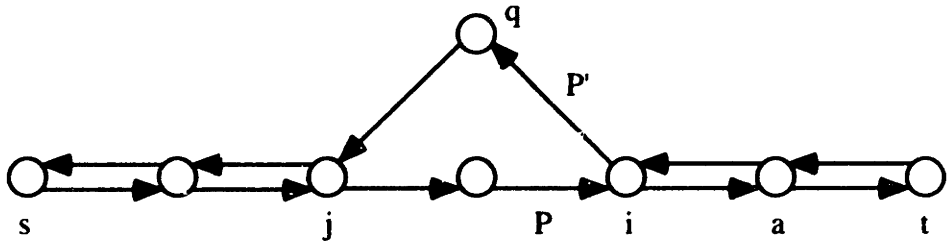


Figure 6-4: Proof that the paths P and P' are twin paths.

arcs) of D.

Proof: First, consider the TSCC induced subdigraph D^* (note that since D is strongly connected, so is D^*). We show that D^* contains a twin path and no other path between any two nodes. As a result, the associated undirected graph of D^* is a tree. We will establish this result by contradiction. Assume the digraph D^* contains a path P from a node s to a node t and a path P' from node t to node s that are not twin paths. Let arc (i,q) be the first arc on P' that does not have a twin arc on the path P and let j be the first node following node i on the path P' that lies on the path P (see Figure 6-4). Then all nodes on P' between nodes i and j and all nodes on P between nodes j and i form a TSCC. But, a TSCC in the contracted digraph corresponds to a TSCC in the original digraph, and so D^* is not the TSCC induced subdigraph of D. Thus, we have a contradiction.

We now show that every pair of twin arcs in D^* corresponds to a pair of twin arcs and no other arcs of D. As a result, every edge in the associated tree is the image of a pair of twin arcs and no other arcs of D. Consider any two adjacent nodes in D^*, say nodes a and t. Node t and node a correspond to TSCCs (possibly single nodes) in the expanded digraph (i.e., D). We refer to the TSCC node a corresponds to as TSCC(a), and the TSCC node t corresponds to as TSCC(t). If the original (expanded) digraph contains two nontwin arcs (i,j) with $i \in$ TSCC(a) and $j \in$ TSCC(t) and (k,l) with $k \in$ TSCC(t) and $l \in$ TSCC(a), then TSCC(a), TSCC(t) and the arcs (i,j) and (k,l) form a larger TSCC, a contradiction. Therefore, only a single arc (a,t) connects TSCC(a) to TSCC(t) and only a single arc, the twin of arc (a,t), joins TSCC(t) to TSCC(a). ■

Since D^*, the TSCC induced subdigraph of the strongly connected digraph D, has the structure of a bidirected tree (that is, a tree with twin arcs in place of each edge [see Figure 6-5]); we refer to D^* as the *TSCC tree.* (Each node of this tree corresponds to a TSCC in D). We also refer to every TSCC that corresponds to a leaf node of the TSCC tree as a *leaf TSCC*, and refer to the arcs (a,b) and (b,a) directed into and out of a leaf TSCC in D^* as *twin leaf arcs.*

Figure 6-5b shows the TSCC tree of the strongly connected digraph shown in Figure 6-

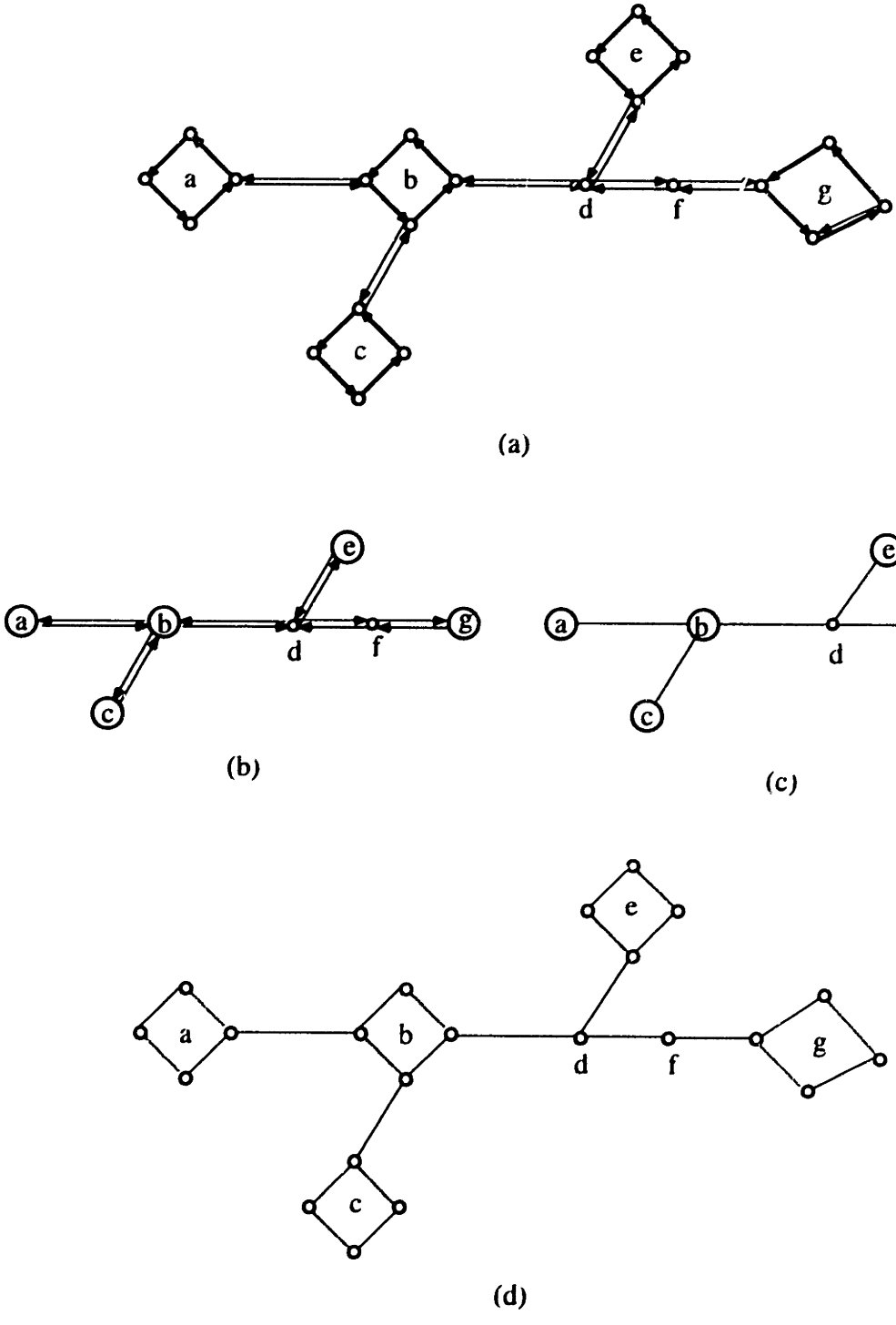


Figure 6-5: Illustration of Theorem 6.2.1, Theorem 6.2.2, and Corollary 6.2.3. (a) Strongly connected digraph D, (b) D^* the TSCC induced subdigraph of D (the TSCC tree), (c) Associated undirected graph of D^*, (d) Associated undirected graph of D.

5a. In this figure, nodes a, c, e and g correspond to leaf TSCCs. The pair of arcs (a, b) and (b, a) are twin leaf arcs in the TSCC tree.

Theorem 6.2.1 implies the following result concerning the relationship between a strongly connected digraph and its associated undirected graph.

Theorem 6.2.2 *The associated undirected graph G of a strongly connected digraph D is 2-edge connected if and only if D is a twinless strongly connected digraph.*

Proof: If D is a twinless strongly connected digraph, then its associated undirected graph G is 2-edge connected. Otherwise, deleting some edge from G disconnects the graph. In D, this edge corresponds to some arc or that arc and its twin. But since eliminating these arcs disconnects D, it is not twinless strongly connected, a contradiction.

To complete the proof, we show that if the associated undirected graph G is 2-edge connected, then D is a twinless strongly connected digraph. We claim the TSCC tree is a single node. If the TSCC tree contained 2 or more nodes, then its associated undirected graph (a tree) has at least one edge. But then G is not 2-edge connected (deleting this edge from the associated undirected graph of the TSCC tree disconnects G), a contradiction. However, the TSCC tree is a single node only when D is a twinless strongly connected digraph. ■

Theorem 6.2.1 and Theorem 6.2.2 imply the following characterization of TSCCs in a strongly connected digraph.

Corollary 6.2.3 *The 2-edge-connected components of the associated undirected graph G of a strongly connected digraph D correspond in a one to one fashion with the TSCCs of D.*

6.2.3 Dual-Ascent and TSCCs

In this section we show how to perform dual-ascent using TSCCs. Suppose we identify all TSCCs in the auxiliary network. We refer to any TSCC that contains a node with a connectivity requirement of 2 as an *admissible TSCC.*

We can now improve the dual objective as follows. Consider the strongly connected component, which we denote by D_S, of the auxiliary network that contains the nodes of T_2. Assume D_S contains more than one admissible TSCC; otherwise, the auxiliary network satisfies the constraints of Formulation (6.2). By Theorem 6.2.1, D_S contains a pair of twin arcs[5] (a, b) and (b, a) whose deletion divides it into two strongly connected components. Moreover, we can choose this pair of twin arcs so that both the resulting strongly connected components contain an admissible TSCC. Let D_S^{ab} and D_S^{ba} denote the

[5] These are the twin arcs of the TSCC tree.

two strongly connected components of D_S obtained by deleting the arcs (a,b) and (b,a). The arc (a,b) is directed into D_S^{ab} and the arc (b,a) is directed into D_S^{ba}.

Let S_{ba} be the set of all nodes, including the nodes of D_S^{ba}, that have a directed path to any node in D_S^{ba} in the auxiliary network after we delete arc (b,a) (recall that the auxiliary network $D_Z = (N, Z)$ contains more nodes and arcs than D_S). Similarly, let S_{ab} be the set of all nodes, including the nodes of D_S^{ab}, that have a directed path to D_S^{ab} in the auxiliary network[6] after we delete arc (a,b). Let $Q = \{\delta^-(S_{ba}) \cup \delta^-(S_{ab})\} \backslash \{(a,b),(b,a)\}$ be the set of all arcs other than arcs (a,b) and (b,a) that are directed into the sets S_{ab} and S_{ba}.

Without loss of generality, assume the root node is in S_{ba} and let t be a node from T_2 in S_{ab}. Let k and h be the commodities with origin node t and destination node t. Note that, from equation (6.3c) *even though arcs* (a,b) *and* (b,a) *are tight we can increase both* w_{ab}^k *and* w_{ba}^h *by* ϵ *for any pair of commodities* k *and* h *by increasing* u_{ab} *by* ϵ.

Suppose we increase w_{ij}^k by ϵ for all arcs in $\delta^-(S_{ba})$ and w_{ij}^h by ϵ for all arcs in $\delta^-(S_{ab})$. Then, in order to maintain feasibility, we need to increase u_{ab} by ϵ. We have thus increased the shortest path lengths ($v_{D(k)}^k$ and $v_{D(h)}^h$) for both commodities k and h by ϵ, and the dual objective increases by ϵ (since u_{ab} has a negative coefficient in the objective function). We can improve the dual in this fashion using the sets S_{ba} and S_{ab} until an arc in Q becomes tight. At this point, any further increase in the dual objective using the sets S_{ba} and S_{ab} is no longer possible.

In this dual-ascent step, we increase w_{ij}^k by ϵ for all arcs in $\delta^-(S_{ba})$ and w_{ij}^h by ϵ for all arcs in $\delta^-(S_{ab})$. Observe that the slack of an arc in Q that is in $\delta^-(S_{ba}) \cap \delta^-(S_{ab})$ decreases by 2ϵ, whereas the slack of an arc in Q that is in exactly one of the two dicuts $\delta^-(S_{ba})$ and $\delta^-(S_{ab})$ decreases by ϵ. Consequently, we define the *effective slack* of arc (i,j) in $\delta^-(S_{ba}) \cap \delta^-(S_{ab})$ as $s_{ij}/2$, and the effective slack of arc (i,j) in exactly one of the two dicuts $\delta^-(S_{ba})$ and $\delta^-(S_{ab})$ as s_{ij}. The arc in Q that becomes tight in the dual-ascent step is the one with minimum effective slack.

Since it is possible to perform dual-ascent by considering the pair of sets S_{ab} and S_{ba}, and they are defined by deleting twin arcs (a,b) and (b,a), we refer to the sets S_{ab} and S_{ba} as *twin ascent sets.* Notice that a pair of twin ascent sets satisfies the following properties: (i) they contain a node from T_2, (ii) they have a single tight arc directed into them, and (iii) moreover, the tight arc directed into one of the sets is the twin of the arc directed into the other set. We refer to the dual-ascent step on the twin ascent sets as a *twin dual-ascent step.*

In Stage 2 of the dual-ascent algorithm, we perform twin dual-ascent steps on twin ascent sets until the network has no more twin ascent sets. This situation occurs when all

[6]Some nodes could be in both set S_{ba} and set S_{ab}. However, observe that no node in T_2 belongs to both set S_{ba} and set S_{ab} since this situation would violate Theorem 6.2.1.

nodes with a connectivity requirement of 2 are in a single TSCC. At the conclusion of these dual-ascent steps, the auxiliary network contains a digraph that satisfies the constraints in Formulation (6.2), and the dual solution provides a lower bound on the optimal solution to the NDLC problem.

Minimal Twin Ascent Sets (Leaf TSCC Rule)

We now discuss a rule for selecting twin ascent sets in Stage 2 that is reminiscent of the minimal ascent set rule used in previous chapters. In the dual-ascent algorithm we consider only twin ascent sets S_{ab} and S_{ba} that are *minimal*, in the sense that one of the sets S_{ab} and S_{ba} is minimal; it does not contain any member set of a twin ascent set pair as a subset. We call sets S_{ab} and S_{ba} *minimal twin ascent sets* and refer to this scheme as the *minimal twin ascent set rule.*

The following property relates leaf TSCCs to twin minimal ascent sets.

Property 6.2.4 *Consider the strongly connected component D_S of the auxiliary network that contains the nodes of T_2. Iteratively delete (peel of) leaf nodes that are not admissible TSCCs from the TSCC tree, so that every leaf TSCC in the resulting TSCC tree is an admissible TSCC. Then twin ascent sets S_{ab} and S_{ba} obtained by deleting twin leaf arcs (a, b) and (b, a) of this tree are minimal twin ascent sets. Furthermore, all minimal twin ascent sets of the digraph can be obtained in this fashion.*

Proof: First, observe from Theorem 6.2.1 that deleting a leaf TSCC that is not admissible does not destroy the twin path between two admissible TSCCs in the TSCC tree. Consequently, none of the twin arcs deleted from the original TSCC tree defines a pair of twin ascent sets.

Therefore, from now on we assume that all leaf TSCCs of the TSCC tree are admissible. Observe that deleting any pair of twin arcs of this TSCC tree splits D_S into two strongly connected components, both of which contain TSCCs. As a result, (deleting) every pair of twin arcs of the TSCC tree defines a pair of twin ascent sets. We now show that deleting only leaf twin arcs defines minimal twin ascent sets, proving the assertion of the theorem.

Suppose we delete twin leaf arcs (a, b) and (b, a). Without loss of generality assume, that TSCC(a) (the TSCC that contains node a in the TSCC tree) is a leaf TSCC. S_{ba} is defined as the set of all nodes, including the nodes of D_S^{ba}, that have a path to any node in D_S^{ba} in the auxiliary network after we delete arc (b, a). We claim that S_{ba} is minimal. This assertion is true because S_{ba} contains only one TSCC, TSCC(a).

Now suppose we delete a pair of twin arcs (a, b) and (b, a) that are not twin leaf arcs. Then, the resulting twin ascent set pair, S_{ab} and S_{ba}, is not minimal. Since (a, b) and (b, a) are not twin leaf arcs, S_{ba} and S_{ab} each contain at least two TSCCs from the TSCC tree.

Furthermore, both the twin ascent sets S_{ab} and S_{ba} contains at least one (admissible) leaf TSCC. By considering the twin leaf arcs adjacent to a leaf TSCC in S_{ab} we obtain a pair of twin ascent sets, and one is a subset of S_{ab}. Similarly, by considering the twin leaf arcs adjacent to a leaf TSCC in S_{ba}, we obtain a pair of twin ascent sets, and one is a subset of S_{ba}. Thus, S_{ab} and S_{ba} are not minimal twin ascent sets. ■

Since every leaf TSCC of the TSCC tree (we assume every leaf TSCC of the tree is admissible) characterizes a pair of minimal twin ascent sets, the twin leaf arcs adjacent to the TSCC define the minimal twin ascent sets, we also refer to the minimal twin ascent set rule as the *leaf TSCC rule.*

These results provide us with all the required ingredients for creating a dual-ascent algorithm for the NDLC problem. However, there are still a number of issues to be resolved: in Section 6.3, we discuss implementation issues; in Section 6.4 we describe methods for obtaining a primal solution. We conclude this section by providing a formal description of the dual-ascent algorithms and then illustrating them with an example.

6.2.4 Dual-Ascent Algorithms

In Stage 1, *Dual-ascent algorithm 1* performs basic dual-ascent steps on minimal ascent sets.

Dual-ascent algorithm 1

1. **(Initialization)** Set
   (a) $w_{ij}^k = 0$ for all $(i,j) \in A$ and $k \in K$.
   (b) $s_{ij} = c_{ij}$ for all $(i,j) \in A$.
   (c) $Z = \{(i,j) : (i,j) \in A; s_{ij} = 0\}$ (the set of zero-slack arcs).
   (d) $u_{ij} = 0$ for all $\{i,j\} \in E$.
   (e) $v_i^k = 0$ for all $i \in N$ and $k \in K$.
   (f) $L = \sum_{k \in K} v_{D(k)}^k - \sum_{\{i,j\} \in E} u_{ij}$ (the lower bound).
2. **(Stage 1)** While the digraph contains a minimal ascent set S,
   (a) Select a node $l \in S$ that has a connectivity requirement and identify the commodity k associated with it (i.e., $D(k) = l$, $O(k) \notin S$). Set $N(k)$ to be the set of nodes that have a directed path to node l.
   (b) Find minimum slack.
   - $\epsilon = \min_{\{(i,j):(i,j) \in \delta^-(N(k))\}} s_{ij}$.

(c) Update relevant dual variables, slacks, the auxiliary network $D_Z = (N, Z)$, and the lower bound L.

- $w_{ij}^k = w_{ij}^k + \epsilon \quad \forall\ (i,j) \in \delta^-(N(k))$,
- $s_{ij} = s_{ij} - \epsilon \quad \forall\ (i,j) \in \delta^-(N(k))$,
- $v_i^k = v_i^k + \epsilon \quad \forall\ i \in N(k)$,
- $L = L + \epsilon$,
- $Z = Z \cup \{(i,j) : (i,j) \in \delta^-(N(k)) \text{ and } s_{ij} = 0\}$.

3. **(Stage 2)** While the auxiliary network D_Z contains a pair of minimal twin ascent sets S_{ab} and S_{ba},

(a) Without loss of generality, assume the root node belongs to S_{ab} and let t be a node in T_2 that belongs to S_{ba}. Select commodities k and h with $D(k) = t$ ($O(k) =$ root) and $O(h) = t$ ($D(h) =$ root).

(b) Find minimum effective slack.

- $\bar{s}_{ij} = \begin{cases} \frac{s_{ij}}{2} & \text{if } (i,j) \in \delta^-(S_{ab}) \cap \delta^-(S_{ba}), \\ s_{ij} & \text{otherwise,} \end{cases}$
- $\epsilon = \min_{\{(i,j):(i,j)\in\{\delta^-(S_{ba})\cup\delta^-(S_{ab})\}\setminus\{(a,b),(b,a)\}\}} \bar{s}_{ij}$.

(c) Update relevant dual variables, the auxiliary network D_Z, and the lower bound L.

- $w_{ij}^k = w_{ij}^k + \epsilon \quad \forall\ (i,j) \in \delta^-(S_{ba})$,
- $w_{ij}^h = w_{ij}^h + \epsilon \quad \forall\ (i,j) \in \delta^-(S_{av})$,
- $\bar{s}_{ij} = \bar{s}_{ij} - \epsilon \quad \{\delta^-(S_{ba}) \cup \delta^-(S_{ba})\}\setminus\{(a,b),(b,a)\}$,
- $s_{ij} = \begin{cases} 2\bar{s}_{ij} & \text{if } (i,j) \in \delta^-(S_{ab}) \cap \delta^-(S_{ba}), \\ \bar{s}_{ij} & \text{otherwise,} \end{cases}$
- $u_{ab} = u_{ab} + \epsilon$,
- $v_i^k = v_i^k + \epsilon \quad \forall\ i \in (S_{ba})$,
- $v_i^h = v_i^h + \epsilon \quad \forall\ i \in (S_{ab})$,
- $L = L + \epsilon$,
- $Z = Z \cup \{(i,j) : (i,j) \in \{\delta^-(S_{ba}) \cup \delta^-(S_{ab})\}\setminus\{(a,b),(b,a)\} \text{ and } s_{ij} = 0\}$.

