A Dual-Ascent Procedure for Large-Scale Uncapacitated Network Design

by

A. Balakrishnan, T.L. Magnanti, and
R.T. Wong

OR 161-87 May 1987

A DUAL-ASCENT PROCEDURE FOR LARGE-SCALE UNCAPACITATED NETWORK DESIGN*

A. Balakrishnan
Krannert Graduate School of Management
Purdue University

T. L. Magnanti
Sloan School of Management
Massachusetts Institute of Technology

R. T. Wong[†]
Krannert Graduate School of Management
Purdue University

May 1987

*Supported by Grant #ECS-8316224 from the Systems Theory and Operations Research program of the National Science Foundation.

[†] Supported in part by ONR contract #N000 14-86-0689 from the Office of Naval Research.

Abstract

The fixed-charge network design problem arises in a variety of problem contexts including transportation, communication, and production scheduling. We develop a family of dual ascent algorithms for this problem. This approach generalizes known ascent procedures for solving shortest path, plant location, Steiner network and directed spanning tree problems. Our computational results for several classes of test problems with up to 500 integer and 1.98 million continuous variables and constraints shows that the dual ascent procedure and an associated drop-add heuristic generates solutions that, in almost all cases, are guaranteed to be within 1 to 3 percent of optimality. Moreover, the procedure requires no more than 150 seconds on an IBM 3083 computer. The test problems correspond to dense and sparse networks, including some models arising in freight transport.

Introduction

The fixed charge network design model, which is a fundamental discrete choice design model, is useful for a variety of applications in transportation, distribution, communication, and several other problem settings that make basic cost tradeoffs between operating costs and fixed costs for providing network facilities (see Magnanti and Wong (1984)). The fixed charge design model is deceptively simple. The basic ingredients are a set of nodes and a set of uncapacitated arcs, and, between selected pairs of nodes, a required flow that must be routed over the network. Each arc has two types of cost: a per unit flow cost and a fixed charge for using the arc. The problem is to select a subset of arcs that minimizes the sum of the flow (or routing) costs and fixed charge costs.

Magnanti and Wong (1984) have shown that the basic fixed charge design model is quite flexible and contains a number of well-known network optimization problems as special cases including the shortest path, minimum spanning tree, uncapacitated plant location, Steiner network, and traveling salesman problems.

Since many of these special cases, e.g., the uncapacitated plant location problem, are known to be difficult to solve (in the parlance of computational complexity theory, they are NP-hard), so is the general fixed charge design model. The closely related budget network design problem, that removes the fixed charge terms from the objective function, but instead limits the sum of fixed charges incurred through a budget constraint, is also NP-hard (Johnson, Lenstra, and Rinnooy Kan (1978)). In fact, Wong (1980) has shown that even finding an approximate budget design solution is an NP-hard problem.

In addition to these theoretical arguments, substantial empirical evidence also confirms the difficulty of solving network design problems. A number of researchers including Hoang (1973), Boyce, Farhi and Weischedel (1973), Dionne and Florian (1979), Boffey and Hinxman (1979), Gallo (1981) and Los and Lardinois (1980) have studied branch and bound algorithms for either the fixed charge or budget design problems. Although these branch and bound algorithms can successfully solve problems with a small number of arcs (up to 40-50), their computation time grows very quickly in the problem size.

For larger scale problems, several approximate solution procedures are available. Billheimer and Gray (1973) and Los and Lardinois utilize add-drop type heuristics for the fixed charge design problem; Dionne and Florian, and Boffey and Hinxman use similar techniques for the budget design problem. Wong (1985) describes a special add heuristic for budget design problems on the Euclidean plane. He shows that asymptotically it is very likely that the cost of the solution generated by this heuristic will, under certain conditions, be very close to the cost of the optimal solution.

Some recent research has extended the range of applicability of optimization-based approaches. Magnanti, Mireault and Wong (1986) have solved network design problems with an accelerated version of Benders' decomposition combined with a preprocessing routine that eliminates unnecessary variables. The preprocessing routine utilizes a dual ascent procedure to be described in this paper. This implementation of Benders' method has been able to solve to optimality undirected network problems with up to 30 nodes and 90 candidate arcs. Lamar, Sheffi and Powell (1984) have proposed an iterative linearization scheme embedded in a branch and bound routine. The algorithm has been successfully applied to a special type of fixed charge directed network design problem that arises in freight routing operations. The test problems include networks with up to 46 nodes and 510 fixed-charge arcs.

In this paper we present a new dual-based approach for solving large-scale fixed charge network design problems. We begin in Section 1 by modeling the fixed charge design problem as an integer program and discussing various formulations, including a new one for the undirected network case. Section 2 describes dual ascent procedures for computing approximate solutions to the dual of the linear programming relaxation of the design problem formulation. The dual solution provides a lower bound for the optimal solution value and this solution, together with the complementary slackness conditions, permits us to generate a feasible network design solution. By using a drop-add heuristic, we further improve the upper bound generated by the feasible design solution. The dual lower bound, when used in conjunction with the upper bound from the heuristic solution, enables problem reduction by eliminating some arcs from the network. Section 3 presents these and other enhancements.

As part of our discussion in Section 2, we also comment on some general principles for designing dual ascent algorithms. We show how our ascent procedure can be viewed as a generalization of Dijkstra's (1959) shortest path algorithm and of Wong's (1984) dual ascent procedure for the Steiner network problem, which itself generalizes Bilde and Krarup's (1977) and Erlenkotter's (1978) ascent algorithm for uncapacitated plant location, as well as the Chu-Liu (1965) and Edmonds (1967) directed spanning tree algorithm. Thus, just as the network design model formulation generalizes the shortest path, facility location, spanning tree and Steiner network problems, our algorithm generalizes the dual ascent methods proposed for these special cases.

Finally, in Section 4, we present extensive computational results for our dual-based procedure on a wide variety of randomly generated test problems, including sparse and dense networks containing up to 45 nodes and up to 595 candidate arcs. We also describe our experience in applying the method to solve some special design problems studied by Lamar, Sheffi and Powell (1984) in the setting of freight routing systems.

The dual based approach, with but a few exceptions, has been able to generate upper and lower bounds that differ by at most only 1 to 3 percent. The maximum CPU time required for the entire dual-based procedure was 150 seconds on an IBM 3083 (model BX) computer. The dual ascent component of this procedure is quite efficient; for larger problems, the add-drop heuristic consumes a major portion of the required CPU time.

In the past, dual ascent procedures have been successfully used to solve several special cases of the network design problem including the uncapacitated plant location and Steiner tree problems. Our results, which show that the dual ascent approach can be successfully applied to a variety of large-scale network design models, confirm the power of the dual ascent solution strategy and demonstrate the effectiveness of the approach for a broader range of network optimization problems.

1. Network Design Problem Formulations

This section introduces and discusses formulations for the general fixed charge network design problem. We focus on the undirected network case and give some alternative formulations including a new formulation when the flow costs are the same for all commodities. Sections 2 and 3 introduce algorithms that utilize these network design formulations. All our subsequent discussions apply with minor modifications to the directed network design problem as well.

1.1 General Network Design Model

The basic ingredients of the model are a set N of nodes and a set A of uncapacitated and undirected arcs that are available for designing a network. The selection of those arcs to be included in the network depends upon tradeoffs between fixed design costs and variable operating costs. Selecting more arcs offers the potential for reducing the operating (or routing) costs at the expense of higher fixed costs. On the other hand, with fewer arcs in the design, the fixed costs are lower but the routing costs increase.

The model permits multiple commodities which might represent distinct physical goods, or the same physical good but with different points of origin and destination. (Some authors, e.g., Rardin and Choe (1979), Rardin (1982), and Lamar, Sheffi and Powell (1986), permit commodities to have multiple origins and destinations. Our model assumes, however, that each commodity has a single origin *or* single destination, which we model as a set of commodities, each with a single source *and* single destination.) We let K denote the set of commodities and for each $k \in K$, assume (by scaling, if necessary) that one unit of flow of commodity k must be shipped from its point of origin, denoted O(k), to its point of destination, denoted D(k).

The model contains two types of variables, one modeling discrete design choices and the other modeling continuous flow decisions. Let y_{ij} be a binary variable that indicates whether ($y_{ij} = 1$) or not ($y_{ij} = 0$) arc {i,j} is chosen as part of the network's design. Let x_{ij}^{k} denote the flow of commodity k on the directed arc (i,j). Note that (i,j) and (j,i) denote directed arcs with opposite orientations corresponding to the undirected arc $\{i,j\}$. Even though arcs in the model are undirected, we refer to the directed arcs (i,j) and (j,i) because the flows are directed. Then, if $y = (y_{ij})$ and $x = (x_{ij}^k)$ are vectors of design and flow variables, the model becomes

$[P_1]$

$$\text{minimize} \quad \sum_{k \varepsilon K} \sum_{\{i,j\} \varepsilon A} (c_{ij}^k x_{ij}^k + c_{ji}^k x_{ji}^k) + \sum_{\{i,j\} \varepsilon A} F_{ij} y_{ij} \tag{1.1}$$

subject to

$$\sum_{j \varepsilon N} x_{ji}^k - \sum_{l \varepsilon N} x_{il}^k = \begin{cases} -1 & \text{if } i = O(k) \\ 1 & \text{if } i = D(k) \\ 0 & \text{otherwise} \end{cases} \qquad \begin{array}{l} \text{all } i \ \varepsilon\ N, \text{ and} \\ \text{all } k \ \varepsilon\ K, \end{array} \tag{1.2}$$

$$x_{ij}^k \leq y_{ij} \qquad \text{all } \{i,j\} \ \varepsilon\ A, \text{ and} \tag{1.3a}$$

$$x_{ji}^k \leq y_{ij} \qquad \text{all } k \ \varepsilon\ K, \tag{1.3b}$$

$$\begin{array}{rcl} x_{ij}^k, x_{ji}^k & \geq & 0 \\ y_{ij} & = & 0 \text{ or } 1 \end{array} \qquad \begin{array}{l} \text{all } \{i,j\} \ \varepsilon\ A, \text{ and} \\ \text{all } k \ \varepsilon\ K. \end{array} \tag{1.4}$$

In this formulation each arc $\{i,j\}$ has a nonnegative fixed design cost F_{ij} and c_{ij}^k is the nonnegative per unit cost for routing commodity k on the directed arc (i,j). In general, c_{ij}^k need not equal c_{ji}^k. Constraints (1.2) imposed upon each commodity k are the usual network flow conservation equations. The "forcing" constraints (1.3a) and (1.3b) state that if $y_{ij} = 0$, i.e., if arc $\{i,j\}$ is not included in the design, then the flow of every commodity k on this arc must be zero in both directions, and if arc $\{i,j\}$ is included in the design, i.e., if $y_{ij} = 1$, the arc flow is unlimited (since the flow of any commodity k on arc (i,j) or (j,i) is at most 1 anyway).