The second algorithm, *Dual-ascent algorithm 2*, differs from *Dual-ascent algorithm 1* in Stage 1. *Dual-ascent algorithm 2* first considers minimal ascent sets that do not contain the root and performs basic dual-ascent steps on them. When the digraph has no more ascent sets that do not contain the root, the algorithm considers transpose minimal ascent sets and performs basic dual-ascent steps on the complement of these sets.

Dual-ascent algorithm 2

1. **(Initialization)** Identical to *Dual-ascent algorithm 1.*

2. **(Stage 1)**

   (a) **(Stage 1a)** While the digraph contains a minimal ascent set S that does not contain the root node,

   i. Select a node $l \in S$ that has a connectivity requirement and identify the commodity k associated with it (i.e., $D(k) = l$). Set $N(k)$ to be the set of nodes that are reachable in the auxiliary network on a directed path from node l.

   ii. Find minimum slack.

   - $\epsilon = \min_{\{(i,j):(i,j)\in\delta^-(N(k))\}} s_{ij}$.

   iii. Update relevant dual variables, slacks, the auxiliary network D_Z, and the lower bound L.

   - $w_{ij}^k = w_{ij}^k + \epsilon \quad \forall\ (i,j) \in \delta^-(N(k))$,
   - $s_{ij} = s_{ij} - \epsilon \quad \forall\ (i,j) \in \delta^-(N(k))$,
   - $v_i^k = v_i^k + \epsilon \quad \forall\ i \in N(k)$,
   - $L = L + \epsilon$,
   - $Z = Z \cup \{(i,j) : (i,j) \in \delta^-(N(k)) \text{ and } s_{ij} = 0\}$.

   (b) **(Stage 1b)** While the digraph contains a transpose minimal ascent set S,

   i. Select a node $l \in S$ that has a connectivity requirement and identify the commodity k associated with it (i.e., $O(k) = l$). Set $N(k)$ to be the set of nodes in the auxiliary network that node l has a directed path to.

   ii. Find minimum slack.

   - $\epsilon = \min_{\{(i,j):(i,j)\in\delta^+(N(k))\}} s_{ij}$.

   iii. Update relevant dual variables, slacks, the auxiliary network D_Z, and the lower bound L.

   - $w_{ij}^k = w_{ij}^k + \epsilon \quad \forall\ (i,j) \in \delta^+(N(k))$,
   - $s_{ij} = s_{ij} - \epsilon \quad \forall\ (i,j) \in \delta^+(N(k))$,
   - $v_i^k = v_i^k + \epsilon \quad \forall\ i \in N \backslash N(k)$,
   - $L = L + \epsilon$,
   - $Z = Z \cup \{(i,j) : (i,j) \in \delta^+(N(k)) \text{ and } s_{ij} = 0\}$.

3. **(Stage 2)** Identical to *Dual-ascent algorithm 1.*

In these algorithms, if our interest is only in the value of the dual objective, that is the lower bound, we need not store the **v** and **w** variables. The slacks and u_{ij} values of the

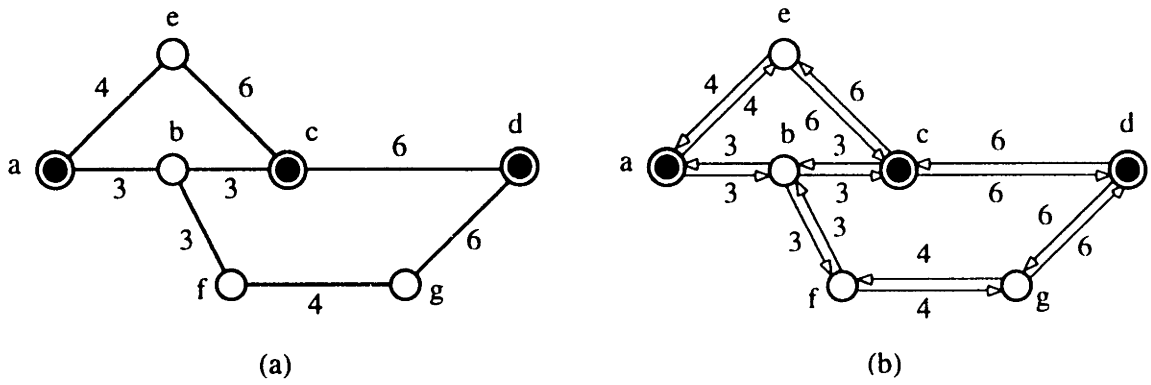


Figure 6-6: Data for dual-ascent example. (a) Nodes a, c and d have connectivity requirement 2. (b) The undirected graph converted into a directed graph.

arcs provide the necessary information to update the lower bound, auxiliary networks, slack and u_{ij} values. Consequently, we can store the slacks, u_{ij} variables, and digraph data in $\mathcal{O}(|N| + |A|)$ space.

6.2.5 Example

To illustrate the two dual-ascent algorithms, consider the example shown in Figure 6-6. In this example, nodes a, c and d have a connectivity requirement of 2, and in the flow model we select node a as the root node.

Figure 6-7 illustrates *Dual-ascent algorithm 1.* The algorithm first performs Stage 1. The sets $\{a\}$, $\{c\}$ and $\{d\}$ are minimal ascent sets. Stage 1 begins by performing a basic dual-ascent step on $\{a\}$. As a result, arc (b, a) becomes tight. The dual objective increases by 3 units to $L = 3$. The algorithm then performs a basic dual-ascent step on the minimal ascent set $\{c\}$. As a result, arc (b, c) becomes tight, and the dual objective increases by 3 units to $L = 6$. The algorithm then performs a basic dual-ascent step on minimal ascent step $\{d\}$. As a result, arcs (c, d) and (g, d) become tight and the dual-objective increases by 6 units to $L = 12$. Figure 6-7a shows the digraph at the conclusion of these iterations. At this point, the sets $\{a, b\}$ and $\{c, b\}$ are minimal ascent sets. The algorithm performs a basic dual-ascent step on the minimal ascent set $\{a, b\}$. Arc (e, a) becomes tight and the dual objective increases by 1 unit to $L = 13$. The dual-ascent algorithm then performs a basic dual-ascent step on the minimal ascent set $\{c, b\}$. Arc (f, b) becomes tight and the dual objective increases by 2 units to $L = 15$. Figure 6-7b shows the digraph at the conclusion of these two iterations. At this juncture, sets $\{a, b, e, f\}$ and $\{c, b, f\}$ are minimal ascent sets. The dual-ascent algorithm performs a basic dual-ascent step on the minimal ascent set $\{a, b, e, f\}$. Arc (c, b) becomes tight and the dual objective increases by 2 units to $L = 17$. The dual-ascent algorithm then performs a basic dual-ascent step on the minimal ascent

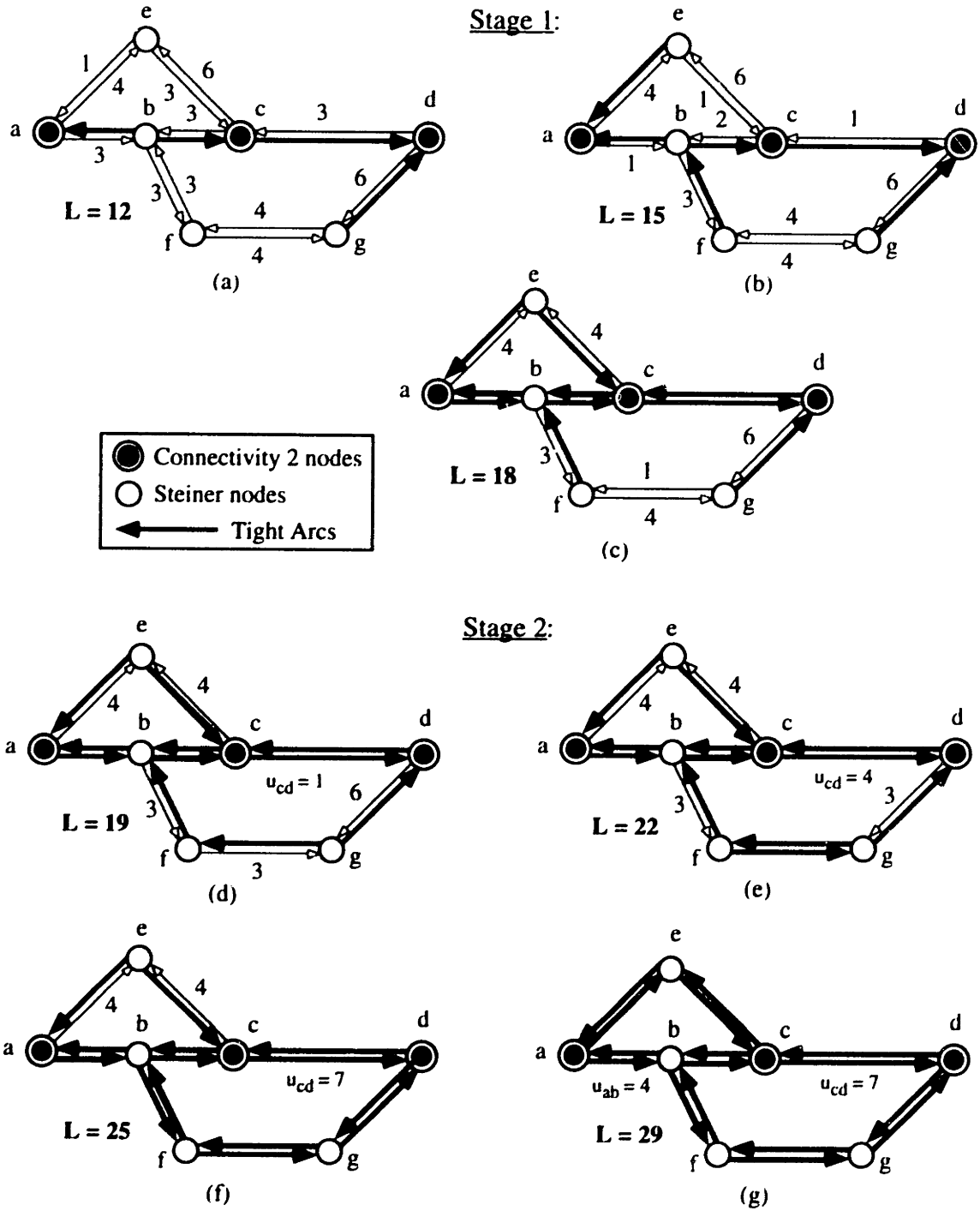


Figure 6-7: Example of *Dual-ascent algorithm 1* for NDLC problem. Numbers next to arcs are slacks; bold arcs have zero slack.

set $\{c, b, f\}$. Arcs (a, b), (e, c) and (d, c) become tight, and the dual objective increases by 1 unit to $L = 18$. Figure 6-7c shows the digraph at the conclusion of these two iterations. At this point, the network has no more ascent sets. Therefore, Stage 1 is complete and the auxiliary network contains a strongly connected component that contains nodes a, c, and d.

The dual-ascent algorithm now performs Stage 2. At the start of Stage 2, the auxiliary network contains three admissible TSCCs—$\{a\}$, $\{c\}$ and $\{d\}$. TSCCs $\{a\}$ and $\{d\}$ satisfy the leaf TSCC rule. The dual-ascent algorithm selects admissible TSCC(d),[7] and identifies the twin leaf arcs (c, d) and (d, c). Therefore, it sets $S_{cd} = \{d, g\}$ and $S_{dc} = \{a, b, c, f\}$ and then performs a twin dual-ascent step on this twin ascent set pair. As a result arc (g, f) becomes tight, and the dual objective increases by 1 unit to $L = 19$. Figure 6-7d shows the digraph at the conclusion of this iteration. TSCCs $\{a\}$ and $\{d\}$ satisfy the leaf TSCC rule. The dual-ascent algorithm selects TSCC(d) and identifies the twin leaf arcs (c, d) and (d, c). It sets $S_{cd} = \{d, g\}$ and $S_{dc} = \{a, b, c, f, g\}$ and then performs a twin dual-ascent step on this twin ascent set pair. As a result, arc (f, g) becomes tight, and the dual objective increases by 3 units to $L = 22$. Figure 6-7e shows the digraph at the conclusion of this iteration. At this point, TSCCs $\{a\}$ and $\{d\}$ satisfy the leaf TSCC rule. The dual-ascent algorithm selects TSCC(d) and identifies the twin leaf arcs (c, d) and (d, c). It then sets $S_{cd} = \{d, g, f\}$ and $S_{dc} = \{a, b, c, f, g\}$ and performs a twin dual-ascent step considering this twin ascent set pair. As a result, arcs (b, f) and (d, g) become tight, and the dual objective increases by 3 units to $L = 25$. Figure 6-7f shows the digraph at the conclusion of this iteration. At this point, TSCCs $\{a\}$ and $\{b, c, d, f, g\}$ satisfy the leaf TSCC rule. The dual-ascent algorithm selects admissible TSCC(a) and identifies the leaf twin arcs (b, a) and (a, b). The dual-ascent algorithm sets $S_{ba} = \{a, e\}$ and $S_{ab} = \{b, c, d, e, f, g\}$ and then performs a twin dual-ascent step on this twin ascent set pair. As a result, arcs (a, e) and (c, e) become tight and the dual objective increases by 4 units to $L = 29$. Figure 6-7g shows the digraph at the conclusion of this iteration.

At this point, the nodes of D form a single TSCC. As a result, the dual-ascent algorithm terminates with a lower bound of 29 units.

Figure 6-8 illustrates *Dual-ascent algorithm 2.* The algorithm first performs Stage 1a. In this stage, it considers minimal ascent sets that do not contain the root node, namely the sets $\{c\}$ and $\{d\}$. The algorithm first performs a basic dual-ascent step on minimal ascent set $\{c\}$. As a result, arc (b, c) becomes tight and the dual objective increases by 3 units to $L = 3$. The algorithm then performs a basic dual-ascent step on minimal ascent step $\{d\}$. Arc (c, d) and (g, d) become tight and the dual-objective increases by 6 units to $L = 9$. Figure 6-8a shows the digraph at the conclusion of these two iterations. At this point, the set $\{c, b\}$ is the only minimal ascent that does not contain the root. The

[7] Again, we let TSCC(i) denote the TSCC that contains node i.

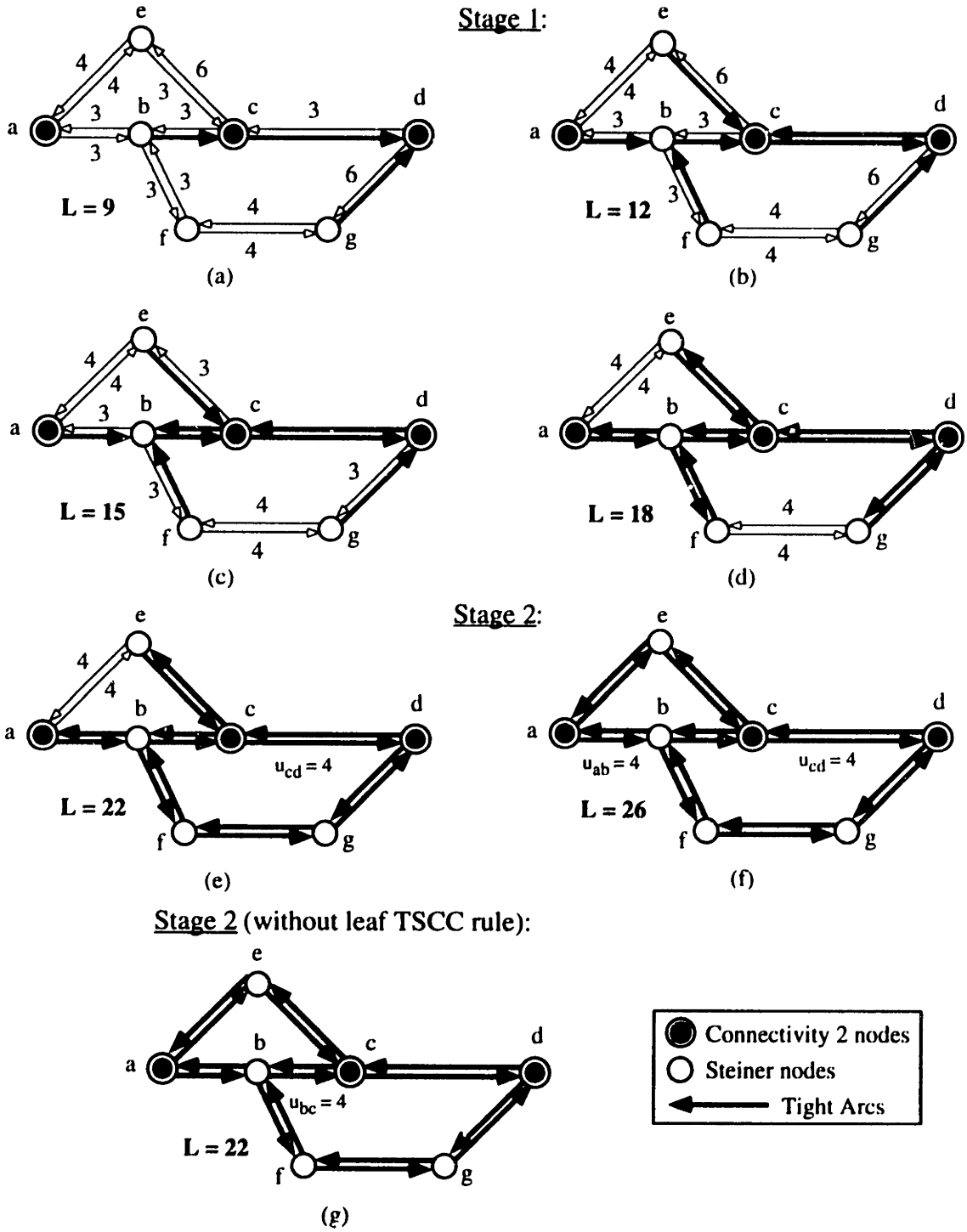


Figure 6-8: Example of *Dual-ascent algorithm 2* for NDLC problem.

dual-ascent algorithm performs a basic dual-ascent step on the minimal ascent set $\{c, b\}$. As a result, arcs (a, b), (f, b), (e, c) and (d, c) become tight and the dual objective increases by 3 units to $L = 12$. Figure 6-8b shows the digraph at the conclusion of this iteration. At this juncture, every ascent set contains the root node.

The algorithm now performs Stage 1b. In this stage, the dual-ascent algorithm identifies transpose minimal ascent sets. The only transpose minimal ascent set is $\{c, d\}$. The dual-ascent algorithm performs a basic dual-ascent step on the complement of set $\{c, d\}$, i.e., $\{a, b, e, f, g\}$. The arc (c, b) becomes tight and the dual objective increases by 3 units to $L = 15$. Figure 6-8c shows the digraph at the conclusion of this iteration. At this point, the only transpose minimal ascent set is $\{b, c, d\}$. The dual-ascent algorithm performs a basic dual-ascent step on the complement of set $\{b, c, d\}$, i.e., $\{a, e, f, g\}$. As a result, the arcs (b, a), (c, e), (b, f) and (d, g) become tight. The dual objective increases by 3 units to $L = 18$. Figure 6-8d shows the digraph at the conclusion of these two iterations. At this point, the network contains no transpose minimal ascent sets. Therefore, Stage 1b is complete, and the auxiliary network contains a strongly connected component that contains nodes a, c and d.

The dual-ascent algorithm now performs Stage 2. At the start of Stage 2, the auxiliary network contains three admissible TSCCs—$\{a\}$, $\{c\}$ and $\{d\}$. TSCCs $\{a\}$ and $\{d\}$ satisfy the leaf TSCC rule. The dual-ascent algorithm selects admissible TSCC(d) and identifies the twin leaf arcs (c, d) and (d, c). The dual-ascent algorithm sets $S_{cd} = \{d, g\}$ and $S_{dc} = \{a, b, c, f\}$ and then performs a twin dual-ascent step considering this twin ascent set pair. As a result, arcs (g, f) and (f, g) become tight, and the dual objective increases by 4 units to $L = 22$. Figure 6-8e shows the digraph at the conclusion of this iteration. At this point, TSCCs $\{a\}$ and $\{b, c, d, f, g\}$ satisfy the leaf TSCC rule. The dual-ascent algorithm selects admissible TSCC(a) and identifies the twin leaf arcs (a, b) and (b, a). The dual-ascent algorithm identifies the sets $S_{ba} = \{a\}$ and $S_{ab} = \{b, c, d, e, f, g\}$. It then performs a twin dual-ascent step considering this pair of twin ascent sets. As a result, arcs (a, e) and (e, a) become tight, and the dual objective increases by 4 units to $L = 26$. Figure 6-8f shows the digraph at the conclusion of this iteration.