For *directed* network design problems, A is the set of available directed arcs and F_{ij} represents the fixed charge of the directed arc (i,j). The formulation uses design variables y_{ij} for all (i,j) ε A and flow variables x^k_{ij} for all (i,j) ε A and k ε K. Inequalities (1.3a) defined for all (i,j) ε A and k ε K constitute the forcing constraints for this case. The flow conservation equations (1.2) remain unchanged, and the summations in the objective function are now taken over (i,j) rather than {i,j} (and the terms $c^k_{ji}x^k_{ji}$ are removed).

The network design problem can also be formulated in a number of different ways. For example, aggregating the forcing constraints (1.3) yields the equivalent set of constraints

$$\sum_k x^k_{ij} + x^k_{ji} \le 2|K|y_{ij} \qquad \text{for all } \{i,j\} \ \varepsilon\ A. \qquad (1.5)$$

Both versions of these constraints force each x^k_{ij} and x^k_{ji} for k ε K to be zero if $y_{ij} = 0$, and become redundant if $y_{ij} = 1$. This alternate formulation is much more compact, however. For one of the problems considered in our computational tests, with 1980 commodities and 500 arcs, the disaggregate formulation contains nearly 2 million forcing constraints (1.3), while the aggregate version contains only 500 of these constraints (1.5).

In addition, it is possible to aggregate commodities by origin (or destination). In this aggregate formulation, x^k_{ij}, which now denotes the total flow on arc (i,j) that originates at node k, corresponds to the aggregation (sum) over all destination nodes of the commodities with origin k. The forcing constraint becomes

$$x^k_{ij} \le |N|y_{ij} \quad \text{and} \quad x^k_{ji} \le |N|y_{ij},$$

and the flow balance constraints (1.2) are altered accordingly (that is, the flow balance constraints for aggregate commodity k impose a requirement of 1 unit for each demand node D(*l*) in the original formulation whose origin O(*l*) is node k). For the problem mentioned in the last paragraph, this aggregate formulation contains 45 rather than 1980 commodities and 45,000 rather than nearly 2 million forcing constraints. (Note that it is also possible, as we do in our modeling approach, to reverse this procedure and disaggregate any commodity naturally formulated with a single origin and multiple destinations as a set of commodities, one for each destination.)

Although the disaggregate formulation contains a considerably larger number of constraints than either of these two aggregate formulations, it is preferred computationally (if it can be solved efficiently). The linear programming relaxation of the disaggregate model is much more tightly constrained than the linear programming version of the aggregate models. Therefore, the disaggregate formulation more closely approximates the integer program and provides a sharper lower bound on the value of the integer programming formulation. Various authors have noted the important algorithmic advantages of using tight linear programming relaxations. For example, see Cornuejols, Fisher and Nemhauser (1977), Davis and Ray (1969), Beale and Tomlin (1972), Geoffrion and Graves (1974), Mairs et al. (1978), Magnanti and Wong (1981), Rardin and Choe, and Williams (1974).

As a final observation concerning the problem formulation (1.1) - (1.4), note that (since the problem is uncapacitated) for a given choice of binary design variables y_{ij}, the problem decomposes into a shortest path problem (defined on the network specified by arcs with $y_{ij} = 1$) for each commodity. Therefore, the problem always has an optimal solution in which all the x variables are integer. Our subsequent analysis and algorithmic development makes heavy use of this problem feature.

1.2 An Improved Formulation for Fixed Charge Network Design

Under certain conditions, the linear programming relaxation of the disaggregate formulation (1.1) - (1.4) can be further tightened with another version of the forcing constraints (1.3). We focus upon undirected network design problems whose flow cost does not vary by commodity, i.e., $c^k_{ij} = c_{ij}$ for all $k \in K$, and $c_{ij} + c_{ji} \geq 0$ for all $\{i,j\} \in A$. Many network design models satisfy this rather mild assumption.