At this point the nodes of D form a single TSCC. As a result the dual-ascent algorithm terminates with a lower bound of 26 units.

This example also illustrates the effectiveness of the leaf TSCC rule. Suppose at the start of Stage 2, *Dual-ascent algorithm 2* deletes twin arcs (b, c) and (c, b) (note, they are not twin leaf arcs). The dual-ascent algorithm sets $S_{bc} = \{e, c, d, g\}$ and $S_{cb} = \{a, b, f\}$, and performs a twin dual-ascent step on this twin ascent set pair. As a result, arcs (a, e), (e, a), (f, g) and (g, f) become tight and the dual objective increases by 4 units to $L = 22$. The auxiliary network is feasible to the problem. Therefore, the lower bound is less than that

provided by the leaf TSCC rule. Figure 6-8g shows the auxiliary network at the conclusion of Stage 2 without the leaf TSCC rule.

This example also seems to indicate that *Dual-ascent algorithm 1* provides a better lower bound than *Dual-ascent algorithm 2*. However, as our computational experiments indicate (see Section 6.6), neither algorithm is more effective in all situations.

6.3 Implementing the Dual-Ascent Algorithms

In this section we show how to efficiently implement the dual-ascent algorithms. We show how to identify the TSCC tree in linear time (and thus identify leaf TSCCs in linear time) and then show how to implement the dual-ascent algorithms so that they run in $\mathcal{O}(|A|^2)$ time and $\mathcal{O}(|N| + |A|)$ space. (To avoid confusion we will describe the complexity of the algorithms in terms of the number of arcs $|A|$ in D, instead of the number of edges $|E|$ in G. To obtain the complexity of these algorithms in terms of $|E|$ simply replace $|A|$ by $|E|$ [since $|A| = 2|E|$].)

In this section we draw on many ideas and implementations described in Section 3.4.3 for Wong's dual-ascent algorithm for the Steiner branching problem. Consequently, to avoid repetition, much of our discussion is brief and refers to the appropriate material in Chapter 3.

6.3.1 Stage 1

Dual-Ascent Algorithm 1

In Stage 1 of this dual-ascent algorithm, we perform dual-ascent steps on minimal ascent sets in the digraph. Recall that in Section 3.4.3 we described an algorithm for identifying root components, and used the root components to identify minimal ascent sets.

For convenience we review the definition of a root component for the Steiner branching problem. Recall in the Steiner branching problem, given a directed graph $D = (N, A)$, a root node r, and a set T ($r \notin T$) of terminal nodes, we wish to design a minimum cost network that contains a directed path from the root node r to every node in T. If the auxiliary network contains a directed path from node i to node j and no directed path from node j to node i, then we say that node i *dangles* from node j. A strongly connected component R is a *root component* if it contains a member of T and no member of $T \cup \{r\}$ dangles from a member of R. Let $S_R = \{j$: there is a directed path in D_Z from node j to a node in $R\}$. We showed (in Section 3.4.2) that the set S_R corresponding to root component R is a minimal ascent set. Furthermore, we showed these are the only minimal ascent sets in the digraph.

We now modify slightly the definition of the root component to include strongly con-

```
Root-Components
1.    for i = 0 to |N| − 1
2.          pred[i] = −1;
3.          color[i] = green;
4.          root[i] = 0;
5.    for i = 0 to |N| − 1
6.          if ((color[i] = green) and (req[i]=1))
7.                then root[i] = 1;
8.                      DFS-Visit(i);
```

Figure 6-9: Algorithm to find root components.

nected components that contain the root node. A strongly connected component R is a *root component* if it contains a node from T ($= T_1 \cup T_2$), the set of nodes with a connectivity requirement greater than zero, and no member of T dangles from a member of R. An identical argument to that in Section 3.4.2 shows that the sets S_R associated with root components are minimal ascent sets, and are the only minimal ascent sets in the digraph.

Since the nodes in a root component are strongly connected, it is sufficient to identify the digraph obtained by contracting all strongly connected components. If a strongly connected component in D_Z contains a node from T, then the node is a *required node* in the contracted network. We assume that the contracted network is topologically sorted (recall, a digraph with no strongly connected components can be topologically sorted). We note that algorithm *Root-Components*, which we present again in Figure 6-9 for convenience, identifies all root components in this topologically sorted digraph.

Consequently, we could implement Stage 1 for *Dual-ascent algorithm 1* in four different ways.

Implementation 1: This implementation identifies a single root component and performs a basic dual-ascent step on the minimal ascent set associated with it. The arguments in Section 3.4.3 (on page 76) show that this implementation requires $\mathcal{O}(|A| \min(|T||N|, |A|))$ time and $\mathcal{O}(|N| + |A|)$ space.

Implementation 2: In this implementation, we find minimal ascent sets $S_1, \ldots, S_p$ corresponding to all root components. We then sequentially perform a basic dual-ascent step on each of the sets $S_1, \ldots, S_p$. (The example in Figure 6-7 is similar to this implementation.)

The correctness of this implementation follows from Proposition 3.4.2.[8] Consequently, this implementation requires $\mathcal{O}(|A| \min(|T||N|, |A|))$ time and $\mathcal{O}(|T||N| + |A|)$ space.

Implementation 3: In this implementation we consider all minimal ascent sets simultaneously, and perform an ascent step on them simultaneously. This procedure requires $\mathcal{O}(|T||A| \min(|T||N|, |A|))$ time and $\mathcal{O}(|T||N| + |A|)$ space.

Implementation 4: This implementation is identical to the *round robin rule* presented for the Steiner branching problem. Initially, for each commodity the dual-ascent algorithm sets $N(k) = \{D(k)\}$. In a dual-ascent step on commodity k, the algorithm identifies the arc (i^*, j^*) in $\delta^-(N(k))$ with minimum slack (in case of ties, it selects any one of the arcs with minimum slack). The algorithm then adds *exactly one node, the node i^*, to the set* $N(k)$. We say that a commodity k is *active* if $O(k)$ is not in $N(k)$; otherwise, commodity k is *inactive*. Note that for an inactive commodity k, the auxiliary network contains a path from $O(k)$ to $D(k)$. The algorithm considers the commodities in a fixed order and cycles through the commodities, performing dual-ascent steps for active commodities, in this fixed order. As mentioned in Chapter 3, Balakrishnan, Magnanti and Wong [BMW89] have shown that the round robin rule simulates the minimal ascent set rule for the Steiner branching problem. The same result (with no change to their proof) is also true here. That is, the round robin rule simulates the minimal ascent set rule for Stage 1 of the dual-ascent algorithm. Arguing similarly as in Section 3.4.3 and noting that the number of commodities is at most $2|T|$ shows that this implementation requires $\mathcal{O}(|T||N||A|)$ time and $\mathcal{O}(|T||N| + |A|)$ space.

Dual-Ascent Algorithm 2

This dual-ascent algorithm performs Stage 1 in two parts—Stage 1a and Stage 1b.

First, in Stage 1a, it uses dual-ascent to construct an auxiliary network that has a path from the root node to every other node with a connectivity requirement. Then, in Stage 1b, it uses dual-ascent to ensure that the auxiliary network has a path from every connectivity requirement 2 node to the root.

Stage 1a is equivalent to Wong's algorithm for the Steiner branching problem presented in Section 3.4.2, with the root node as the root and the nodes in $T \backslash \{\text{root node}\}$ as the required nodes. Consequently, we could implement it in any of the four ways described in Section 3.4.3.

Stage 1b is equivalent to Wong's algorithm on the transpose of the digraph and can therefore be implemented identically as Stage 1a. As an example, we briefly describe how

[8] Proposition 3.4.2 asserts that if S is a minimal ascent set, then, after a sequence of basic dual-ascent steps, S is either a minimal ascent set or no longer an ascent set (i.e., it has a tight arc directed into it).

the round robin rule could be implemented for Stage 1b. (To obtain the steps of the round robin rule, we simply transform the operations of the round robin rule from the transpose of the network to the network.) Initially, for each commodity the dual-ascent algorithm sets $N(k) = \{O(k)\}$ for all commodities with $D(k) =$ root node. In a dual-ascent step on commodity k, the algorithm identifies the arc (i^*, j^*) in $\delta^+(N(k))$ with minimum slack (in case of ties, it selects any one of the arcs with minimum slack). The algorithm then adds exactly one node, the node i^*, to the set $N(k)$. The algorithm considers the commodities (with $D(k) =$ root node) in a fixed order and cycles through the commodities, performing dual-ascent steps for active commodities, in this fixed order.

As a result, assuming similar implementations for Stage 1a and Stage 1b,

1. Implementation 1 requires $\mathcal{O}(|A| \min(|T||N|, |A|))$ time and $\mathcal{O}(|N| + |A|)$ space,
2. Implementation 2 requires $\mathcal{O}(|A| \min(|T||N|, |A|))$ time and $\mathcal{O}(|T||N| + |A|)$ space,
3. Implementation 3 requires $\mathcal{O}(|T||A| \min(|T||N|, |A|))$ time and $\mathcal{O}(|T||N| + |A|)$ space, and
4. Implementation 4 requires $\mathcal{O}(|T||N||A|)$ time and $\mathcal{O}(|T||N| + |A|)$ space.

6.3.2 Stage 2

At the end of Stage 1 the auxiliary network contains (i) a directed path from the root node to every other node with a connectivity requirement, and (ii) a directed path from every node with a connectivity requirement of 2 to the root node.

In Stage 2 of the dual-ascent algorithm, we identify minimal twin ascent sets and perform twin dual-ascent steps on them. To identify these minimal ascent sets, we need to identify all the twinless strongly connected components of D_S, the strongly connected component of the auxiliary network that contains the nodes of T_2. Recall, from Section 3.4.3 (see Figure 3-5 on page 74), we can identify all strongly connected components in a digraph in linear time. Therefore we can identify D_S from D_Z (the auxiliary network) in linear time. In this section we describe a linear time algorithm to identify all twinless strongly connected components of D_S.

Identification of Twinless Strongly Connected Components

Recall from Corollary 6.2.3 that to identify the TSCCs of a strongly connected digraph, it is sufficient to identify all 2-edge-connected components of its associated undirected graph. We can convert D_S to its associated undirected graph G_S in $\mathcal{O}(|N| + |A|)$ time. A well-known method for identifying all 2-edge-connected components is based on depth first search (see

exercise 23.2 in [CLR90], or exercise 3.7 in [Eve79]). For completeness, in Figure 6-10 we describe the algorithm.

A *bridge* is an edge whose removal disconnects a graph. Algorithm *Bridges* identifies all bridges in an undirected graph. By deleting the bridges of a graph, we obtain the 2-edge-connected components of a graph. Notice that the bridges correspond to the image of edges of the TSCC tree. As a result, the algorithm *Bridges* contains the necessary information to construct the TSCC tree.

Algorithm *Bridges* performs depth first search. The arrays **pred**, **color**, **start**, and **finish** are the same as in algorithm *DFS* (described in Section 3.4.3). Recall that the **pred** array defines the depth first search forest $D_{\mathbf{pred}} = (N, A_{\mathbf{pred}})$, with

$$A_{\mathbf{pred}} = \{(\mathbf{pred}[i], i) : i \in N \text{ and } \mathbf{pred}[i] \neq -1\}.$$

Note that the depth first forest defines a directed graph even though we perform depth first search on an undirected graph. The forest contains a directed path from the root of each tree to all the nodes in that tree. The **start** array indicates the order in which the *DFS* algorithm first examined the nodes. That is, if $\mathbf{start}[i] < \mathbf{start}[j]$, then the algorithm visits node i before node j. The **low** array provides us with the necessary information to identify those edges that are bridges.

To help the reader understand this algorithm, we briefly sketch a proof that shows *Bridges* is correct. For more details, refer to [Eve79]. For convenience, assume that the graph is connected. In this analysis, we orient the edges $\{i, j\}$ of the graph that do not appear as arcs on the depth first search forest, orienting a nonforest edge $\{i, j\}$ from node i to node j if $\mathbf{start}[i] > \mathbf{start}[j]$ (i.e., orient it in the direction that the depth first search algorithm first traversed edge $\{i, j\}$). Let $L(i)$ denote the set of nodes that can be reached from node i via a directed path on the depth first search forest followed by at most one nonforest arc. Define $\mathbf{low}[i] = \min_{j \in L(i)} \mathbf{start}[j]$. If $\{k, l\}$ is a bridge, then deleting it disconnects the graph. Let $G^k_{\{k,l\}}$ and $G^l_{\{k,l\}}$ denote the two components obtained by deleting bridge $\{k, l\}$: $G^k_{\{k,l\}}$ contains node k and $G^l_{\{k,l\}}$ contains node l. Suppose, in depth first search, we first reach node l by traversing $\{k, l\}$, i.e., $\mathbf{pred}[l] = k$. Then all the nodes in $G^l_{\{k,l\}}$ must be unscanned when we first reach node l; otherwise some edge of the network besides $\{k, l\}$ connects $G^k_{\{k,l\}}$ and $G^l_{\{k,l\}}$. Therefore, $\mathbf{low}[l] = \mathbf{start}[l]$. If $\{k, l\}$ is not a bridge, i.e., it lies on a cycle, one of its ancestors[9] is in $L(l)$. Therefore, $\mathbf{low}[l] < \mathbf{start}[l]$. The algorithm uses this property of the **low** array to identify bridges.

Since the algorithm *Bridges* performs depth first search, it has a worst case running time of $\mathcal{O}(|N| + |E|)$. Constructing the TSCC tree from the algorithm *Bridges* is a linear

[9] An *ancestor* of node i is any node that can reach node i on the depth first forest $D_{\mathbf{pred}}$.

```
Bridges(G)
1.    for i ∈ N
2.          pred[i] = −1;
3.          color[i] = green;
4.    time = 0;
5.    for i ∈ N
6.          if (color[i] = green)
7.                then x =DFS-Visit(i);
8.    Output list of bridges.

DFS-Visit(i)
1.    color[i] = yellow.
2.    time = time + 1;
3.    start[i] = time;
4     low[i] = time;
5.    while i's adjacency list is nonempty
6.          select a node j from i's adjacency list;
7.          if (color[j] = green)
8.                then pred[j] = i;
9.                     low[i] = min(low[i],DFS-Visit(j));
10.               else if ((color[j] = yellow) and (pred[i] ≠ j))
11.                    then low[i] = min(low[i],start[j]);
12.   color[i] = red;
13.   time = time + 1;
14.   finish[i] = time;
15.   if ((pred[i] ≠ −1) and (low[i]=start[i]))
16.         then add {i,pred[i]} to the list of bridges;
15.   return(low[i]).
```

Figure 6-10: Algorithm to find bridges of a graph.

time procedure, requiring $\mathcal{O}(|N| + |A|)$ time. Note that once we have the TSCC tree, we can identify all twin leaf arcs and leaf TSCC's in $\mathcal{O}(|N| + |A|)$ time.

Methods for Implementing Stage 2

We now describe three methods for implementing Stage 2. Note that we can reduce the size of the digraph D in the dual-ascent procedure by contracting every twinless strongly connected component in the auxiliary network D_Z into a single node in the digraph D. If this contraction creates parallel arcs, we retain the arc with the smallest slack. Although contraction does not improve the complexity analysis of any of the implementations we discuss, it might prove to be useful in practice, especially when solving large-scale problems.

Implementation 1: This implementation first identifies a single leaf TSCC on the TSCC tree and performs a twin dual-ascent step on the minimal twin ascent set associated with it. (This is similar to the examples in Figure 6-7 and 6-8.) The algorithm continues in this fashion (it identifies a leaf TSCC (either the same one or a different one) on the TSCC tree and performs a twin dual-ascent step on the minimal twin ascent sets associated with the leaf TSCC) until the TSCC tree is a single node.

We can identify the twin ascent set pair S_{ab} and S_{ba} associated with twin leaf arcs (a, b), and (b, a) as follows. Consider the digraph $D_Z^{ab} = (N, Z \backslash \{(a, b), (b, a)\})$ (i.e., the digraph obtained by deleting arcs (a, b) and (b, a) from D_Z). By performing depth first search from TSCC(a) on the transpose of D_Z^{ab}, we can identify S_{ba}, and by performing depth first search from TSCC(b) on the transpose of D_Z^{ab} we can identify S_{ab}. Therefore, we can identify the minimal twin ascent sets associated with a leaf TSCC in $\mathcal{O}(|A|)$ (since $Z \subseteq A$) time. Performing a twin dual-ascent step requires identifying the arc with minimum effective slack, a computation that requires $\mathcal{O}(|A|)$ time if we sequentially scan all the arcs in the digraph. The dual-ascent algorithm performs at most $|A|$ iterations because, after $|A|$ iterations, all the arcs in the original digraph D are tight. Consequently, the dual-ascent algorithm is an $\mathcal{O}(|A|^2)$ procedure. If we do not store the dual variables $\mathbf{v}$ and $\mathbf{w}$, this implementation requires $\mathcal{O}(|N|+|A|)$ space to store the adjacency lists of the digraph, the auxiliary network, and the slacks. It simultaneously keeps track of two dicuts, and requires $\mathcal{O}(|N|)$ space to do so.

Implementation 2: In this implementation, we find all minimal twin ascent sets (i.e., the twin ascent sets corresponding to every pair of twin leaf arcs). We then sequentially perform a twin dual-ascent step on each of the twin sets. We continue by again finding the TSCC tree, identifying all minimal twin ascent sets, and sequentially performing twin dual-ascent steps on them. The correctness of this implementation follows from the following property.

Proposition 6.3.1 *Suppose S_{ab} and S_{ba} are a pair of minimal twin ascent sets. Then after a sequence of twin dual-ascent steps, S_{ab} and S_{ba} either form a pair of minimal twin ascent sets or are no longer twin ascent sets (i.e., one of S_{ab} and S_{ba} has two tight arcs directed into it).*

Proof: By definition, one of the two set S_{ab} and S_{ba} is minimal. Suppose S_{ab} is minimal. Then all nodes with with a connectivity requirement of 2 in S_{ab} are in TSCC(b). Therefore any twin ascent set that contains TSCC(b) must contain S_{ab}. Consequently as long as S_{ab} and S_{ba} are twin ascent sets, they are a pair of minimal twin ascent sets. ■

If the dual-ascent algorithm attempts to perform a twin dual-ascent step on the pair S_{ab} and S_{ba} and they are not twin ascent sets, then one of them has a tight arc in $Z\backslash\{(a,b),(b,a)\}$ directed into it. The algorithm finds that the minimum effective slack of some arc in $\{\delta^-(S_{ba}) \cup \delta^-(S_{ab})\}\backslash\{(a,b),(b,a)\}$ is zero and makes no change to the dual variables and lower bound.

If the TSCC tree contains p twin leaf arc pairs, then we can identify the minimal twin ascent set pairs corresponding to every pair of twin leaf arcs of the TSCC tree in $\mathcal{O}(p|A|)$ steps (finding the minimal twin ascent set pair for one twin leaf arc pair requires $\mathcal{O}(|A|)$ time). The algorithm now performs p twin dual-ascent steps, one for each of the p minimal twin ascent set pairs, each requiring $\mathcal{O}(|A|)$ steps. Therefore, p iterations require $\mathcal{O}(|A|+p|A|+p|A|)$ steps. So on the average, each of the p iterations requires $\mathcal{O}(|A|)$ steps. Now, arguing similarly as in Implementation 1, we can prove that this implementation requires $\mathcal{O}(|A|^2)$ time. Since this implementation needs to simultaneously keep track of at most $\mathcal{O}(|T_2|)$ dicuts (the maximum number of admissible TSCCs is $|T_2|$, and so the maximum number minimal twin ascent set pairs is $|T_2|$), it requires $\mathcal{O}(|T_2||N|)$ space.

Implementation 3: This implementation identifies the minimal twin ascent sets corresponding to each pair of twin leaf arcs and then performs the dual-ascent steps *simultaneously* on all minimal twin ascent sets. We can determine all minimal twin ascent set pairs in $\mathcal{O}(|T_2||A|)$ steps (the maximum number of minimal twin ascent set pairs is $|T_2|$). To determine the arc that first becomes tight, for each arc we need to determine the rate at which its slack decreases. We can do so by sequentially scanning the arcs and determining how many ascent sets an arc is directed into. Therefore, each iteration requires $\mathcal{O}(|T||A|)$ time. Arguing similarly as in Implementation 1 shows that this implementation is an $\mathcal{O}(|T||A|^2)$ procedure. Since this implementation needs to simultaneously keep track of $\mathcal{O}(|T_2|)$ dicuts, it requires $\mathcal{O}(|T_2||N|)$ space.

In this section we showed how to identify all TSCCs in linear time (and how to identify a single minimal twin ascent set pair in linear time). We then gave several simple implementations for each stage of the dual-ascent algorithm. Our analysis shows that if

we use Implementation 1 for both Stage 1 and Stage 2, the dual-ascent algorithm requires $\mathcal{O}(|N|+|A|)$ space and $\mathcal{O}(|A|^2)$ time. However, we believe that by using more sophisticated data structures and analysis, it might be possible to improve the running times of these algorithms.

6.4 Primal Heuristics

At the conclusion of the dual-ascent procedure, the auxiliary network D_Z contains a solution that is feasible to Formulation (6.2). It might be possible to delete some arcs from the auxiliary network and retain a feasible solution to Formulation (6.2).

We consider two classes of heuristics—drop heuristics and exchange heuristics.[10] In our implementation we first perform drop heuristics to obtain a feasible solution (that is minimal), and then apply the improvement heuristics to this solution.

6.4.1 Drop Heuristics

In this section we describe two LIFO drop heuristics. Recall that in a LIFO drop heuristic, we sequentially consider the arcs in the opposite order that they became tight (i.e., the opposite order that the algorithm adds them to the auxiliary network).

The first heuristic sequentially considers the arcs of the auxiliary network in a LIFO order and deletes them if their deletion does not create an infeasible solution. However, because the NDLC problem is an undirected problem, we might be able to improve on this solution. Figure 6-11 shows a solution obtained by using a LIFO drop rule. In this example, nodes c and d have a connectivity requirement of 2. Observe that the only directed path from node c to node d includes the path from node b to node a, and the only directed path from node d to node c includes the path from node b to node a. However, in the undirected network, we can delete the path from node b to node a. Therefore, at the end of the directed LIFO drop heuristic, we consider the associated undirected graph of the heuristic solution and sequentially delete edges in any order, deleting an edge if it retains the solution's feasibility.

In the second LIFO drop heuristic, we consider the associated undirected graph of the auxiliary network and perform a LIFO drop heuristic on it. To perform the LIFO drop heuristic, we construct the LIFO list as follows. We replace arc (a,b) or (b,a) or both on the LIFO list by edge $\{i,j\}$. If the auxiliary network contains some twin arcs, then some edges will appear twice on the LIFO list. We can consider deleting these edges based on when the algorithm added the former of (i,j) and (j,i) to the auxiliary network or added

[10] An exchange heuristic is a local search heuristic that considers a feasible solution and improves it by a combination of adding and deleting edges from the solution.

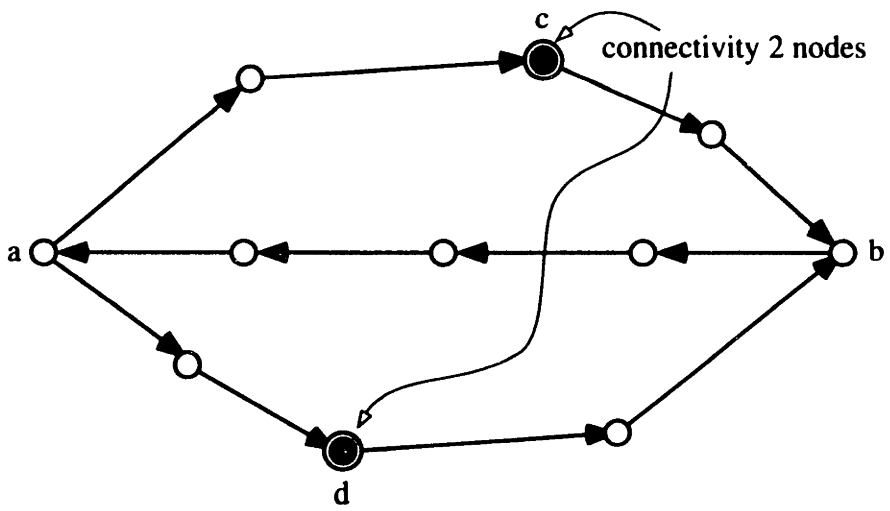


Figure 6-11: Nodes c and d are connected via the path from b to a in the directed graph. The arcs on the path from b to a can be deleted if the network is made undirected.

the latter of these arcs to the network. We refer to these procedures as the LIFO first and LIFO last rules.

To implement either drop heuristic, we need to check if the solution is feasible when we drop an arc (or edge). To do so, we can use the algorithm *Bridges* to ascertain whether all nodes with a connectivity requirement of 2 belong to the same TSCC, and the algorithm *DFS* on the directed graph to ensure that the network contains a path from the root node to every node with a connectivity requirement of 1. As a result, we can check feasibility of a solution in $\mathcal{O}(|A|)$ time, thus implementing the drop heuristic in $\mathcal{O}(|A|^2)$ time (since we can drop no more than $|A|$ arcs).

6.4.2 Exchange Heuristics

At the conclusion of either of the two drop heuristics, the resulting network design is a minimal undirected network satisfying the connectivity requirements (i.e., if we delete an edge from the network design, it is no longer feasible). To obtain further improvements, we consider exchange heuristics. We adapt and apply some of the improvement heuristics—one-opt, two-opt and two-for-one—that Monma and Shallcross [MS89] have developed considered heuristics for the NDLC problem (actually, they consider the node connectivity version, but their methods are also applicable for the edge-connectivity version of the NDLC problem).

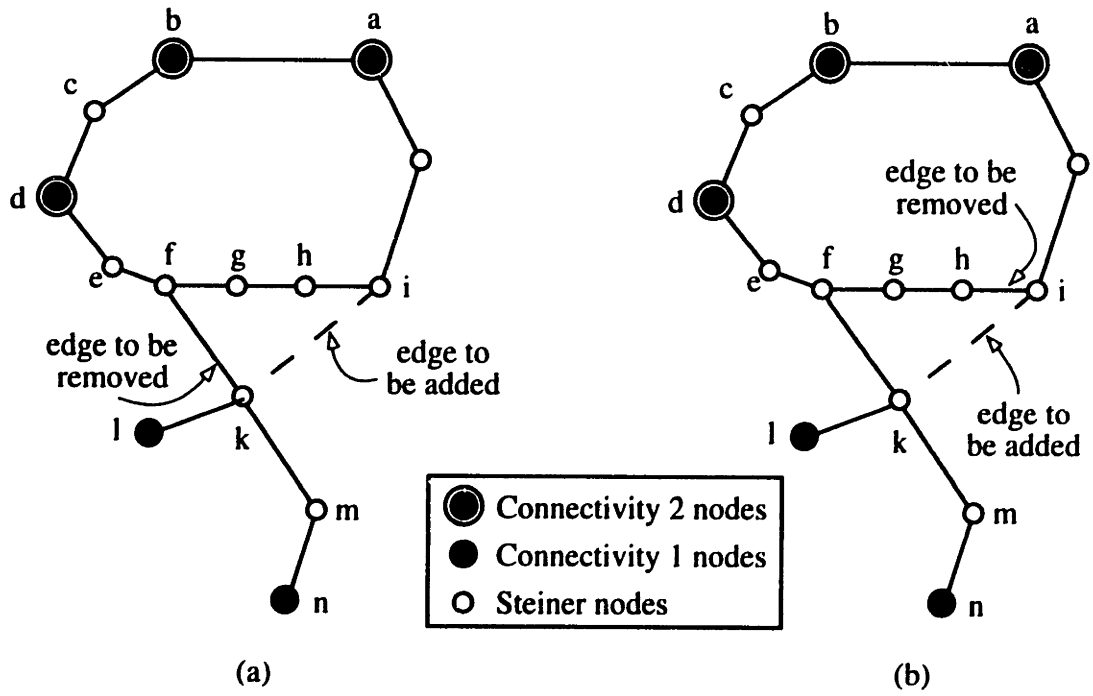


Figure 6-12: One-opt exchange.

One-Opt Heuristic

The one-opt heuristic attempts to remove an edge $\{x, y\}$ from the solution and replace it by another edge $\{x, z\}$ not in the solution. We make the replacement if the resulting network is feasible and has a lower cost. Figure 6-12a illustrates a one-opt exchange. Sometimes, as illustrated in Figure 6-12b, a one-opt exchange creates a solution that is not minimal. That is, if we remove edge $\{i, h\}$ from the network and add edge $\{i, k\}$, then we can also delete edges $\{f, g\}$ and $\{g, h\}$. To delete such edges, we use a drop heuristic (i.e., drop edges in any order, deleting an edge if it retains the solution's feasibility) after applying a prespecified number of successful one-opt changes.

Two-for-One

In this heuristic, we replace two "back to back" cycles by a single cycle if the resulting solution has a lower cost. Figure 6-13a illustrates a two-for-one exchange heuristic. In this example, we remove the edges $\{a, b\}$ and $\{e, h\}$, replacing them by a single edge $\{b, h\}$.

Two-Opt

In this heuristic, we replace two edges $\{a, b\}$ and $\{c, d\}$ in a cycle by the crossing edges $\{a, d\}$ and $\{b, c\}$ to form a new cycle (if the resulting solution is cheaper). Figure 6-13b illustrates

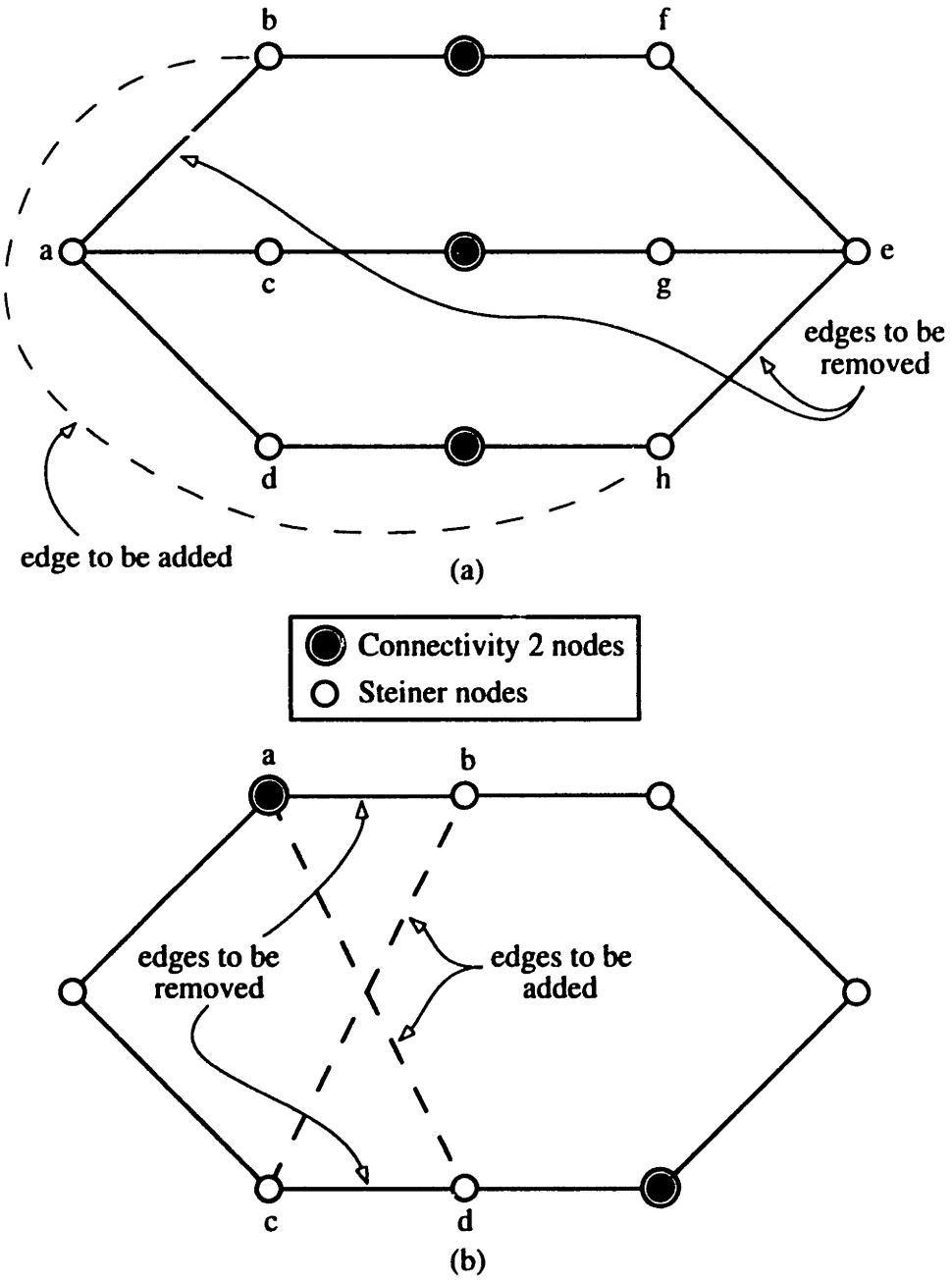


Figure 6-13: (a) Illustration of a two-for-one exchange. (b) Illustration of a two-opt exchange.

a two-opt exchange.

6.5 Extensions

To this point, we have considered a model that prohibits edge replication. With replication, a design could use some edges two or more times. However, problems with nonnegative edge costs always have an optimal solution that does not replicate an edge more than twice since we can delete additional copies of the edge (beyond two) while retaining the solution's feasibility. Therefore, to model the NDLC problem with edge replication, we delete constraints (6.1d) and (6.2d) from models (6.1) and (6.2) for any edge $\{i, j\}$ that is replicable.

What are the consequences of this change to the dual-ascent algorithms? If an edge $\{i, j\}$ is replicable in the undirected problem, a solution to the directed problem that contains twin arcs (i, j) and (j, i) is feasible. Therefore, in the dual-ascent algorithm, if the auxiliary network contains twin arcs (i, j) and (j, i) and edge $\{i, j\}$ is replicable, we contract nodes i and j. (If this contraction creates parallel arcs, we retain the arc with the smallest slack.) With this change, the dual-ascent applies to the NDLC problem that permits edge replication. We note that both the drop heuristics and exchange heuristics are applicable to this problem.

Although the NDLC problem is undirected, we have developed the dual-ascent algorithm on a directed network. Consequently, this dual-ascent algorithm is also applicable to the Steiner 1-arc-connected subdigraph problem (a directed network design problem).[11] This problem can be formulated in a similar fashion to Formulation 6.2, except without constraint (6.2d) (i.e., $y_{ij} + y_{ji} \leq 1$).[12] Consequently, the dual-ascent algorithm is applicable to this problem as well. Recall that the auxiliary network at the conclusion of Stage 1 satisifies all of the primal constraints except $y_{ij} + y_{ji} \leq 1$. Since the formulation for the Steiner 1-arc-connected subdigraph problem does not contain this constraint, the auxiliary network at the conclusion of Stage 1 is a feasible solution for this problem. Therefore, we do not need to apply Stage 2 of the dual-ascent algorithm for this problem. Of the primal heuristics we described only the directed LIFO drop heuristic is applicable to the Steiner 1-arc-connected subdigraph problem; the others were designed specifically for undirected problems.

[11] In the Steiner 1-arc-connected subdigraph problem, we are given a directed network $D = (N, A)$, arc costs, and a set of nodes T that must be strongly connected. We need to design a minimum cost strongly connected network that contains the nodes in T.

[12] Select some node in T as the root node and create commodities as follows. For each node i from T, other than the root node, create two commodities: one with the root node as the origin, node i as the destination, and a requirement of 1; and one with node i as the origin, the root node as the destination, and a requirement of 1.

6.6 Computational Experiments

In this section we report on the application of the dual-ascent procedure to a set of NDLC problems.

We implemented both *Dual-ascent algorithm 1* and *Dual-ascent algorithm 2* as well as the primal heuristics. To implement Stage 1 of both dual-ascent algorithms, we used Balakrishnan, Magnanti and Wong's round robin rule. To implement Stage 2 of both dual-ascent algorithms, we used Implementation 1. That is, we identified a single TSCC that satisfies the leaf TSCC rule and performed a dual-ascent step on the pair of sets associated with this TSCC. To obtain a primal solution, we first applied the two LIFO heuristics. We then applied the one-opt heuristic, the two-for-one heuristic, and the two-opt heuristic sequentially to the solution of the two LIFO heuristics (that is, we applied the two-for-one heuristic to the solution of the one-opt heuristic, applied the two-opt heuristic to the solution of the two-for-one heuristic and so on). We repeated this sequence (one-opt, two-for-one, and two-opt) until the method no longer improved the solution. We coded our algorithms in C, and performed all our tests on a Sun SPARCstation 10 (Model 41) workstation (a 96 MIPS machine).

We would have liked to test our dual-ascent algorithm on some real-world problems. However, because problems in the telecommunications industry appear to be classified, we instead tested the dual-ascent algorithm and primal heuristics on some randomly generated problems with similar characteristics to existing telecommunication network design problems. From this experience, we hope to learn how the dual-ascent method performs and to classify its strengths and weaknesses.

6.6.1 Random Problem Generator

Our random problem generator creates NDLC test problems. It uses the following four input parameters: the number of nodes $|N|$, the number of edges $|E|$, the number of nodes with a connectivity requirement of two $|T_2|$, and the number of nodes with a connectivity requirement of one $|T_1|$. It then randomly selects $|N|$ nodes on a 100×100 grid and arbitrarily creates $|E|$ edges between these nodes. To ensure that the problems are feasible, it also creates a cycle that contains all the nodes. The generator randomly selects the nodes with a connectivity requirement of 1, and the nodes with a connectivity requirement of 2. If the edge costs are to be Euclidean, then the cost of an edge $\{i, j\}$ on this graph is the integer part of the Euclidean distance between nodes i and j. Otherwise, the cost of an edge is a random integer between 1 and 100. We also used this generator to test our dual-ascent algorithm on the Steiner tree problem (as reported in Section 3.4.3).

Nodes	Edges	$\|T_1\|$	$\|T_2\|$
36	65	12	24
77[a]	112	65	14
39	71	31	8
46	98	36	10
116	173	108	8
116	173	108	8
116	173	84	32

[a]Table I from [GMS92a] contains this incorrect data ($65 + 14 \neq 77$).

Table 6.1: Grötschel, Monma and Stoer's [GMS92a] test data.

6.6.2 Results

We limited our computational tests to problems with Euclidean edge costs, since telecommunication applications typically have this structure.

Test Problems that Resemble Telecommunication Data

We attempted to generate test problems that closely resembled those in the telecommunications industry. Grötschel, Monma and Stoer [GMS92a] solved seven connectivity problems for the Regional Bell Operating Companies. Table 6.1 shows the data for these seven problems. In these test problems the edge costs were Euclidean.

To simulate these test problems, we generated problems with 40 nodes and 80 edges, and problems with 120 nodes and 180 edges. In these problems, all the nodes have a nonzero connectivity requirement. For the 40 node problems, we varied the fraction of nodes in T_2 from 20 to 60 percent, and for the 120 node problems, we varied the fraction of nodes in T_2 from 10 to 30 percent.

Tables 6.2, 6.3 and 6.4 summarize our computations on problems with these characteristics. The first table summarizes the computational performance of *Dual-ascent algorithm 1* and the primal heuristics. The second summarizes the computational performance of *Dual-ascent algorithm 2* and the primal heuristic. The third combines the results of *Dual-ascent algorithm 1* and *Dual-ascent algorithm 2*. It selects the best upper bound and best lower bound generated by the two dual-ascent algorithms. The columns *LIFO1* and *LIFO2* indicate the performance of the two LIFO heuristics (for LIFO2 we used the LIFO first rule). The column *Best* shows the performance of the best primal solution generated (i.e., the best solution after the exchange heuristics were applied). The column *Running Time* measures the running time in CPU seconds of the algorithm. It includes the time for input, output, the dual-ascent phase, and the primal heuristics. For the combined algorithm in Table 6.4,

the running time is the sum of the running times of *Dual-ascent algorithm 1* and *Dual-ascent algorithm 2*. The columns *Ave Gap* and *Max Gap* measure the gap between the upper and lower bounds, calculated as $\frac{\text{Upper bound} - \text{Lower bound}}{\text{Lower bound}} \times 100$.

The results indicate that the average gaps of LIFO2 seem to be marginally better than those of LIFO1. For several problem instances, the maximum gap of LIFO2 exceed those of LIFO1. *Dual-ascent algorithm 2* performs slightly better than *Dual-ascent algorithm 1*. The computational performance of the combined algorithm is worth noting. Both the average gaps and the maximum gap values are significantly lower than *Dual-ascent algorithm 1*, and a little lower than *Dual-ascent algorithm 2*. Typically, one algorithm generates a better upper bound and the other algorithm generates a better lower bound. For the 40 node problems, the average gaps vary from 5 percent to 11 percent, and the maximum gaps vary from 9 percent to 16 percent. For the 120 node problems, the average gaps vary from 4 percent to 8 percent, and the maximum gaps vary from 5 percent to 10 percent. The error gap increases as the fraction of nodes in T_2 increases. The running time for these problems, whose structure is typical of telecommunication applications, varied from about 3 seconds for 40 node problems to about 30 seconds for 120 node problems.