Consider an arc $\{i,j\} \in A$ and two commodities k and h with origins O(k) and O(h) and a common destination D(k) = D(h), which, for convenience, we assume is node 1. Suppose we have an optimal solution (x,y) to the integer program P_1 that routes each commodity k on a shortest path (see our final comment in Section 1.1), and $x_{23}^k = 1$. Then it is always possible to set $x_{32}^h = 0$ and maintain optimality. To see this result, let d_{ij} denote the minimum routing cost from node i to node j as defined on the network specified by the optimal choice of the y_{ij} variables. Since $x_{23}^k = 1$ in the optimal solution and the flow between every origin-destination pair is carried on shortest paths $d_{21} = c_{23} + d_{31}$. If $x_{32}^h = 1$, then the total routing cost for commodity h will be

$$\begin{aligned} d_{O(h),3} + c_{32} + d_{21} &= d_{O(h),3} + c_{32} + c_{23} + d_{31} \\ &\geq d_{O(h),3} + d_{31} \quad (\text{since } c_{32} + c_{23} \geq 0). \end{aligned}$$

Therefore, we can set $x_{32}^h = 0$ without loss of optimality. Similarly, we can reverse the roles of x_{23}^k and x_{32}^h and, therefore, the constraint

$$x_{23}^k + x_{32}^h \leq 1$$

will not affect the value of the optimal solution of the design model. In general, this inequality becomes

$$x_{ij}^k + x_{ji}^h \leq y_{ij} \qquad \text{for all } \{i,j\}\varepsilon A \text{ if } D(k)=D(h). \qquad (1.6)$$

The same type of argument applies when commodities have a common origin O(k) = O(h); so we also have

$$x_{ij}^k + x_{ji}^h \leq y_{ij} \qquad \text{for all } \{i,j\}\varepsilon A \text{ if } O(k)=O(h). \qquad (1.7)$$

Notice that when h = k, (1.6) and (1.7) reduce to

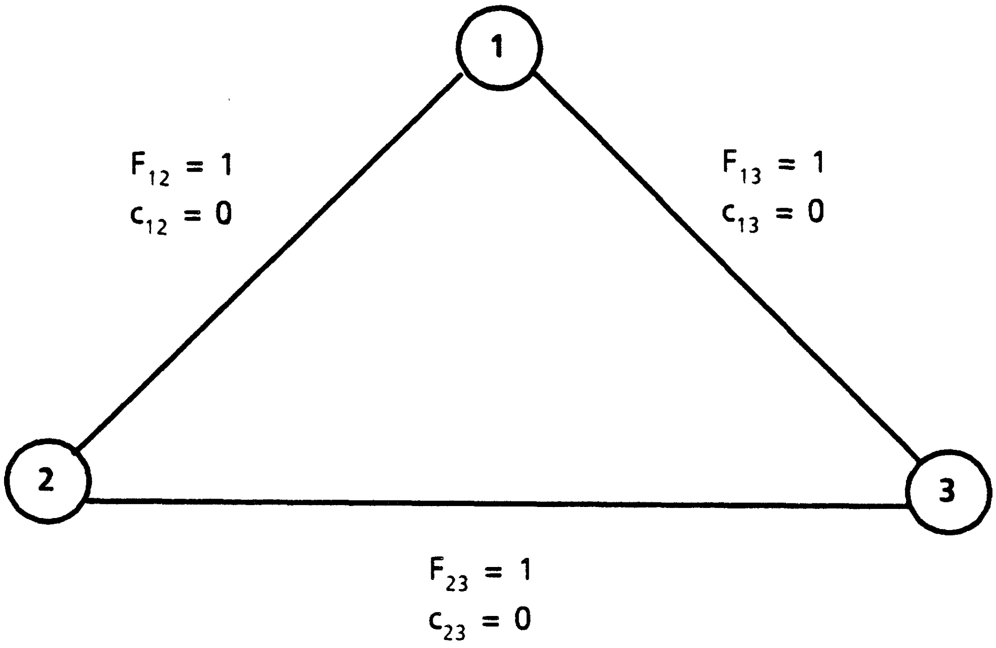
$$x_{ij}^k + x_{ji}^k \leq y_{ij} \qquad \text{for all } \{i,j\}\varepsilon A,\ k\varepsilon K.$$

Since the flow variables are nonnegative, these inequalities are at least as tight as (1.3). So if we substitute (1.6) and (1.7) for (1.3),the linear programming relaxation of the resulting formulation P_2 will be at least as tight as LP_1, the linear programming relaxation of P_1.

LP_2, the linear programming relaxation of P_2, can be strictly tighter than LP_1. Consider, for example, the 3-node network shown in Figure 1.1, with

Figure 1.1

Example for Comparing LP_1 and LP_2



Commodity	Origin	Destination
k	2	1
h	3	1

$F_{12} = F_{13} = F_{23} = 1$ and $c_{ij} = 0$ for all $\{i,j\}$. Suppose that the problem contains two commodities k and h with origins $O(k) = 2$, $O(h) = 3$ and destinations $D(k) = D(h) = 1$. For this problem, the optimal solution to LP_1 is

$$y_{12} = y_{13} = y_{23} = x_{21}^{k} = x_{23}^{k} = x_{31}^{k} = x_{21}^{h} = x_{31}^{h} = x_{32}^{h} = 1/2.$$

The total solution cost is 3/2. Notice that this solution violates the constraint $x_{23}^{k} + x_{32}^{h} \leq y_{23}$ from (1.6). An optimal solution to LP_2 is

$$y_{12} = y_{13} = x_{21}^{k} = x_{31}^{h} = 1$$

which has a total solution cost of 2. Hence, LP_2 is strictly tighter than LP_1 for this example.

For clarity, we first discuss solution methods for the original formulation P_1. Section 2 introduces a general dual ascent framework for approximately solving the dual of LP_1, the linear programming relaxation of problem P_1. We present two alternative implementations of the general dual ascent strategy, and demonstrate how these algorithms generalize the ascent methods proposed earlier for some special cases of the network design problem. Section 3 deals with algorithmic modifications to handle the tighter constraints (1.6) and (1.7), and other enhancements.

2. Dual Ascent Algorithms for the NDP

Since the network design problem is NP-hard, we focus on methods for generating good lower bounds and heuristic solutions, rather than solving the problem optimally. The network design problem's special structure makes it a particularly attractive candidate for applying dual ascent, a technique that attempts to generate good lower bounds relatively fast by solving the linear programming dual problem approximately. Besides generating lower bounds, dual ascent solutions can also be used to identify feasible network designs that serve as starting solutions for local improvement heuristics. Further, they can aid in reducing the problem by eliminating some design variables.

Dual ascent has been applied successfully to several network design related models including the uncapacitated facility location problem (Bilde and Krarup, Erlenkotter, Van Roy and Erlenkotter (1982)), the generalized assignment problem (Fisher, Jaikumar and Van Wassenhove (1986)), the Steiner tree problem (Wong (1984)), the set covering problem (Kedia and Fisher (1986)), and the set partitioning problem (Fisher and Kedia (1986)).

In this section, we outline a general dual ascent framework for the network design problem and develop two algorithms that can be interpreted in terms of this framework. The next section describes various enhancements of the second algorithm including dual-based heuristic and problem reduction methods.

Consider the linear programming dual DP_1 of the formulation P_1.

[DP_1]

$$\text{maximize} \quad z_D = \sum_{k \varepsilon K} v^k_{D(k)} \tag{2.1}$$

subject to

$$\left.\begin{array}{l} v^k_j - v^k_i \le c^k_{ij} + w^k_{ij} \\ v^k_i - v^k_j \le c^k_{ji} + w^k_{ji} \end{array}\right. \qquad \begin{array}{c} \text{for all } k \; \varepsilon \; K, \\ \{i,j\} \; \varepsilon \; A, \end{array} \tag{2.2}$$

$$\sum_k w_{ij}^k + \sum_k w_{ji}^k \le F_{ij} \qquad \text{for all } \{i,j\} \in A \tag{2.3}$$

$$w_{ij}^k \ge 0,\ w_{ji}^k \ge 0 \qquad \text{for all } k \in K, \text{ and all } \{i,j\} \in A \tag{2.4}$$

In this formulation, $\{v_i^k\}$ for all $i \in N$ and $k \in K$ is the dual variable corresponding to the flow conservation equation (1.2) for commodity k at node i, while w_{ij}^k and w_{ji}^k for all $k \in K$ and $\{i,j\} \in A$ correspond to the forcing constraints (1.3a) and (1.3b), respectively. For each commodity $k \in K$, one of the flow conservation equations (1.2) is redundant; hence we have arbitrarily set $v_{O(k)}^k$ equal to zero.

2.1 Dual Ascent Framework

The dual ascent strategy that we consider consists of iteratively modifying the w_{ij}^k, w_{ji}^k values and the v_i^k values in order to increase the dual objective function value monotonically. Observe that, for any given vector w = $\{w_{ij}^k, w_{ji}^k\}$ that satisfies constraints (2.3) of DP_1, the 'best' v-values are obtained by solving (2.1) - (2.2). This subproblem decomposes by commodity; the subproblem $[SP_k(w)]$ corresponding to commodity k is

$[SP_k(w)]$

$$\text{maximize} \quad v_{D(k)}^k \tag{2.5}$$

subject to

$$\left.\begin{aligned} v_j^k - v_i^k &\le \hat{c}_{ij}^k \\ v_i^k - v_j^k &\le \hat{c}_{ji}^k \end{aligned}\right\} \quad \text{for all } k \in K, \text{ and all } \{i,j\} \in A \tag{2.6}$$

$$\text{where} \quad \hat{c}_{ij}^k = c_{ij}^k + w_{ij}^k, \text{ and} \qquad \text{for all } \{i,j\} \in A,$$
$$\hat{c}_{ji}^k = c_{ji}^k + w_{ji}^k \qquad \text{and, all } k \in K.$$

Observe that $[SP_k(w)]$ is the dual of a shortest path problem from origin O(k) to destination D(k) using the *modified arc lengths* $\hat{c}_{ij}^k = c_{ij}^k + w_{ij}^k$ and $\hat{c}_{ji}^k =$

$c_{ji}^k + w_{ji}^k$. For a given set of w-values, setting v_i^k for all $i \varepsilon N$ equal to the length of the shortest path from origin O(k) to node i, with $\hat{c}_{ij}^k$ and $\hat{c}_{ji}^k$ as arc lengths, gives one optimal solution of $[SP_k(w)]$. (In the remainder of this section, we implicitly assume that all shortest paths are determined using the modified arc lengths $\hat{c}_{ij}^k$ and $\hat{c}_{ji}^k$.) In particular, the optimal value of the subproblem $[SP_k(w)]$ is $v_{D(k)}^k$, the length of the shortest path from origin O(k) to destination D(k). Therefore, the dual objective function value can be increased by increasing the length of the shortest origin-to-destination (abbreviated as O-D) path for one or more commodities through appropriate increases in w-values (and, hence, in $\hat{c}$-values). These observations suggest the following ascent strategy:

> Iteratively increase one or more w-values (and hence the modified arc lengths $\hat{c}_{ij}^k$ or $\hat{c}_{ji}^k$) so that
>
> (a) constraints (2.3) remain feasible, and
>
> (b) the shortest O-D path length $v_{D(k)}^k$ increases for at least one commodity $k \varepsilon K$ at each stage.