To obtain a better understanding of the error gaps, we solved some of the test problems with an LP solver. We developed a GAMS model for Formulation (6.2) and solved the linear programs with CPLEX version 2.1 on a Sun SPARCstation 10 (Model 41). The linear programming models are fairly large. For instance the mixed integer programming model of a problem with 40 nodes, 80 edges, $|T_1| = 16$, and $|T_2| = 24$, contains as many as 8320 continuous variables, 160 binary variables, and 10480 constraints.[13] We solved only the 40 node problems using GAMS/CPLEX because CPLEX was unable to find an optimal solution (to the LP) for the 120 node problems in a reasonable amount of computational time (24 hours). We used the dual-simplex method to solve the LPs, since it was 10-15 times faster than the simplex method for these problems. Table 6.5 shows the results of this study. The running time for GAMS/CPLEX is quite large. The average running time varies from around 290 seconds to 1372 seconds, and increases as the fraction of nodes in T_2 increases. The columns *Ave Gap* and *Max Gap* for the *Heuristic-LP Gap* measure the gap between the heuristic solution and LP solution value, calculated as $\frac{\text{Upper bound} - \text{LP solution value}}{\text{LP solution value}} \times 100$. The columns *Ave Gap* and *Max Gap* for the *LP-Dual-Ascent Gap* measure the gap between the lower bound generated by the dual-ascent algorithm and the LP solution value, calculated as $\frac{\text{LP solution value} - \text{Lower bound}}{\text{LP solution value}} \times 100$. The average *Heuristic-LP Gap* varies from less than 2 percent to 3.5 percent, while the maximum gap varies from about 4 percent to 9 percent. These results show that although the error gaps between the primal

[13] In general model (6.2) contains $2|E|$ binary variables, $2|E|(2(|T_2| - 1) + |T_1|)$ continuous flow variables, and $|E| + (2(|T_2| - 1) + |T_1|)(2|E| + |N|)$ constraints.

Size	Terminal Nodes	Trials	LIFO1		LIFO2		Best		Running Time[a]	
$(\|N\|, \|E\|)$	$(\|T_1\|, \|T_2\|)$		Ave Gap	Max Gap	Ave Gap	Max Gap	Ave Gap	Max Gap	Ave Time	Max Time
(40,80)	(32,8)	10	7.10%	10.03%	6.59%	10.43%	6.29%	9.03%	1.4	1.8
	(24,16)	10	12.91%	19.12%	12.25%	19.65%	11.44%	18.13%	1.35	1.82
	(16,24)	10	14.35%	20.80%	13.98%	20.80%	13.86%	20.80%	1.42	1.71
(120,180)	(108,12)	10	4.77%	7.52%	4.71%	6.60%	4.35%	6.13%	9.62	13.89
	(96,24)	10	7.82%	10.53%	7.50%	8.89%	7.10%	8.64%	13.78	24.25
	(84,36)	10	9.64%	11.45%	9.48%	12.22%	9.03%	11.45%	16.68	23.36

[a]Seconds on a Sun SPARCstation 10 (Model 41).

Table 6.2: Computational experiments with *Dual-ascent algorithm 1.*

Size	Terminal Nodes	Trials	LIFO1		LIFO2		Best		Running Time[a]	
$(\|N\|, \|E\|)$	$(\|T_1\|, \|T_2\|)$		Ave Gap	Max Gap	Ave Gap	Max Gap	Ave Gap	Max Gap	Ave Time	Max Time
(40,80)	(32,8)	10	5.30%	8.77%	5.54%	10.44%	4.65%	8.77%	1.34	1.64
	(24,16)	10	11.17%	19.78%	10.39%	19.78%	9.61%	18.07%	1.83	2.29
	(16,24)	10	12.52%	18.60%	11.38%	14.34%	11.38%	14.34%	1.54	2.22
(120,180)	(108,12)	10	4.27%	6.38%	4.26%	6.01%	3.88%	5.20%	10.36	13.78
	(96,24)	10	7.44%	11.13%	7.01%	8.93%	6.58%	8.67%	13.67	20.04
	(84,36)	10	9.08%	11.79%	8.98%	10.64%	8.42%	10.86%	18.29	22.80

[a]Seconds on a Sun SPARCstation 10 (Model 41).

Table 6.3: Computational experiments with *Dual-ascent algorithm 2.*

Size	Terminal Nodes	Trials	Combined Algorithm		Running Time	
$(\|N\|, \|E\|)$	$(\|T_1\|, \|T_2\|)$		Ave Gap	Max Gap	Ave Time	Max Time
(40,80)	(32,8)	10	4.56%	8.77%	2.74	3.4
	(24,16)	10	8.73%	16.50%	3.18	3.79
	(16,24)	10	11.25%	14.18%	2.96	3.79
(120,180)	(108,12)	10	3.79%	5.20%	19.98	27.67
	(96,24)	10	6.39%	8.51%	27.45	44.49
	(84,36)	10	8.17%	10.52%	34.96	44.96

Table 6.4: Combining *Dual-ascent algorithm 1* and *Dual-ascent algorithm 2*.

Terminal Nodes $(\|T_1\|, \|T_2\|)$	Trials	GAMS/CPLEX		Heuristic-LP Gap		LP-Dual-Ascent Gap	
		Ave Time	Max Time	Ave Gap	Max Gap	Ave Gap	Max Gap
(32,8)	10	290.56	507.27	1.75%	3.91%	2.66%	4.55%
(24,16)	10	934.84	1753.33	3.23%	8.85%	5.02%	6.64%
(16,24)	10	1372.45	1913.83	3.43%	5.09%	7.00%	9.39%

Table 6.5: Comparison of LP solution with lower and upper bounds of 40 node problem.

feasible solution and the dual-ascent algorithm are substantial (over 11 percent on average for one problem class), the heuristic solution is typically within 4 percent of optimality. The *LP-Dual Ascent Gap* is larger than the *Heuristic-LP Gap*. It varies from 3 percent to 7 percent increasing as the fraction of nodes in T_2 increases. We will discuss this issue further in the next section.

Other Computational Experiments

We performed two other computational experiments. In the first, we varied the fraction of Steiner nodes, i.e., nodes with a connectivity requirement of 0. In these tests, we generated problems with 100 nodes and a 1000 edges and varied the fraction of Steiner nodes from 25 percent to 75 percent. In each of these problems, we held the ratio $|T_2|/|T_1|$ fixed at 0.10. Table 6.6 summarizes these computations. The average error gap is close to 4 percent, and increases slightly as the number of Steiner nodes increases. The maximum error gap increases from 5 percent to 14 percent.[14] The running time of the algorithm decreases significantly as the number of Steiner nodes increases. We believe that our use of the round robin rule implementation of Stage 1 leads to this behavior. We will elaborate on this issue shortly.

Our second experiment varied the problem size. For these experiments, we generated

[14]Only one problem had an error gap greater than 10 percent.

Size	Steiner Nodes	Trials	Combined Algorithm		Running Time	
$(\|N\|, \|E\|)$			Ave Gap	Max Gap	Ave Time	Max Time
	25	10	3.72%	5.63%	23.67	44.11
(100,1000)	50	10	4.04%	5.29%	16.67	27.42
	75	10	4.14%	14.84%	10.29	17.04

Table 6.6: Performance as a function of the number of Steiner nodes.

Size	Terminal Nodes	Trials	Combined Algorithm		Running Time	
$(\|N\|, \|E\|)$	$(\|T_1\|, \|T_2\|)$		Ave Gap	Max Gap	Ave Time	Max Time
(100,1000)	(90,10)	10	3.92%	6.13%	26.60	49.91
(200,2000)	(180,20)	10	3.59%	4.82%	174.45	287.55
(300,3000)	(270,30)	10	4.17%	6.00%	957.91	1416.53

Table 6.7: Performance as a function of problem size.

test problems with 100 to 300 nodes, and 1000 to 3000 edges. For each problem, T_2 contained 10 percent of the nodes, and the problem contains no Steiner nodes. Table 6.7 summarizes these computations. The running time of the algorithm increases significantly from 26 seconds for the 100 node problems to 957 seconds for 300 node problems. The average error gap was around 4 percent and the maximum error gap was around 8 percent. There does not appear to be any noticeable relationship between the average error gap and the problem size. The error gaps decreased slightly from the 100 to 200 node problems and increased slightly from the 200 to 300 node problems. The mixed integer programming models for these problems are huge. For example, the mixed integer programming model of the 300 node problems contain 6000 binary constraints, 1,968,000 continuous flow variables, and 2,069,400 constraints. As an experiment, we attempted to solve the LP relaxations of one of these models using GAMS/CPLEX. The model was so large, however, that the problem exceeded the memory capacity (i.e., the RAM) of the workstation.

6.6.3 Comments

In their computational work, Grötschel, Monma and Stoer were able to successfully solve the problems shown in Table 6.1 to optimality. They used the underlying network structure to reduce the problem size. For example, many parts of the telecommunication networks they considered were trees, and thus could be "forced" into the solution (i.e., fixed). Consequently, they were able to reduce the size of the problems shown in Table 6.1. For example, they reduced the 116 node problem to a problem with 39 nodes and 86 edges. Our test problems do not have this structure because we ensure that all the nodes of the graph lie

on a cycle. Also, except for a few crossing edges, the networks they considered were usually planar. Thus, the "real" problems have far richer structure that we could conceivably exploit.

We now elaborate on two issues. First, we consider the round robin rule. For convenience, we restate the rule as applied to *Dual-ascent algorithm 1*. Initially, for each commodity the dual-ascent algorithm sets $N(k) = \{D(k)\}$. In a dual-ascent step for commodity k, the algorithm identifies the arc (i^*, j^*) in $\delta^-(N(k))$ with the minimum slack (in case of ties, it selects any one of the arcs with the minimum slack). The algorithm then adds *exactly one node, the node i^*, to the set $N(k)$*. The algorithm considers the commodities in a fixed order and cycles through the commodities, performing dual-ascent steps for active commodities in this fixed order.

Consider a root component, and let L be the set of active commodities whose destination is in this root component. For the moment, assume that $N(k) = N(l)$ for any $k, l \in L$. When the dual-ascent algorithm performs a dual-ascent step on the first commodity it considers from the set L, say commodity k, it selects an arc (i^*, j^*) in $\delta^-(N(k))$ with the minimum slack, reduces its slack to zero, and adds node i^* to $N(k)$. In subsequent steps, for each active commodities $l \in L$, the algorithm finds that $\delta^-(N(l))$ has a tight arc (i^*, j^*) directed into it. Therefore, it performs a dual-ascent step that does not change the dual variables, and only adds node i^* to $N(l)$. We refer to a dual-ascent iteration that does not change the dual variables as a *degenerate dual-ascent iteration*. The algorithm continues to perform degenerate dual-ascent steps until $N(k) = N(l)$ for all $k, l \in L$.

The dual-ascent algorithm performs degenerate dual-ascent iterations in another scenario. Suppose the dual-ascent algorithm adds $D(l)$ to $N(k)$ for some pair of commodities k and l (we are not assuming that their destinations lie in the same root component). Then, if $N(l)$ contains some nodes that are not in $N(k)$, the algorithm performs degenerate dual-ascent iterations on $N(k)$ until all the nodes that $N(l)$ contains are in $N(k)$. (This is true because $D(l) \in N(k)$, and so the auxiliary network contains a directed path from all the nodes in $N(l)$ to $D(k)$.) As the number of commodities in the model increases, the number of dual-ascent iterations that do not improve the dual objective increase rapidly. Therefore, the running time of the dual-ascent algorithm increases as the number of Steiner nodes decreases, and also increases sharply with problem size (since the number of commodities is proportional to the number of required nodes). Consequently, as mentioned in Chapter 3, we believe Implementations 1, 2, and 3, might provide better running times for Stage 1, especially the Implementations 1 and 3, that strictly increase the lower bound in every dual-ascent step.

Why might the performance of the dual-ascent algorithm deteriorates as the fraction of nodes in T_2 increases? On a careful examination of the dual-ascent algorithm, we found

that there are situations when $v_i^k > v_j^k$ and both w_{ij}^k and w_{ji}^k are greater than zero. Notice that if $v_i^k > v_j^k$, we can set $w_{ij}^k = 0$ and $w_{ji}^k = v_i^k - v_j^k$, i.e., reduce the w values, while retaining the dual solution's feasibility. However, as a result, some tight arcs might become loose, and so the auxiliary network might no longer be feasible. This situation presents us with an opportunity to apply the dual-ascent algorithm again and increase the lower bound. Arguing identically as we did for Wong's algorithm (see Section 3.4.2), we can show that at the conclusion of Stage 1, $w_{ij}^k = \max(0, v_j^k - v_i^k)$ for all commodities. Therefore, the violations to $w_{ij}^k = \max(0, v_j^k - v_i^k)$ occur in Stage 2 (and thus for commodities with destinations in T_2). In some recent research [Rag94], we have made nontrivial modifications to the dual-ascent algorithm to ensure that $w_{ij}^k = \max(0, v_j^k - v_i^k)$ for all commodities $k \in K$ and arcs $(i, j) \in A$. We believe that, when implemented, the computational performance of this version of the dual-ascent algorithm will improve upon our results, especially for problems with a large fraction of nodes in T_2.

6.7 Concluding Remarks

In this chapter, using a directed flow model that is equivalent to the improved models presented in Chapter 5, we developed a new dual-ascent algorithm for the network design problem with low connectivity constraints. This algorithm generalizes Wong's dual-ascent algorithm for the Steiner branching problem.

Our algorithm is based upon several concepts from both the optimization and computer science communities. In Chapter 5 we showed how to improve formulations for the unitary network design problems (which includes the NDLC problem). Much of this work was ground deeply in ideas from the mathematical programming literature. In this chapter our emphasis was on developing an efficient dual-ascent algorithm. To that end, we drew upon several ideas and concepts from the graph theory and computer science literature. In particular, we introduced a graphical structure called twinless strongly connected components, and used them to identify ascent directions for the dual to the LP relaxation of the directed flow model.

Our computational results are quite encouraging. We have quickly solved NDLC problems that are an order of magnitude larger than anyone else has solved in the literature. For problems with a small fraction of requirement 2 nodes (those in T_2), our code performs relatively well, with an average error gap of about 6 percent when the fraction of required nodes in T_2 is less than 20 percent. As the fraction of nodes in T_2 increases to about 60 percent, the average gaps between the upper and lower bounds generated by the dual-ascent algorithm increase to about 11 percent. However, as compared to the optimal LP solution, the heuristic solution is typically within 3-4 percent of optimality, indicating

that this heuristic procedure is reasonably good. Further developments to this code, as suggested in the previous section, should improve its performance (both its average error gaps and its running time).

Chapter 7

A Strong Formulation for the Network Design Problem with Connectivity Requirements

In the course of this thesis we have described several models for the network design problem with connectivity requirements (NDC) and its special cases. Our starting point was the well-known cutset and undirected flow formulations (models (1.1) and (1.2)). By using a directing procedure, for the unitary NDC problem, we improved upon these formulations. This development raises the following question. Can we develop better models for nonunitary NDC problems by directing them?

In this chapter we first study the Steiner forest problem, a special case of the NDC problem. Recall that in this problem we are given a graph $G = (N, E)$ and node sets $T_1, T_2, \ldots, T_P$ with $T_i \cap T_j = \phi$ for all node set pairs $(i \neq j)$. We wish to design a graph at minimum cost that connects the nodes in each node set (possibly by including multiple node sets in any connected component of the graph). There are several reasons for studying the Steiner forest problem. First, it is an important network design problem with applications, for example, in VLSI design (see [LMSL92]). Perhaps more importantly, strong models for the Steiner forest problem lead to significantly improved models for the nonunitary NDC problem. The main contribution of this chapter is a strong formulation for the Steiner forest problem, and its generalization as a strong directed model for the NDC problem.

To our knowledge, the model we present in this chapter is the first directed model for the Steiner forest problem in the literature. For the Steiner forest problem, we are aware of only the well-known cutset formulation and the undirected flow model presented in Chapter 1. For the NDC problem with general edge-connectivity requirements, Balakrishnan, Magnanti and Mirchandani [BMM94a] described an enhanced cutset formulation that is stronger than

the cutset model (for the Steiner forest problem, their formulation is equivalent to the cutset formulation). We will briefly compare the enhanced cutset model with the directed model for the NDC problem that we present in this chapter.

The strong formulation for the Steiner forest problem also leads to a stronger formulation for a more general multi-level network design problem. In the two-level network design problem, we are given a network with two facility types available for each edge—primary and secondary—and a partition of the nodes into two types—primary nodes and secondary nodes. In this scenario, primary edges are more expensive than secondary edges. We wish to design a minimum cost connected network that connects the primary nodes to each other by primary edges. A more general version of this problem splits the primary nodes into primary sets $T_1, \ldots, T_P$ with $T_i \cap T_j = \phi$ for all primary node set pairs. We wish to design a minimum cost connected network that connects the nodes in each primary set by primary edges. We can interpret this problem as a Steiner forest problem overlayed on a tree (the primary edges in the optimal solution define a forest, and the edges in the optimal solution, both primary and secondary, define a tree). Developing a stronger model for the Steiner forest problem yields better models for this two-level network design problem (and for the multi-level network design problem).

This chapter is organized as follows. We first show how to direct the Steiner forest problem. Next, we describe three equivalent flow-based formulations for the Steiner forest problem. We then describe our main result: how to direct the NDC problem with edge-connectivity requirements. Finally, we summarize future research directions.

7.1 Directing the Steiner Forest Problem

For convenience, we once again describe the cutset and undirected flow formulations for the Steiner forest problem.

Cutset formulation for the Steiner forest problem

$$\text{Minimize} \quad \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{7.1a}$$

$$\text{subject to:} \quad \sum_{\{i,j\} \in \delta(S)} x_{ij} \geq 1 \quad \begin{array}{l} \text{for all } S,\ \phi \subset S \subset N, \\ S \cap T_i \neq \phi \text{ and } S \backslash T_i \neq \phi \text{ for some i,} \end{array} \tag{7.1b}$$

$$x_{ij} \geq 0 \quad \text{and integer.} \tag{7.1c}$$

In the following undirected flow formulation for the Steiner forest problem, we select a root node for each node set and send one unit of flow from the root node of each node set to every node in that node set.

Undirected flow formulation for Steiner forest problem:

$$\text{Minimize} \quad \sum_{\{i,j\}\in E} c_{ij}x_{ij} \tag{7.2a}$$

$$\text{subject to:} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k = \begin{cases} -1 & \text{if } i = O(k); \\ 1 & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } i \in N \text{ and } k \in K \tag{7.2b}$$

$$\left.\begin{matrix} f_{ij}^k \\ f_{ji}^k \end{matrix}\right\} \leq x_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{7.2c}$$

$$f_{ij}^k, f_{ji}^k \geq 0 \qquad \text{for all } \{i,j\} \in E \text{ and } k \in K \tag{7.2d}$$

$$x_{ij} \geq 0 \qquad \text{and integer; for all } \{i,j\} \in E. \tag{7.2e}$$

If we assume each $c_{ij} \geq 0$, these formulations always have an optimal solution that is a Steiner forest, and so each component of the forest is a tree. Nodes belonging to any node set T_i, for any i, lie in the same component. As an example, Figure 7-1 shows the optimal solution to a Steiner forest problem with five components. One component contains the two node sets T_1 and T_4. All the other components do not contain nodes from other node sets.

How might we direct the Steiner forest problem? Since each component in the optimal solution is a tree, we could arbitrarily choose a node in each component and direct each tree away from it. Unfortunately, before we solve the problem, although we know that nodes in each node set will lie in the same component, we do not know the number of components in the optimal solution and the node sets they contain. The problem is to determine, *a priori*, the root node for each component. For this reason, directing the Steiner forest problem raises difficulties not encountered in directing the Steiner tree problem and unitary NDC problems.

In directing the Steiner forest, for each set T_i, we choose an arbitrary root node $r_i \in T_i$. We then direct each component (tree) away from the lowest indexed root node that it contains. In the example shown in Figure 7-2(a), one component contains two node sets T_1 and T_4. Since T_1 is the lowest indexed node set in this component, we have directed the component away from the root node r_1 of node set T_1. All the other components contain nodes from only one node set T_i, for $i = 1, 2, 3, 4, 5$, and we direct each of them away from the root node r_i of node set T_i. Figure 7-2(b) shows the forest after we have applied the "directing procedure."

For notation, if $j \in T_i$, we let $\rho(j) = r_i$ denote the root node of the node set that node j belongs to. We refer to r_i as node j's root node. We also define $T = T_1 \cup T_2 \cup \ldots T_P$, and let R be the set of all root nodes, that is, $R = \{r_1, r_2, \ldots, r_P\}$.

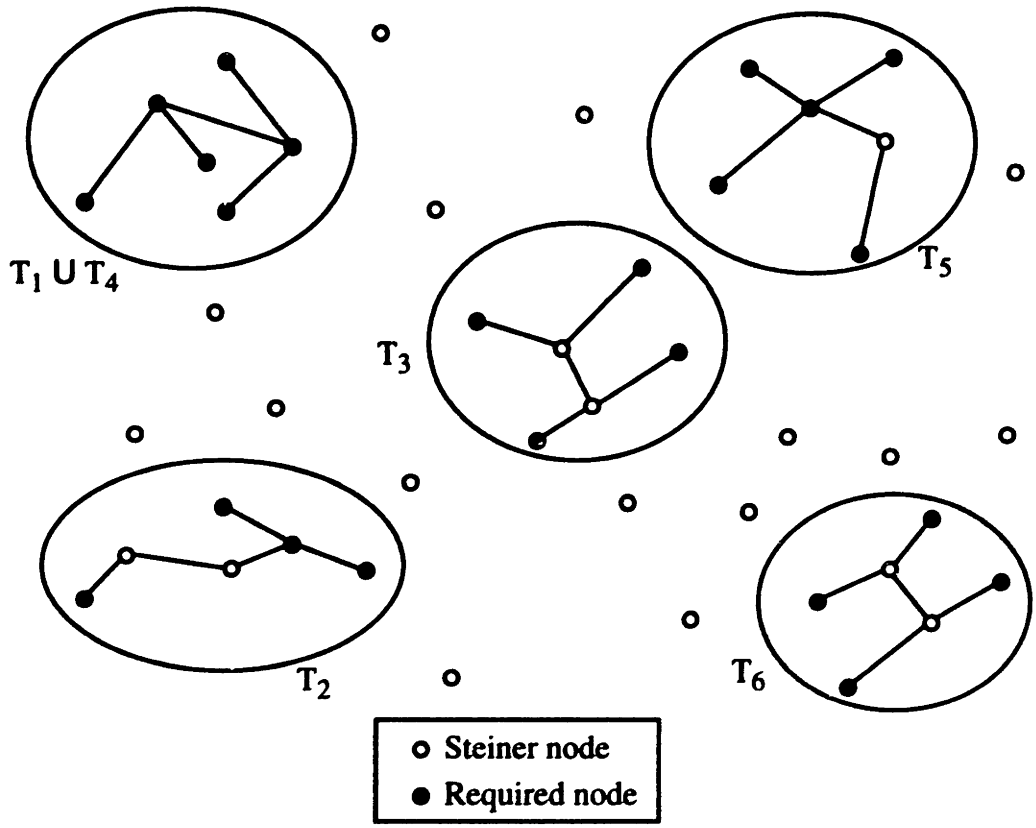


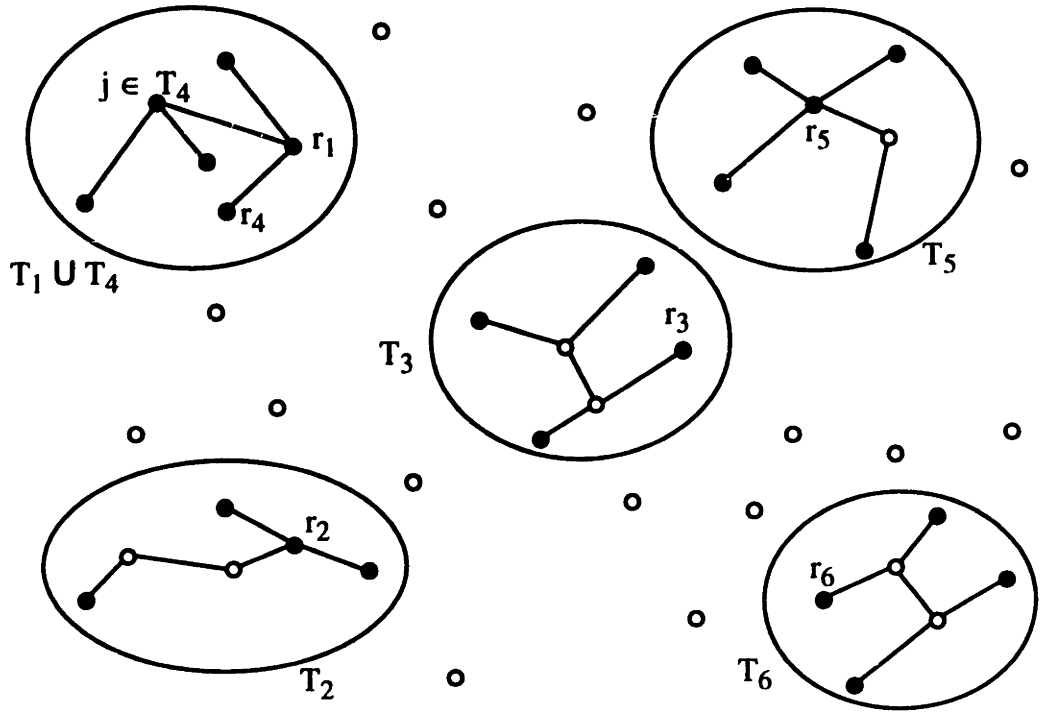
Figure 7-1: An undirected forest.

7.2 Formulations for the Steiner Forest Problem

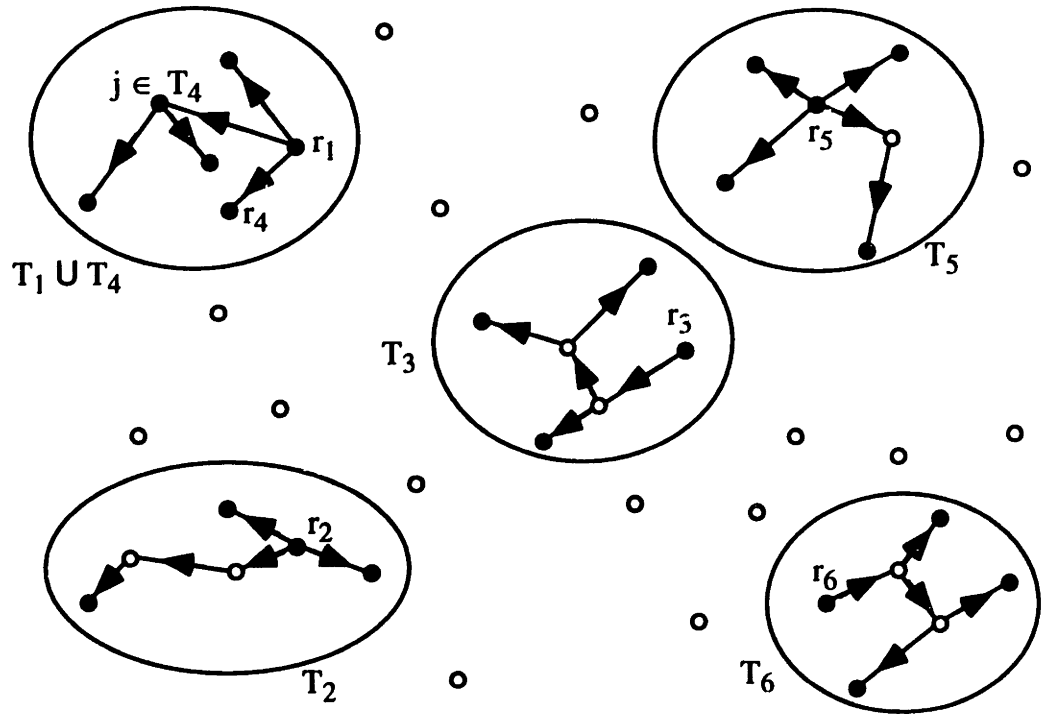
We model the Steiner forest problem using commodity flows. Observe that the network we obtain after directing the Steiner forest contains a directed path from the lowest indexed root node in a component to all other nodes in that component. Therefore, we can send a unit of flow from the root node of each directed component to every node in that component.

For each node $j \in T_i$, with $j \neq r_i$, and for each $p \leq i$, we define a commodity with origin node r_p and destination node j, and for each root node r_i and for each $p < i$, we define a commodity with origin node r_p and destination node r_i. In the optimal solution, it is possible to send a unit of flow from the lowest indexed root node of a component to each required node in that component. Let $CO(q)$ denote the set of all commodities that have node q as their origin, and $CD(q)$ denote the set of all nodes that have node q as their destination and let K denote the set of all commodities. We also define $\mathcal{H} = \{S : S \subset K, \text{and} |S \cap CO(r_j)| = 1 \text{for all } j = 1, \ldots, P\}$. That is, each member of $\mathcal{H}$ is a set of commodities with exactly one commodity having each root node as its origin.

We now describe three different flow based formulations for the Steiner forest problem. The first formulation is on an undirected graph and the latter two are directed formulations.



(a) Undirected forest.



(b) Direct each forest away from the lowest indexed root node that it contains.

Figure 7-2: Example of the directing procedure.

In the first formulation, we let x_{ij} be 1 if the network design contains edge $\{i,j\}$ and be 0 otherwise. The improved undirected flow formulation for the Steiner forest problem has the following form.

Improved undirected flow formulation for Steiner forest problem:

$$\text{Minimize} \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{7.3a}$$

$$\text{subject to} \quad \sum_{j\in N} f^k_{ji} - \sum_{l\in N} f^k_{il} \begin{cases} \geq -1 & \text{if } i = O(k); \\ \leq 1 & \text{if } i = D(k); \\ = 0 & \text{otherwise;} \end{cases} \begin{matrix} \forall i \in N; \\ \text{and} \\ k \in K; \end{matrix} \tag{7.3b}$$

$$\begin{matrix} \sum_{k\in CO(i)} \sum_{j\in N} f^k_{jD(k)} \\ - \sum_{k\in CO(i)} \sum_{l\in N} f^k_{D(k)l} \end{matrix} = 1 \qquad \text{for all } i \in T\backslash R \tag{7.3c}$$

$$\begin{matrix} \sum_{j\in N} f^k_{jD(k)} \\ - \sum_{l\in N} f^k_{D(k)l} \end{matrix} \leq \begin{matrix} \sum_{j\in N} f^{k^*}_{jD(k^*)} \\ - \sum_{l\in N} f^{k^*}_{D(k^*)l} \end{matrix} \qquad \begin{matrix} \forall\, j \in T\backslash R,\ \forall k \in CD(j), \\ \text{s.t. } O(k) = O(k^*), \text{ and} \\ D(k^*) = \rho(j); \end{matrix} \tag{7.3d}$$

$$\sum_{k\in H} f^k_{ij} + \sum_{k\in \bar{H}} f^k_{ji} \leq x_{ij} \qquad \begin{matrix} \text{for all } \{i,j\} \in E, \text{ and} \\ \text{all } H, \bar{H} \text{ pairs in } \mathcal{H} \end{matrix} \tag{7.3e}$$

$$\sum_{i\in N} \sum_{k\in H} f^k_{ij} \leq 1 \qquad \begin{matrix} \text{for all } j \in N, \text{ and} \\ \text{all } H \text{ in } \mathcal{H} \end{matrix} \tag{7.3f}$$

$$\left.\begin{matrix} f^k_{ij} \\ f^k_{ji} \end{matrix}\right\} \geq 0 \qquad \begin{matrix} \text{for all } \{i,j\} \in E, \text{ and} \\ k \in K \end{matrix} \tag{7.3g}$$

$$x_{ij} \in \{0,1\} \qquad \text{for all } \{i,j\} \in E. \tag{7.3h}$$

Constraints (7.3b) and (7.3c) ensure that each node in $T\backslash R$ obtains a unit of flow from either its root node, or the root node of a lower indexed node set. Constraint (7.3d) ensures that if node $i \in T_j$, $i \neq r_j$, is supplied by a commodity k whose origin is not the root node of set T_j, then its root node also is supplied from the origin of commodity k (i.e., its root node belongs to the same component that it belongs to). Constraint (7.3e) follows from the property that in an optimal solution flow travels in only one direction across an edge, and all the flow across an edge originates from the same source (the root node of the component the edge belongs to). Constraint (7.3f) follows from the fact that flow into any node in a component originates from a single node (the root node of that component).

We can simplify some notation in equations 7.3c and 7.3d, if we set $f^k_{D(k)l} = 0$ for all commodities $k \in K$, and all nodes $l \in N$. If commodity k flows out of node $D(k)$, constraints (7.3b) and (7.3c) imply that it flows around a cycle and returns to $D(k)$. By decreasing the flow on this cycle, we can obtain a feasible solution with $f^k_{D(k)l} = 0$. We will make this change to equation (7.3c) and (7.3d) in the next two formulations.

To formulate the problem on a digraph, $D = (N, A)$, we replace each undirected edge $\{i, j\}$ by two directed arcs (i, j) and (j, i), each with the same cost as the undirected edge $\{i, j\}$. In the directed flow formulation, let y_{ij} be 1 if arc (i, j) is in the design, and 0 otherwise. The directed formulation is

Directed flow formulation for Steiner forest problem:

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij} y_{ij} \tag{7.4a}$$

$$\text{subject to} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k \begin{cases} \geq -1 & \text{if } i = O(k); \\ \leq 1 & \text{if } i = D(k); \\ = 0 & \text{otherwise;} \end{cases} \begin{matrix} \forall i \in N; \\ \text{and} \\ k \in K; \end{matrix} \tag{7.4b}$$

$$\sum_{k\in CO(i)} \sum_{j\in N} f_{jD(k)}^k = 1 \qquad \text{for all } i \in T\backslash R \tag{7.4c}$$

$$\sum_{j\in N} f_{jD(k)}^k \leq \sum_{j\in N} f_{jD(k^*)}^{k^*} \qquad \begin{matrix} \forall\, j \in T\backslash R, \forall k \in CD(j), \\ \text{s.t. } O(k) = O(k^*), \text{ and} \\ D(k^*) = \rho(j); \end{matrix} \tag{7.4d}$$

$$\sum_{k\in H} f_{ij}^k \leq y_{ij} \qquad \begin{matrix} \text{for all } (i,j) \in A, \text{ and} \\ H \in \mathcal{H} \end{matrix} \tag{7.4e}$$

$$y_{ij} + y_{ji} \leq 1 \qquad \text{for all } \{i, j\} \in E \tag{7.4f}$$

$$\sum_{i\in N} \sum_{k\in H} f_{ij}^k \leq 1 \qquad \begin{matrix} \text{for all } j \in N, \text{ and} \\ \text{all } H \text{ in } \mathcal{H} \end{matrix} \tag{7.4g}$$

$$f_{ij}^k \geq 0 \qquad \begin{matrix} \text{for all } (i,j) \in A, \text{ and} \\ k \in K \end{matrix} \tag{7.4h}$$

$$f_{D(k)l}^k = 0 \qquad \begin{matrix} \text{for all } l \in N, \text{ and} \\ k \in K \end{matrix} \tag{7.4i}$$

$$y_{ij} \in \{0, 1\} \qquad \text{for all } (i, j) \in A. \tag{7.4j}$$

In this formulation, constraints (7.4b), (7.4c), (7.4d), and (7.4g) are the same as those in Formulation (7.3). Constraint (7.4f) is valid because the optimal solution to the Steiner forest problem contains at most one of the two arcs (i, j) and (j, i). Constraint (7.4e) is valid because all the flow on an arc originates from the same root node.

Yet another way to formulate the Steiner forest problem is to use a layered network. In this approach, we create P copies, $D^1, \ldots, D^P$ of the digraph D. That is, we replace each arc (i, j) by P arcs $(i, j)^1, \ldots, (i, j)^P$, each with cost $c_{ij}^p = c_{ij}$ for $p = 1 \ldots, P$. If the root node of a component is node r_p, then the component contains only arcs from the digraph D^p. For example, in Figure 7-2(b) the component that contains node sets T_1 and T_4 is rooted at node r_1. As a result, all arcs in this component belong to the digraph D^1.

Let y_{ij}^p be a binary variable representing whether arc $(i,j)^p$ is in the network design. We formulate the problem as follows.

Multilevel directed flow formulation for Steiner forest problem:

$$\text{Minimize} \quad \sum_{(i,j)\in A}\sum_{p=1}^{P} c_{ij}y_{ij}^p \tag{7.5a}$$

$$\text{subject to} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k \begin{cases} \geq -1 & \text{if } i = O(k); \\ \leq 1 & \text{if } i = D(k); \\ = 0 & \text{otherwise;} \end{cases} \begin{matrix} \forall i \in N; \\ \text{and} \\ k \in K; \end{matrix} \tag{7.5b}$$

$$\sum_{k\in C(i)}\sum_{j\in N} f_{jD(k)}^k = 1 \qquad \text{for all } i \in T\backslash R \tag{7.5c}$$

$$\sum_{j\in N} f_{jD(k)}^k \leq \sum_{j\in N} f_{jD(k^*)}^{k^*} \qquad \begin{matrix} \forall\, j \in T\backslash R, \forall k \in CD(j), \\ \text{s.t. } O(k) = O(i^*), \text{ and} \\ D(k^*) = \rho(j); \end{matrix} \tag{7.5d}$$

$$f_{ij}^k \leq y_{ij}^p \qquad \begin{matrix} \text{for all } (i,j) \in A, k \in K; \\ \text{such that } O(k) \in r_p, \end{matrix} \tag{7.5e}$$

$$\sum_{p=1}^{P}(y_{ij}^p + y_{ji}^p) \leq 1 \qquad \text{for all } \{i,j\} \in E \tag{7.5f}$$

$$\sum_{i\in N}\sum_{k\in H} f_{ij}^k \leq 1 \qquad \begin{matrix} \text{for all } j \in N, \text{ and} \\ \text{all } H \text{ in } \mathcal{H} \end{matrix} \tag{7.5g}$$

$$f_{ij}^k \geq 0 \qquad \begin{matrix} \text{for all } (i,j) \in A, \text{ and} \\ k \in K; \end{matrix} \tag{7.5h}$$

$$f_{D(k)l}^k = 0 \qquad \begin{matrix} \text{for all } l \in N, \text{ and} \\ k \in K \end{matrix} \tag{7.5i}$$

$$y_{ij}^p \in \{0,1\} \qquad \begin{matrix} \text{for all } (i,j) \in A, \text{ and} \\ p = 1, \ldots, P. \end{matrix} \tag{7.5j}$$

In this formulation, constraints (7.5b), (7.5c), (7.5d), and (7.5g) are the same as those in Formulations (7.3) and (7.4). Constraint (7.5f) is valid because the optimal solution contains at most one arc from $(i,j)^1, \ldots, (i,j)^P, (j,i)^1, \ldots, (j,i)^P$. Constraint (7.5e) is valid because any flow sent on arc $(i,j)^p$ must originate at node r_p (arc $(i,j)^p$ belongs to a component only if the component's root node is r_p).

The improved undirected flow formulation, the directed flow formulation, and the multilevel directed formulation are closely related. We show that the LP relaxations of all three formulations are equivalent.

Theorem 7.2.1 *If the flow costs are zero, the LP relaxations of the improved undirected flow formulation, the directed flow formulation, and the multilevel directed formulation are equivalent.*

Proof: In all three formulations, the flow variables are identical. In transforming the solution of the LP relaxation of one formulation to a solution to the LP relaxation of another, we will make no changes in the flow variables. As a result, to prove that the formulations are equivalent we need to show how to transform the edge or arc variables from one formulation to another so that after the transformation, the solution satisfies the constraints that contain edge or arc variables.

1. To transform a solution to the LP relaxation of (7.5) to a solution to the LP relaxation of (7.4) with no change in cost, we let $y_{ij} = \sum_{p=1}^{P} y_{ij}^p$. Then replacing $\sum_{p=1}^{P} y_{ij}^p$ by y_{ij} in constraint (7.5f) yields constraint (7.4f). Observing that any combination of flows with different sources originate from different root nodes, we obtain the constraint

$$\begin{aligned} \sum_{k \in H} f_{ij}^k &\leq \sum_{\{p.r_p = O(k); k \in H.\}} y_{ij}^p \qquad \text{where } H \in \mathcal{H}. \\ &\leq \sum_{p=1}^{P} y_{ij}^p \\ &= y_{ij}. \end{aligned}$$

2. To transform a solution to the LP relaxation of (7.4) to a solution to the LP relaxation of (7.3) with no change in cost, we let $x_{ij} = y_{ij} + y_{ji}$. Then replacing $y_{ij} + y_{ji}$ by x_{ij} in constraint (7.4f), we obtain $x_{ij} \leq 1$. By adding constraint (7.4e) for arcs (i,j) and (j,i) and replacing $y_{ij} + y_{ji}$ by x_{ij}, we obtain constraint (7.3e).