To satisfy condition (a), we consider increasing w_{ij}^k and w_{ji}^k values corresponding only to those arcs {i,j} for which constraint (2.3) has slack. Suppose the w-values at some iteration satisfy constraint (2.3) corresponding to arc $\{i,j\} \varepsilon A$ as a strict inequality, i.e., the fixed charge F_{ij} for arc {i,j} is not yet completely 'used up' by the w_{ij}^k and w_{ji}^k values. Let s_{ij} represent the slack or "unabsorbed fixed charge" in this constraint. The ascent strategy consists of using this slack to increase the w_{ij}^k or w_{ji}^k value for one or more commodities $k \varepsilon K$ so that the shortest path length $v_{D(k)}^k$ (and, hence, the lower bound Z_D) increases. *Essentially, therefore, the dual ascent procedure seeks to selectively allocate the unabsorbed fixed charges s_{ij} in order to increase the length of the shortest O-D path for one or more commodities at each iteration.*

We have not yet specified how to select the arc(s) whose slack must be allocated and how to allocate this slack to the various commodities. Different arc selection and slack allocation schemes give rise to different implementations of the dual ascent method. We next discuss two alternative implementations - one that changes a single w-value at each iteration and another that simultaneously increases w-values corresponding to several arcs. The first method which we call the Path Diversion method, although not likely to be the best implementation, is simple to state and illustrates several features of the dual ascent approach. The enhancements and computational results to be presented in Sections 3 and 4 pertain to the second method, called the Labeling method.

2.2 Path Diversion Method for Dual Ascent

This implementation increases one w-value (i.e., corresponding to one directed arc (i,j) and one commodity k) at each iteration. Initially, all w-values are set to zero and $v^k_{D(k)}$ for all $k \varepsilon K$ is set equal to the length of the shortest path from O(k) to D(k), using c^k_{ij} and c^k_{ji} as arc lengths. At any intermediate iteration, consider arcs {i,j} that currently have positive slacks s_{ij}. To allocate this slack effectively, we must identify a commodity k so that increasing w^k_{ij} or w^k_{ji} will increase the shortest path length $v^k_{D(k)}$. Clearly, if commodity k has a current shortest O-D path that does not include the directed arc (i,j) (or (j,i)), then increasing w^k_{ij} (or w^k_{ji}) will not increase the shortest path length $v^k_{D(k)}$. Hence, we need to consider only those arcs with positive slack that belong to all the current shortest O-D paths for at least one commodity. To identify such arc-commodity combinations, we compute for each directed arc (i,j) and commodity k the shortest O-D path length excluding (directed) arc (i,j). Let l^k_{ij} denote this shortest path length; l^k_{ij} must be greater than or equal to the current shortest path length $v^k_{D(k)}$. If $l^k_{ij} = v^k_{D(k)}$, then arc (i,j) does not belong to one or more current shortest O-D paths for commodity k; therefore, increasing w^k_{ij} does not affect the shortest path length and hence the dual objective function value Z_D. Suppose $l^k_{ij} > v^k_{D(k)}$. Then increasing w^k_{ij} by up to $(l^k_{ij} - v^k_{D(k)})$ causes a corresponding increase in $v^k_{D(k)}$; further increases in w^k_{ij} leave the dual objective function value unchanged. Thus, at each iteration the algorithm

(a) selects a directed arc (i,j) and commodity k satisfying

$$s_{ij} > 0 \quad \text{and} \quad l^k_{ij} > v^k_{D(k)}, \quad \text{and}$$

(b) increases w^k_{ij} and $v^k_{D(k)}$ by $\min \{s_{ij}, [l^k_{ij} - v^k_{D(k)}]\}$.

When several arc-commodity combinations satisfy the conditions of step (a), a variety of selection rules could be used to choose one of the eligible combinations: for instance, a 'greedy' rule would select the arc-commodity combination that gives the maximum dual objective function increase computed in step (b). The algorithm terminates when no arc-commodity combination satisfies the conditions of step (a).

The main disadvantage of this method is its excessive computational requirements to reevaluate l^k_{ij} for several arcs and commodities at each iteration. Further, since it increases just one w-value at a time, the procedure requires a relatively large number of iterations. In contrast, the algorithm we describe next modifies several w-values simultaneously and uses a labeling scheme to efficiently determine the required changes in the w-values and v-values and to update the dual solution at each iteration. Some preliminary computational experiments showed that this method also produces better lower bounds than the Path Diversion method.

2.3 Labeling Method for Dual Ascent

We now consider an alternative implementation of the dual ascent strategy described in Section 2.1. At each iteration, the method simultaneously increases several w-values corresponding to a single commodity. In order to interpret this method in terms of the general dual ascent framework, we first introduce some notation and underlying concepts.