3. To transform a solution to the LP relaxation of (7.3) to a solution to the LP relaxation of (7.5) with no change in cost, we set $y_{ij}^p = \max_{\{k:O(k)=r_p\}} f_{ij}^k$ and $y_{ji}^p = \max_{\{k:O(k)=r_p\}} f_{ji}^k$, for all $\{i,j\} \in E$ and $p = 2, \ldots, P$. Also, set $y_{ij}^1 = \max_{\{k:O(k)=r_1\}} f_{ij}^1$ and $y_{ji}^1 = x_{ij} - y_{ij}^1 - \sum_{p=2}^{P}(y_{ij}^p + y_{ji}^p)$. Constraint (7.3e) is $\sum_{k \in H} f_{ij}^k + \sum_{k \in \bar{H}} f_{ji}^k \leq x_{ij}$ for all $H, \bar{H}$ pairs in $\mathcal{H}$. Therefore,

$$\sum_{p=1}^{P} \max_{\{k:O(k)=r_p\}} f_{ij}^k + \sum_{p=1}^{P} \max_{\{k:O(k)=r_p\}} f_{ji}^k \leq x_{ij}$$

Substitute (i) y_{ij}^p in place of $\max_{\{k:O(k)=r_p\}} f_{ij}^k$ for $p = 1, \ldots, P$, (ii) y_{ji}^p in place of $\max_{\{k:O(k)=r_p\}} f_{ji}^k$ for $p = 2, \ldots, P$, and (iii) $\sum_{p=1}^{P}(y_{ij}^p + y_{ji}^p)$ in place of x_{ij}, in the previous equation. We find that $\max_{\{k:O(k)=r_1\}} f_{ji}^1 \leq y_{ji}^1$. Therefore, the resulting solution satisfies constraint (7.5e) ($f_{ij}^k \leq y_{ij}^p$ if $O(k) = r_p$). Noting that x_{ij} is less than or equal to 1 shows that the solution satisfies constraint (7.5f) ($\sum_{p=1}^{P} y_{ij}^p + y_{ji}^p \leq 1$).

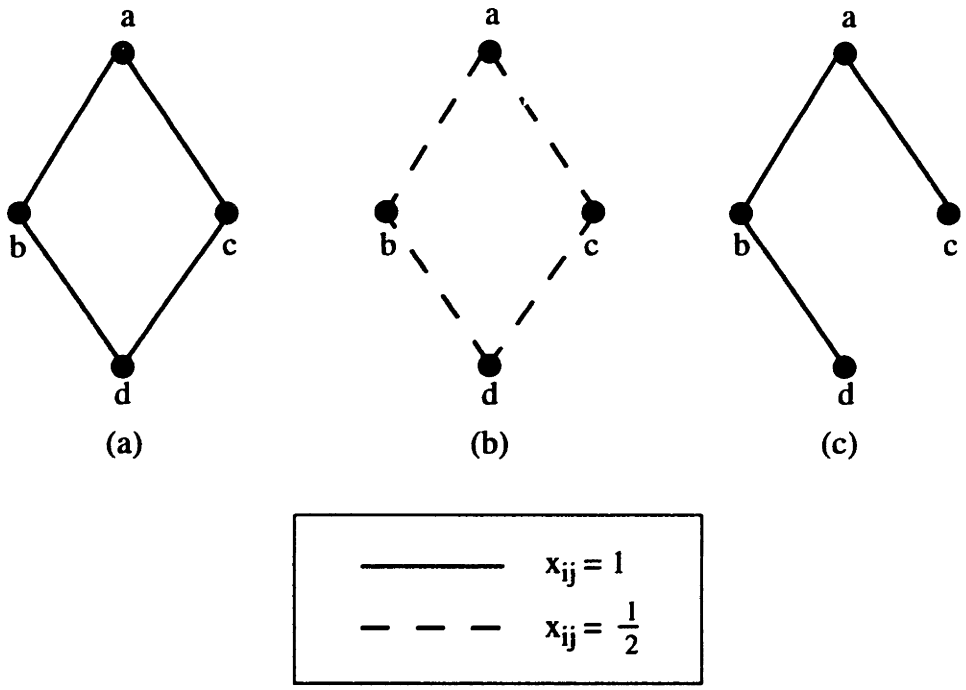


Figure 7-3: (a) Graph with unit edge costs and $T_1 = \{a, d\}$, and $T_2 = \{b, c\}$. (b) Optimal solution to LP relaxation of cutset formulation. (c) Optimal solution to LP relaxation of improved undirected flow formulation.

■

Figure 7-3 shows that the the improved undirected flow formulation is stronger than the cutset formulation (or undirected flow formulation, since they are equivalent) for the Steiner forest problem. In this example $T_1 = \{a, d\}$, $T_2 = \{b, c\}$, and each edge has a cost of 1 unit. The optimal solution to the cutset formulation sets $x_{ab} = x_{bd} = x_{dc} = x_{ca} = 0.5$ with a cost of 2 units. In the improved undirected flow formulation, we select node a as the root of node set T_1, and select node b as the root of node set T_2. This formulation contains four commodities. Commodities 1, 2, and 3 have origin node a and destinations nodes b, c and d. Commodity 4 has origin node b and destination node c. The optimal solution to the LP relaxation of the improved undirected flow formulation sets $x_{ab} = x_{bd} = x_{ad} = 1$, $f^2_{ac} = f^1_{ab} = f^3_{ab} = f^3_{bd} = 1$ (notice that all the commodities originate at the node a, the lowest indexed root node of the optimal solution), and has a cost of 3 units.

Researchers have previously shown that the LP relaxations of the improved undirected flow formulation and the directed flow formulation (the multi-level directed flow formulation is identical to the directed flow formulation for the Steiner tree problem) provide integer solutions (i.e., the design variables are integer) for the Steiner tree problem when the underlying graph is series-parallel (see [PLG85]). Is a similar result true for the Steiner forest

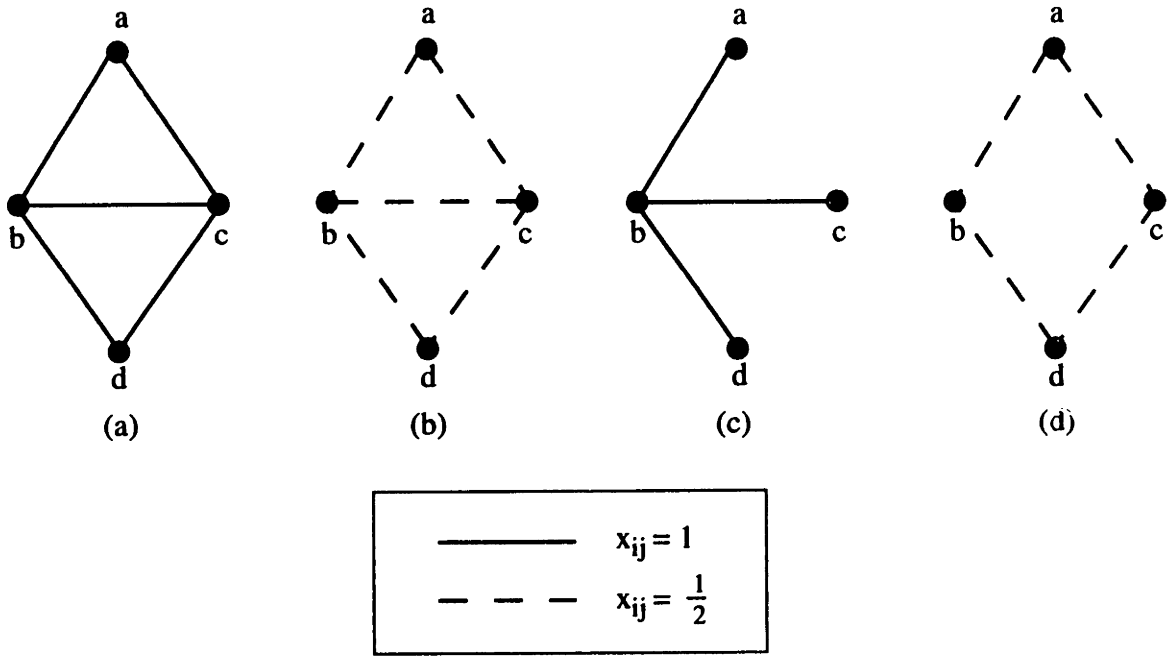


Figure 7-4: (a) Graph with unit edge costs and $T_1 = \{a, d\}$, and $T_2 = \{b, c\}$. (b) Optimal solution to LP relaxation of improved undirected flow formulation. (c) Optimal integer solution. (d) Optimal solution to LP relaxation of cutset formulation.

problem? The example in Figure 7-3 suggests that this might be the case, since the underlying graph in that example is series-parallel. However, Figure 7-4 provides a example showing that this property is not true. This example is very similar to the example in Figure 7-3. The graph, a series-parallel graph, contains an additional edge between node b and node c. In this example $T_1 = \{a, d\}$, $T_2 = \{b, c\}$ and each edge has a cost of 1 unit. In the improved undirected flow formulation, we select the root nodes and commodities as in the previous example. The optimal solution to the LP relaxation of the improved undirected flow formulation is $x_{ab} = x_{bd} = x_{dc} = x_{ca} = x_{bc} = 0.5$, $f^3_{ab} = f^3_{bd} = f^3_{ac} = f^3_{cd} = f^1_{ab} = f^2_{ac} = f^4_{bc} = 0.5$, with a cost of 2.5. The optimal integer solution is $x_{ab} = x_{bd} = x_{bc} = 1$ with a cost of 3 units. The optimal solution to the LP relaxation of the cutset formulation (or undirected flow formulation) is the same as in the previous example and has a cost of 2 units. Thus the result is not true for Steiner forest problems.

Before we conclude this section, we note that we modeled the directing procedure using commodity flows. Unlike the Steiner tree problem and the unitary NDC problem, there does not seem to be any obvious way to formulate a directed cut model. This shows the flexibility and power of flow models for modeling network design problems with connectivity constraints.

7.3 Directing the NDC Problem

In Section 5.1.2 we showed how to direct unitary NDC problems. In this section we show how to generalize the directing procedure we have just presented for the Steiner forest problem to obtain a directed model for all NDC problems. As a result, we obtain a stronger formulation for the NDC model with edge-connectivity requirements.

To sketch the basic idea underlying the directing procedure, we first consider NDC problems where the connectivity requirements r_{st} are even or one. Any solution to the NDC problem consists of one or more connected components. By following the procedure described in Section 5.1.1 for each connected component of the integer solution to the NDC problem, we can direct each component. However, like the Steiner forest problem, because the problem is nonunitary, we do not know a priori the number of connected components in the optimal solution and the required nodes[1] they contain. (We do know that if we delete the edges $\{s,t\}$ with $r_{st} = 0$ from the requirement spanning tree, then the nodes that belong to the same tree (component) of the requirement forest will be in the same component of the solution to the NDC problem.) By combining the directing procedure for the unitary NDC problem and the directing procedure for the Steiner forest problem, we obtain a directed model for the NDC problem.

The following commodity selection procedure outlines the essential idea of the directing procedure. In this discussion, we no longer restrict the connectivity requirements to be even or one. We first use the directing procedure described in Chapter 5 to direct the problem for commodities with $r_{st} \geq 2$. We then apply the Steiner forest problem's directing procedure and direct the bridge edges in each component of the optimal integer solution to the NDC problem.

Commodity selection procedure for Formulation (7.6)

1. Find the requirement spanning tree.
2. Delete all edges with $r_{st} = 0$ from the requirement spanning tree.
3. For each edge $\{s,t\}$ of the requirement spanning tree with $r_{st} \geq 2$, create two commodities: one with origin node s and destination node t, and the other with origin node t and destination node s; each of these commodities has a flow requirement of $r_{st}/2$. Let L denote this set of commodities.
4. Contract each edge $\{s,t\}$ with $r_{st} \geq 2$ in the requirement spanning tree, creating a forest F in which $r_{ij} = 1$ for all edges $\{i,j\}$. Identify the connected components

[1] We say a node is required if it has a connectivity requirement. Recall a node i has a connectivity requirement if $r_{ij} \geq 1$ for some node j.

$T_1, T_2, \ldots, T_P$ of this forest. Denote any node in F created by contraction by any of the nodes it contains in the original requirement spanning tree (e.g., if contracting nodes s and t creates a node in F, then we denote the contracted node by s). Select a contracted node in each set T_i as the root node r_i of the node set T_i. (If node set T_i does not contain a contracted node, then arbitrarily select any one of the nodes as the root node.) Now create commodities as described for the Steiner forest problem with node sets $T_1, T_2, \ldots, T_P$, and root nodes $r_1, r_2, \ldots r_P$. Let K denote this set of commodities.

Using this set of commodities we obtain the following improved undirected flow formulation.

Improved undirected flow formulation for NDC problem:

$$\text{Minimize} \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{7.6a}$$

$$\text{subject to} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k \begin{cases} \geq -1 & \text{if } i = O(k); \\ \leq 1 & \text{if } i = D(k); \\ = 0 & \text{otherwise;} \end{cases} \Bigg\} \begin{array}{l} \forall i \in N; \\ \text{and} \\ k \in K; \end{array} \tag{7.6b}$$

$$\sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k = \begin{cases} -q_k & \text{if } i = O(k); \\ q_k & \text{if } i = D(k); \\ 0 & \text{otherwise;} \end{cases} \Bigg\} \begin{array}{l} \forall i \in N; \\ \text{and} \\ k \in L; \end{array} \tag{7.6c}$$

$$\sum_{k\in CO(i)} \sum_{j\in N} f_{jD(k)}^k = 1 \qquad \text{for all } i \in T\backslash R \tag{7.6d}$$

$$\sum_{j\in N} f_{jD(k)}^k \leq \sum_{j\in N} f_{jD(k^*)}^{k^*} \qquad \begin{array}{l} \forall\, j \in T\backslash R, \forall k \in CD(j), \\ \text{s.t. } O(k) = O(k^*), \text{ and} \\ D(k^*) = \rho(j); \end{array} \tag{7.6e}$$

$$\sum_{k\in H} f_{ij}^k + \sum_{k\in \bar{H}} f_{ji}^k \leq x_{ij} \qquad \begin{array}{l} \text{for all } \{i,j\} \in E, \text{ and} \\ \text{all } H, \bar{H} \text{ pairs in } \mathcal{H} \end{array} \tag{7.6f}$$

$$\sum_{i\in N} \sum_{k\in H} f_{ij}^k \leq 1 \qquad \begin{array}{l} \text{for all } j \in N, \text{ and} \\ \text{all } H \text{ in } \mathcal{H} \end{array} \tag{7.6g}$$

$$f_{ij}^k + f_{ij}^h \leq x_{ij} \qquad \text{for all } k, h \in K \cup L \tag{7.6h}$$

$$f_{D(k)l}^k = 0 \qquad \text{for all } l \in N \text{ and } k \in K \tag{7.6i}$$

$$x_{ij} \leq 1 \qquad \text{for all } \{i,j\} \in E \tag{7.6j}$$

$$\left.\begin{matrix} f_{ij}^k \\ f_{ji}^k \end{matrix}\right\} \geq 0 \qquad \begin{array}{l} \text{for all } \{i,j\} \in E, \text{ and} \\ k \in K \cup L \end{array} \tag{7.6k}$$

$$x_{ij} \geq 0 \qquad \text{and integer, for all } \{i,j\} \in E. \tag{7.6l}$$

In this formulation, $T = T_1 \cup \ldots \cup T_P$, $R = \{r_1, \ldots, r_P\}$, and $\rho(j)$ denotes node j's root node. $CO(k)$, $CD(k)$, and $\mathcal{H}$ are defined as for the Steiner forest problem for the commodities $k \in K$.

If the requirement spanning tree contains no edge $\{s,t\}$, with $r_{st} = 1$, Formulation (7.6) contains no commodities in K, and so the model contains only constraints (7.6c), (7.6h), (7.6j), (7.6k), and (7.6l). In this case, an argument similar to the one we used in Section 5.1.2 shows that this formulation is equivalent to the undirected flow formulation. Consequently, the improved formulation for the NDC problem is stronger than the undirected flow formulation only when the maximum spanning tree of the requirement graph contains some edge $\{s,t\}$ with $r_{st} = 1$.

The enhanced cutset formulation (5.6) of Balakrishnan, Magnanti, and Mirchandani [BMM94a] also applies to the nonunitary NDC problem. For the Steiner forest problem, the enhanced cutset model is equivalent to the cutset model. Therefore, there are instances for the nonunitary NDC problem when the improved flow formulation is stronger than the enhanced cutset formulation. We conjecture that this is true in general, and so the improved undirected flow formulation (7.6) is stronger than the enhanced cutset formulation (5.6)

7.4 Conclusions

The main contribution of this chapter is the strong formulation for the NDC problem. To develop this formulation, we first showed how to direct the Steiner forest problem. Based upon this directing procedure, we presented three equivalent formulations for the Steiner forest problem. We then generalized one of these formulations to the NDC problem with edge-connectivity requirements. To our knowledge, these formulations are the first directed formulations for the Steiner forest problem and the NDC problem with edge-connectivity requirements.

Although we have not proved this result in this chapter, the max-flow min-cut theorem implies that (i) the choice of root node for each node set, and (ii) the order of node sets does not affect the optimal objective value of the LP relaxation of the improved flow formulation for the Steiner forest problem or the NDC problem.

There are many unanswered questions concerning the new directed model for the NDC problem. How strong is the directed model? Is it significantly stronger than the undirected flow or cutset models? Can we derive new classes of valid inequalities for nonunitary NDC problems by projecting out the flow variables from this model? (Like we did for the unitary NDC problem.) Can we derive an improved dual-ascent procedure (or perhaps a Lagrangian relaxation method) based on this improved formulation? In our future research, we plan to investigate these issues.

Chapter 8

Summary and Final Remarks

In this thesis we considered the network design problem with connectivity requirements (NDC). This problem has many applications, especially to telecommunication network design, and is of considerable theoretical interest because it generalizes several well-known combinatorial optimization problems.

Our research was motivated by the need for efficient solution procedures for large-scale NDC problems. Consequently, we devoted our attention to understanding the structural properties of solutions to the NDC problem, and exploiting these properties to develop mixed integer programming models and computationally attractive solution procedures. In particular, we designed

1. A dual-ascent algorithm for a general NDC problem. This algorithm optimally solves special cases including the k-edge-disjoint path problem and the k-node-disjoint path problem. The algorithm is to date the only heuristic procedure that applies to the general NDC problem.

2. A dual-ascent algorithm for the network design problem with low connectivity requirements (NDLC). This algorithm generalizes Wong's [Won84] well-known algorithm for the Steiner branching problem.

Since a dual-ascent algorithm attempts to approximately solve the dual of the LP relaxation of a formulation, it is important to use strong formulations of a problem in order to obtain good lower bounds from a dual-ascent procedure. With this objective in mind, we developed strong models for the NDC problem. We started Chapter 3 by reviewing a well-known directing procedure for the minimum spanning tree problem and Steiner tree problem. In Chapter 5 we generalized this directing procedure and developed a directed model for all unitary NDC problems with edge-connectivity requirements. This model generalizes a directed model described by Goemans [Goe90] for the survivable network design

problem (SND) when the connectivity requirements of the nodes are even or one. By projecting out the flow variables from the improved model for the unitary NDC problem, we generalized some known classes of facet-defining inequalities for the Steiner tree problem—partition, odd-hole, and combinatorial design—to the unitary NDC problem. We continued our development of strong models for the NDC in Chapter 7 by showing how to direct the Steiner forest problem. We then described a directed model for the Steiner forest problem, and later generalized this model to all NDC problems with edge-connectivity requirements. The directed model for the Steiner forest problem and the nonunitary NDC problem are new.

Because flow models, although compact, are fairly large, we devised methods to reduce the size of these formulations by minimizing the number of commodities they contain. For this purpose, the main tool we used for edge-connectivity problems is the Gomory-Hu requirement spanning tree. We showed how to model edge-connectivity problems and the node-connectivity version of the SND problem using $\mathcal{O}(|N|)$ commodities. We also outlined a scheme for reducing the number of commodities in the flow model of NDC problems with general node-connectivity requirements.

We established a Lagrangian duality framework to derive dual-ascent algorithms for network design problems. Using this framework, we developed dual-ascent algorithms for the Steiner branching problem, the NDC problem, and the NDLC problem. We introduced the notion of a *minimal ascent set*, and using this concept provided a simple interpretation of Wong's algorithm for the Steiner branching problem (and Edmonds' algorithm for the branching problem). We also used minimal ascent sets, and a generalization of this notion (i.e., minimal twin ascent sets), in designing the dual-ascent algorithm for the NDLC problem. Throughout this research, we devoted considerable attention to devising efficient implementations of these algorithms. For this purpose, several graph search algorithms proved to be particularly useful.

We implemented some of our dual-ascent algorithms, testing them on some important NDC problems—the network design problem with low connectivity requirements, and the Steiner tree problem. These algorithms solve NDLC problems that are an order of magnitude larger than previously solved (problems with upto 300 nodes and 3000 edges). Our results indicate that the dual-ascent algorithms are quick, and generate excellent heuristic solutions (typically within 4 percent of optimality for the NDLC problem, and within 1 percent of optimality for the Steiner tree problem), as well as reasonably good lower bounds. The quality of the lower bound of the dual-ascent algorithm for the NDLC problem deteriorates as the fraction of nodes with a connectivity requirement of 2 increases. We believe that certain modifications to this dual-ascent algorithm, as described in Chapter 6, will improve its performance both in terms of providing a better lower bound and a faster running time.

We view our research as synthesizing concepts from two important areas: mathematical programming and computer science. In developing mixed integer programming models, we relied on concepts that are traditional to the optimization community; while developing dual-ascent algorithms, we used many algorithmic ideas from the computer science literature. The use of both these areas proved crucial to the development of the dual-ascent algorithms. Without a good model, the dual-ascent procedure is bound to generate poor lower bounds. In the dual-ascent algorithms, we usually related some graphical structure (e.g., twinless strongly connected components) to an ascent direction. Without an efficient algorithm for identifying such structures, the running time of a dual-ascent procedure might be excessively large. By combining lessons learned from both fields we developed quick and efficient dual-ascent methods.

The following *recipe* summarizes the main lesson learned about using dual-ascent as an algorithmic approach to solve combinatorial optimization problems.

Recipe for a dual-ascent algorithm

1. Develop a good model for the problem. (For network problems, perhaps by directing the problem.)

2. Study special cases of the problem that are polynomially solvable and devise dual-ascent methods that solve them optimally.

3. Generalize the dual-ascent method to the more general problem.

This is the approach we have taken for developing our algorithms, and one that we have found to be particularly effective. Algorithms designed to optimally solve polynomially solvable cases seem to capture the properties of a "good" solution. Consequently, when applied to the more general $\mathcal{NP}$-hard problem, these algorithms seem to generate reasonably good solutions.

In conclusion, a dual-ascent procedure developed for a strong formulation of a problem, followed by an efficient implementation, can be a very effective solution methodology for (approximately) solving combinatorial optimization problems.

Appendix A

Abbreviations

DFS	Depth first search
DNDC	Directed network design problem with connectivity requirements
IP	Integer program
LP	Linear program
MIP	Mixed integer program
MST	Minimum spanning tree problem
NDC	Network design problem with connectivity requirements
NDLC	Network design problem with low connectivity requirements
RBOC	Regional Bell Operating Companies
SND	Survivable network design problem
TSCC	Twinless strongly connected component
TSP	Traveling salesman problem

Bibliography

[ABCC94] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Solving traveling salesman problems. In *15th International Symposium on Mathematical Programming, Ann Arbor, Michigan*, August 1994.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, 1993.

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for $\mathcal{NP}$-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23:11–24, 1989.

[Bal84] A. Balakrishnan. *Valid inequalities and algorithms for the network design problem with an application to LTL consolidation.* PhD thesis, Massachusetts Institute of Technology, 1984.

[BBM90] D. Bienstock, E. F. Brickell, and C. L. Monma. On the structure of minimum-weight k-connected spanning networks. *SIAM Journal on Discrete Mathematics*, 3(3):320–329, 1990.

[Ben92] J. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal of Computing*, 4:387–411, 1992.

[Ber91] D. P. Bertsekas. *Linear Network Optimization: Algorithms and Codes.* MIT Press, 1991.

[BH93] S. C. Boyd and T. Hao. An integer polytope related to the design of survivable communication networks. *SIAM Journal on Discrete Mathematics*, 6(4):612–630, 1993.

[BLW87] W. Bern, E. L. Lawler, and L. Wong. Linear time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8:216–235, 1987.

[BM76] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* North-Holland, 1976.

[BM93] M. Baïou and A. R. Mahjoub. The 2-edge connected Steiner subgraph polytope of a series-parallel graph. Technical report, Départment d'informatique, Université de Bretagne Occidentale, 6 Avenue Victor Le Gorgeu, B.P. 452, 29275 Brest Cedex, France, 1993.

[BMM94a] A. Balakrishnan, T. L. Magnanti, and P. Mirchandani. Doubling or splitting: Strategies for survivable network design. Technical Report OR 297-94, OR-Center, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1994.

[BMM94b] A. Balakrishnan, T. L. Magnanti, and P. Mirchandani. A dual-based algorithm for multi-level network design. *Management Science*, 40:567–581, 1994.

[BMMN95] M. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser. *Network Models.* Handbooks in Operations Research and Management Science. North-Holland, 1995. To appear.

[BMW89] A. Balakrishnan, T. L. Magnanti, and R. T. Wong. A dual ascent procedure for large-scale uncapacitated network design. *Operations Research*, 37:716–740, 1989.

[BP83] E. Balas and W. R. Pulleyblank. The perfectly matchable subgraph polytope of a bipartite graph. *Networks*, 13:495–516, 1983.

[BR92] P. Berman and V. Ramaiyer. Improved approximation algorithms for the Steiner tree problem. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 325–334, 1992.

[CF70] W. Chou and H. Frank. Survivable communication networks and the terminal capacity matrix. *IEEE Transactions on Circuit Theory*, CT-17:192–197, 1970.

[Cho89] S. Chopra. The dominant of the spanning tree polytope. *Operations Research Letters*, 1989.

[Cho92a] S. Chopra. The equivalent subgraph polytope and directed cut polyhedra on series-parallel graphs. *SIAM Journal on Discrete Mathematics*, 5:475–490, 1992.

[Cho92b] S. Chopra. Polyhedra of the equivalent subgraph problem and some edge connectivity problems. *SIAM Journal on Discrete Mathematics*, 5:321–337, 1992.

[Chr76] N. Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1976.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[CMW89] R. H. Cardwell, C. L. Monma, and T. H. Wu. Computer-aided design procedures for survivable fiber optic networks. *IEEE Journal on Selected Areas in Communications*, 7:1188–1197, 1989.

[CR94] S. Chopra and M. R. Rao. The Steiner tree problem I: Formulations, compositions and extensions of facets. *Mathematical Programming*, 64(2):209–229, 1994.

[CRRW91a] C. R. Coullard, A. Rais, R. L. Rardin, and D. K. Wagner. The 2-connected Steiner subgraph polytope for series-parallel graphs. Technical Report 91-32, Purdue University, West Lafayette, Indiana, 1991.

[CRRW91b] C. R. Coullard, A. Rais, R. L. Rardin, and D. K. Wagner. The dominant of the 2-connected Steiner subgraph polytope for W_4-free graphs. Technical Report 91-34, Purdue University, West Lafayette, Indiana, 1991.

[CS89] G.-R. Cai and Y.-G. Sun. The minimum augmentation of any graph to k-edge connected graph. *Networks*, 19:151–172, 1989.

[CW81] N. Christofides and C. A. Whitlock. Network synthesis with connectivity constraints—a survey. In Brans, editor, *Operational Research '81*, pages 705–723. North-Holland, 1981.

[Dah92] G. Dahl. *Contributions to the design of survivable directed networks.* PhD thesis, University of Oslo, Norway, 1992.

[Dah94] G. Dahl. The design of survivable directed networks. *Telecommunication Systems*, 2:349–377, 1994.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DW71] S. E. Dreyfus and R. A. Wagner. The Steiner tree problem in graphs. *Networks*, 1:195–207, 1971.

[EA78] C. El-Arbi. Une heuristique pour le problème de l'arbre de Steiner. *RAIRO Operations Research*, 12:207–212, 1978.

[Edm67] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, B 71:241–245, 1967.

[Edm70] J. Edmonds. Submodular functions, matroids, and a certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schonheim, editors, *Combinatorial Structures and their Applications*, pages 69–87. Gordon and Breach, 1970.

[Edm73] J. Edmonds. Edge-disjoint branchings. In B. Rustin, editor, *Combinatorial Algorithms*, pages 91–96. Academic Press, 1973.

[Edm79] J. Edmonds. Matroid intersection. *Annals of Discrete Mathematics*, 4:185–204, 1979.

[Erl78] D. Erlenkotter. A dual based procedure for uncapacitated facility location. *Operations Research*, 26:992–1009, 1978.

[ET76] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5:653–665, 1976.

[Eul36] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Comment. Academiae Sci. I. Petropolitanae*, 8:128–140, 1736.

[Eve79] S. Even. *Graph Algorithms.* Computer Science Press, Rockville, MD, 1979.

[FBM89] D. Fernandez-Baca and C. U. Martel. On the efficiency of maximum flow algorithms on networks with small integer capacities. *Algorithmica*, 4:173–189, 1989.

[FC69] H. Frank and W. Chou. Connectivity considerations in the design of survivable networks. Technical report, Executive Office of the President, Office of Emergency Preparedness, Washington, DC, January 1969.

[FC70] H. Frank and W. Chou. Connectivity considerations in the design of survivable networks. *IEEE Transactions on Circuit Theory*, CT-17:486–490, 1970.

[FF56] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[FJ81] G. N. Frederickson and J. JáJá. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10:270–283, 1981.

[FJ82] G. N. Frederickson and J. JáJá. On the relationship between the biconnectivity augmentation and traveling salesman problems. *Theoretical Computer Science*, 19:189–201, 1982.

[Fra92] A. Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM Journal on Discrete Mathematics*, 5(1):25–53, 1992.

[Fra94] A. Frank. Connectivity augmentation problems in network design. In John R. Birge and Katta G. Murty, editors, *Mathematical Programming: State of the Art 1994*. The University of Michigan, 1994.

[FRT89] D. Fussell, V. Ramachandran, and R. Thurimella. Finding triconnected components by local replacements. In *Proceedings of the 16th ICALP*, number 372 in Lecture notes in Computer Science, pages 379–393. Springer-Verlag, 1989.

[FS71] D. R. Fulkerson and L. S. Shapley. Minimal k-arc connected graphs. *Networks*, 1:91–98, 1971.

[FT89] A. Frank and É. Tardos. An application of submodular flows. *Linear Algebra and its Applications*, 114/115:329–348, 1989.

[Gab91] H. N. Gabow. A matroid approach to finding edge-connectivity and packing arborescences. In *Proceedings of the 23rd Annual Symposium on the Theory of Computing*, pages 112–122, 1991.

[GB93] M. X. Goemans and D. J. Bertsimas. Survivable networks, linear programming relaxations and the parsimonious property. *Mathematical Programming*, 60(2):145–166, June 1993.

[GGP+94] M. X. Goemans, A. Goldberg, S. Plotkin, D. B. Shmoys, E. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 1994.

[GGW93] H. N. Gabow, M. X. Goemans, and D. P. Williamson. An efficient approximation algorithm for the survivable network design problem. In *Proceedings of the Third MPS Conference on Integer Programming and Combinatorial Optimization*, pages 57–74, 1993.

[GH61] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *SIAM Journal of Applied Mathematics*, 9:551–570, 1961.

[GM90] M. Grötschel and C. L. Monma. Integer polyhedra arising from certain network design problems with connectivity constraints. *SIAM Journal on Discrete Mathematics*, 3:502–523, 1990.

[GM93] M. X. Goemans and Y.-S. Myung. A catalog of Steiner tree formulations. *Networks*, 23(1):19–28, January 1993.

[GMS92a] M. Grötschel, C. L. Monma, and M. Stoer. Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research*, 40:309–330, 1992.

[GMS92b] M. Grötschel, C. L. Monma, and M. Stoer. Facets for polyhedra arising in the design of communication networks with low-connectivity requirements. *SIAM Journal on Optimization*, 2, 1992.

[GMS92c] M. Grötschel, C. L. Monma, and M. Stoer. Polyhedral and computational investigations for designing communication with high survivability requirements. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Heilbronner Str. 10, Berlin, Germany, 1992.

[GMS95] M. Grötschel, C. L. Monma, and M. Stoer. Design of survivable networks. In M. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Models*, Handbooks in Operations Research and Management Science. North-Holland, 1995. To appear.

[Goe90] M. X. Goemans. *Analysis of linear programming relaxations for a class of connectivity problems.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, 1990. Also Working paper OR 233-90, Operations Research Center, M.I.T.

[Goe93] M. X. Goemans. Worst-case comparison of valid inequalities for the TSP. Technical report, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, 1993. To appear in Mathematical Programming 1995.

[Goe94a] M. X. Goemans. Private Communication, 1994.

[Goe94b] M. X. Goemans. The Steiner tree polytope and related polyhedra. *Mathematical Programming*, 63(2):157–182, January 1994.

[Gom63] R. E. Gomory. An algorithm for integer solutions to linear programs. In R. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, 1963.

[GP85] M. Grötschel and M. Padberg. Polyhedral theory. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys, editors, *The traveling salesman problem: a guided tour of combinatorial optimization.* Wiley, 1985.

[GT91] M. X. Goemans and K. T. Talluri. 2-change for k-connected networks. *Operations Research Letters*, 1991.

[GW92] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 307–316, 1992. To appear in *SIAM Journal of Computing*, 1995.

[Har62] F. Harary. The maximum connectivity of a graph. *Proceedings of the National Academy of Science U.S.A.*, 48:1142–1146, 1962.

[HK70] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[Hof60] A. J. Hoffman. Some recent applications of the theory of linear inequalities to extremal combinatorial analysis. In R. Bellman and M. Hall, editors, *Combinatorial Analysis*, pages 113–128. American Mathematical Society, Providence, RI, 1960.

[HP85] K. Hoffman and M. Padberg. LP-based combinatorial problem solving. *Annals of Operations Research*, 4:145–194, 1985.

[HRW92] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*, volume 53 of *Annals of Discrete Mathematics*. North-Holland, 1992.

[Joh90] D. S. Johnson. Local search and the traveling salesman problem. In G. Goos and J. Hartmanis, editors, *Automata, Languages and Programming*, number 443 in Lecture notes in Computer Science, pages 446–461. Springer-Verlag, 1990.

[Jor94] T. Jordán. On the optimal vertex-connectivity augmentation. To appear in *Journal of Combinatorial Theory B*, 1994.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KM89] C.-W. Ko and C. L. Monma. Heuristic methods for designing highly survivable communication networks. Technical report, Bellcore, Morristown, New Jersey, 1989.

[KMB81] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[Kru56] J. B. Kruksal. On the shortest spanning tree of graph and the traveling salesman problem. *American Mathematical Society*, 7:48–50, 1956.

[KT93] S. Khuller and R. Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14:214–225, 1993.

[KU86] Y. Kajitani and S. Ueno. The minimum augmentation of directed tree to a k-edge-connected directed graph. *Networks*, 16:181–197, 1986.

[KV92] S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 759–770, 1992.

[Law76] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[Lev71] A. Y. Levin. Algorithm for the shortest connection of a group of graph vertices. *Soviet Math Doklady*, 12:1477–1481, 1971.

[LMSL92] C.-L. Li, S. T. McCormick, and D. Simchi-Levi. The point-to-point delivery and connection problems: Complexity and algorithms. *Discrete Applied Mathematics*, 36:267–292, 1992.

[Mar86] R. K. Martin. A sharp polynomial size linear programming formulation of the minimum spanning tree problem. Technical report, University of Chicago, 1986.

[Men27] K. Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10:96–115, 1927.

[Min89] M. Minoux. Network synthesis and optimum network design problems: models, solution methods and applications. *Networks*, 19:313–360, 1989.

[MMP90] C. L. Monma, B. L. Munson, and W. R. Pulleyblank. Minimum-weight two-connected spanning networks. *Mathematical Programming*, 46:153–171, 1990.

[MPL94] F. Margot, A. Prodon, and Th. M. Liebling. Tree polytope on 2-trees. *Mathematical Programming*, 63(2):183–192, 1994.

[MS89] C. L. Monma and D. F. Shallcross. Methods for designing communication networks with certain two-connected survivability constraints. *Operations Research*, 37:531–541, 1989.

[MS91] F. Margot and M. Schaffers. Integrality proofs with a silicon flavor for polytopes on graphs definable by compositions. Technical Report RO911217, Départment de Mathématiques, EPF Lausanne, Lausanne, Switzerland, 1991.

[MS94] F. Margot and M. Schaffers. Integrality proofs by projection for composition of polytopes. Preprint, July 1994.

[MW84] T. L. Magnanti and R. T. Wong. Network design and transportation planning: models and algorithms. *Transportation Science*, 18:1–55, 1984.

[MW95] T. L. Magnanti and L. A. Wolsey. Optimal trees. In M. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Models*, Handbooks in Operations Research and Management Science. North-Holland, 1995. To appear.

[NW60] C. St. J. A. Nash-Williams. On orientations, connectivity, and odd vertex pairings in finite graphs. *Canadian Journal of Mathematics*, 12:555–567, 1960.

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Ple81] J. Plesnik. A bound for the Steiner tree problem in graphs. *Math. Slovaca*, 31:151–163, 1981.

[PLG85] A. Prodon, Th. M. Liebling, and H. Gröflin. Steiner's problem on two-trees. Technical Report RO850315, Départment de Mathématiques, EPF Lausanne, Lausanne, Switzerland, 1985.

[Pri57] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[Rag92] S. Raghavan. Linear time algorithms for connectivity problems on series-parallel graphs. Technical Report 45313-920904-01TM, AT&T Bell Laboratories, Holmdel, New Jersey, 1992.

[Rag94] S. Raghavan. Unpublished research notes, 1994.

[RG77] A. Rosenthal and A. Goldner. Smallest augmentations to biconnect a graph. *SIAM Journal on Computing*, 6:55–66, 1977.

[Rob39] H. E. Robbins. A theorem on graphs with an application to a problem of traffic control. *American Mathematical Monthly*, 46:281–283, 1939.

[RW93] R. Ravi and D. P. Williamson. An approximation algorithm for the minimum cost k-vertex-connected subgraph problem. Unpublished manuscript, 1993.

[Sas87] T. Sastry. Some bounds on the gaps between the dual-ascent, linear programming and the integer programming solutions to the Steiner tree problem. Unplublished manuscript, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, 1987.

[SC92] S. Sridhar and R. Chandrasekaran. Integer solution to synthesis of communication networks. *Mathematics of Operations Research*, 17(3):581–585, 1992.

[SP93] Standard and Poor's Industry Surveys : Telecommunications, April 1 1993.

[SP94] Standard and Poor's Industry Surveys : Telecommunications, June 2 1994.

[Sto92] M. Stoer. *Design of Survivable Networks.* Number 1531 in Lecture Notes in Mathematics. Springer-Verlag, 1992.

[Suu74] J. W. Suurbaale. Disjoint paths in a network. *Networks*, 4:125–145, 1974.

[SWK69] K. Steiglitz, P. Weiner, and D. J. Kleitman. The design of minimum cost survivable networks. *IEEE Transactions on Circuit Theory*, CT-16:455–460, 1969.

[TEM93a] Assignment: Network survivability—report on the collective efforts of the network reliability council. Supplement to TE&M, 1993.

[TEM93b] Telephone Engineering and Management, August 1993.

[TEM94] Telephone Engineering and Management, January 1994.

[TM80] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematica Japonica*, 24:573–577, 1980.

[TTW93] S. Taoka, D. Takafuji, and T. Watanabe. Simplicity preserving augmentation of the edge connectivity of a graph. Technical Report COMP93-73, Department of Circuits and Systems, Faculty of Engineering, Hiroshima University, Hiroshima, Japan, 1993.

[WC83] J. A. Wald and C. J. Colbourn. Steiner trees, partial 2-trees and minimum IFI networks. *Networks*, 13:159–167, 1983.

[WGMV93] D. P. Williamson, M. X. Goemans, M. Mihail, and V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. In

Proceedings of the 25th Annual ACM Symposium on Foundations of Computer Science, pages 338–343, 1993. To appear in *Combinatorica*.

[Whi32] H. Whitney. Non-separable and planar graphs. *Transactions of the American Mathematical Society*, 34:339–362, 1932.

[Wil93] D. P. Williamson. *On the design of approximation algorithms for a class of graph problems.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, 1993.

[Win85] P. Winter. Generalized Steiner problem in Halin networks. In *12th International Symposium on Mathematical Programming, Cambridge, Massachusetts*, 1985.

[Win87] P. Winter. Generalized Steiner problems in series-parallel networks. *Journal of Algorithms*, 7:549–566, 1987.

[WN87] T. Watanabe and A. Nakamura. Edge-connectivity augmentation problems. *Computer System Science*, 35:96–144, 1987.

[WN88] T. Watanabe and A. Nakamura. 3-connectivity augmentation problems. In *IEEE International Symposium on Circuits and Systems*, pages 1847–1850, 1988.

[Won80] R. T. Wong. Integer programming formulations of the traveling salesman problem. In *Proceedings of the IEEE International Conference on Circuits and Computers*, pages 149–152, 1980.

[Won84] R. T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984.

[Yan88] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. In *Proceedings of the 20th Annual Symposium on the Theory of Computing*, pages 223–228, 1988.

[Zel93] A. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9:463–470, 1993.