For any commodity $k \in K$, consider a partition $(N_1(k),N_2(k))$ of the node set N so that $O(k) \in N_1(k)$ and $D(k) \in N_2(k)$. We wish to identify directed arcs (i,j) whose w^k_{ij} values must be increased in order to increase the shortest O-D distance $v^k_{D(k)}$. Let A(k) be the set of all directed arcs (i,j) incident from $N_1(k)$ to $N_2(k)$, that is $\{i,j\} \in A$, $i \in N_1(k)$, and $j \in N_2(k)$. We refer to A(k) as the (directed) *cutset* for commodity k induced by the node partition $(N_1(k),N_2(k))$. Clearly, increasing w^k_{ij} for every directed arc (i,j)

ε A(k) increases the shortest path distance from O(k) to all nodes of $N_2(k)$ (including the destination D(k)) and hence increases the dual objective function value Z_D. However, not all these w^k_{ij} values need to be increased in order to raise $v^k_{D(k)}$. In particular, if an arc (i,j) ε A(k) does not belong to <u>any</u> shortest O-D path for commodity k, then the corresponding w^k_{ij} value can be left unchanged. To identify shortest path arcs of A(k), we note that if arc (i,j) belongs to at least one shortest O-D path, then

$$v^k_j - v^k_i = c^k_{ij} + w^k_{ij} \quad (= \hat{c}^k_{ij}) \ . \qquad (2.7)$$

We refer to arcs satisfying equation (2.7) as *tight* arcs and let A'(k) denote the set of tight arcs in the cutset A(k). The previous argument implies that we need to consider increasing w-values only for arcs in A'(k). For each arc (i,j) ε A'(k), the current slack s_{ij} determines the maximum amount by which w^k_{ij} can increase to maintain feasibility in the dual constraint (2.3); let

$$\delta_1 = \min \{s_{ij} : (i,j) \ \varepsilon \ A'(k)\}.$$

Also, as we increase the w-values for the arcs in A'(k), the shortest path distance, i.e., the v-value, to each node in $N_2(k)$ increases. Consequently, one or more arcs in the set A"(k) = A(k)\A'(k) may become tight; let

$$\delta_2 = \min \{(c^k_{ij} + w^k_{ij}) - (v^k_j - v^k_i) : (i,j) \ \varepsilon \ A''(k)\}.$$

Then, increasing w^k_{ij} by $\delta = \min \{\delta_1, \delta_2\}$ (and correspondingly decreasing the slack s_{ij} by δ) for all arcs (i,j) ε A'(k) increases all shortest path lengths v^k_l to nodes l of $N_2(k)$ by δ, and improves the lower bound. We refer to this updating procedure as the *simultaneous w-increasing* step. Observe that, when $\delta = \delta_1$, the slack s_{ij} for one of the tight arcs reduces to zero; on the other hand, when $\delta = \delta_2$, an arc of A"(k) becomes tight.

Our ascent algorithm mechanizes this strategy of increasing, for each commodity, multiple w-values corresponding to arcs of a suitably chosen cutset. The procedure initializes all w-values to zero and sets v^k_i equal to the shortest path length from O(k) to node i (using the variable costs c^k_{ij} and c^k_{ji} as arc lengths), for all i ε N and k ε K. Also, initially, $N_2(k)$ = {D(k)} and $N_1(k)$ = N \ {D(k)} for all k ε K. At each iteration, the algorithm sequentially considers the commodities k for which $O(k) \notin N_2(k)$. For every such commodity, the implementation performs the simultaneous w-increasing step once. If, as a result, the slack s_{ij} for some tight arc (i,j) becomes zero, then node i is transferred from $N_1(k)$ to $N_2(k)$. We refer to this augmentation of the set $N_2(k)$ as *labeling* and to nodes of $N_2(k)$ as *labeled* nodes. (We show later, in Section 2.5, that this dual ascent labeling step is closely related to, and, in fact, generalizes, the labeling operation in Dijkstra's shortest path algorithm.) The algorithm stops when the origin O(k) is labeled for all commodities $k \varepsilon K$. This procedure can be described formally as follows:

Labeling Method

Step 0: Initialization

Set $w^k_{ij} \leftarrow 0$ and $w^k_{ji} \leftarrow 0$	for all $\{i,j\} \varepsilon A$ and $k \varepsilon K$
$s_{ij} \leftarrow F_{ij}$	for all $\{i,j\} \varepsilon A$
$v^k_i \leftarrow$ shortest path length from O(k) to node i	for all $i \varepsilon N$, $k \varepsilon K$
$N_1(k) \leftarrow N \backslash \{D(k)\}$	for all $k \varepsilon K$
$N_2(k) \leftarrow D(k)$	for all $k \varepsilon K$
$Z_D \leftarrow v^k_{D(k)}$.	

Set CANDIDATES = $\{k \varepsilon K : O(k) \varepsilon N_1(k)\}$.

Step 1: Ascent Iterations

Select a commodity $k \varepsilon$ CANDIDATES,
set $A(k) = \{(i,j) : i \varepsilon N_1(k),\ j \varepsilon N_2(k)\}$, and

(a) calculate the amount of w-increase:

$$\text{Set}\quad A'(k) = \{(i,j):\ c^k_{ij} - (v^k_j - v^k_i) \leq 0,\ (i,j) \varepsilon A(k)\},$$
$$= \{(i,j):\ c^k_{ij} + w^k_{ij} - (v^k_j - v^k_i) = 0,\ (i,j) \varepsilon A(k)\},$$

$$\delta_1 = \min \{s_{ij}:\ (i,j) \varepsilon A'(k)\},$$

$$\delta_2 = \min \{c^k_{ij} + w^k_{ij} - v^k_j + v^k_i : (i,j) \varepsilon A''(k) = A(k)\backslash A'(k)\},$$

$$\delta = \min (\delta_1, \delta_2).$$

(b) update relevant w-values, slacks and shortest path lengths:

$w^k_{ij} \leftarrow w^k_{ij} + \delta$ for all $(i,j) \varepsilon A(k)$

$s_{ij} \leftarrow s_{ij} - \delta$ for all $(i,j) \varepsilon A'(k)$

$v^k_l \leftarrow v^k_l + \delta$ for all $l \varepsilon N_2(k)$

$Z_D \leftarrow Z_D + \delta.$

(c) label a new node:

If $\delta = \delta_1$, for some $(i^*,j^*) \varepsilon A'(k)$ satisfying $s_{i^*j^*} = 0$, set

$$N_1(k) \leftarrow N_1(k)\backslash\{i^*\}$$
$$N_2(k) \leftarrow N_2(k) \cup \{i^*\}.$$

Remove commodity k from CANDIDATES and repeat Step 1;

Step 2: Stopping Rule

If $O(k) \varepsilon N_2(k)$ for all $k \varepsilon K$, STOP.

Otherwise, set CANDIDATES = $\{k \varepsilon K : O(k) \varepsilon N_1(k)\}$, and return to Step 1.

Remarks:

(1) By design, the algorithm always maintains dual feasibility: that is,

$$c^k_{ij} + w^k_{ij} - v^k_j + v^k_i \geq 0 \qquad \text{for all } (i,j) \varepsilon A \text{ and } k \varepsilon K.$$

Also, because of the mechanics in Step 1, whenever an arc in A(k) becomes tight, it remains so throughout the execution of the algorithm.

(2) Note that the algorithm only *increases* the values of the w variables. It is possible to show that when the algorithm terminates, the dual objective value can possibly be increased by reallocating fixed costs (by decreasing some w^k_{ij} values and increasing others). This possibility suggests a more elaborate iterative procedure that accounts for such tradeoffs. Our

computational experience in Section 4 shows, however, that simply increasing the w_{ij}^k values, as in this implementation, does surprisingly well.

(3) Step 1 does not specify the order for considering commodities. The implementation that we tested groups together all commodities with a common origin. Thus, it first examines all commodities with node 1 as the origin, next those with node 2 as origin, and so forth. In some preliminary tests for design problems with complete demand (i.e., with required flows between every node pair), we found that employing more sophisticated priority rules for sequencing the commodities in this step did not significantly or consistently improve the performance. Further, as shown in Appendix 1, the commodity-grouping scheme that we adopted, when applied to the Steiner tree problem, gives the same results as Wong's (1984) dual ascent algorithm for this problem.

(4) Step 1(c) labels at most one node in each iteration. In some degenerate cases, the slacks for several arcs might simultaneously reduce to zero in 1(b) making more than one node of $N_1(k)$ eligible for labeling. In such cases, subsequent iterations transfer the remaining 'eligible' nodes to N_2 (k), one at a time, before achieving further ascent. Once again, this scheme generalizes the Steiner tree dual ascent method discussed in Appendix 1.

Properties of the Dual Solutions Produced by the Labeling Method

The intermediate and final dual solutions produced by this ascent algorithm satisfy several important properties.

Property 1 (Shortest Path Property):

(i) *At every step in the algorithm, v_i^k for all $i \varepsilon N$ and $k \varepsilon K$ represents the shortest path distance from origin O(k) to node i.*

(ii) *For every commodity k, all nodes in $N_2(k)$ belong to at least one shortest O-D path for commodity k.*

Initially, all v-values represent shortest path lengths from the origin. By increasing (in Step 1(b)) v_{l}^{k} for all $l \in N_2(k)$ by δ at every iteration, the algorithm ensures that statement (i) is satisfied. Also, the labeling scheme of Step 1(c) ensures that statement (ii) is satisfied. We use an induction argument to prove this second property. Initially, $N_2(k)$ contains only the destination D(k), and hence satisfies this conditon. Consider any intermediate iteration and assume that all nodes currently in $N_2(k)$ lie on at least one shortest O-D path for commodity k. A node i is labeled (i.e., transferred from $N_1(k)$ to $N_2(k)$) in this iteration only if one of the incident arcs, say, (i,j) belongs to cutset A(k), is tight, and has its slack reduced to zero. Since arc (i,j) is tight, equation (2.7) is satisfied, implying that node i, when it is labeled, must lie on at least one shortest O-D path for commodity k (since, by the induction hypothesis, node $j \in N_2(k)$ belongs to at least one shortest O-D path). Because of these two properties, equation (2.7) serves as a necessary as well as sufficient condition for identifying shortest path arcs of the directed cutset A(k).

Property 2 (Zero-Slack Path Property):

For every commodity k, all nodes in $N_2(k)$ are connected to the destination D(k) via shortest paths containing only zero slack arcs.

The previous induction argument can be extended to establish this property. (A node i is added to $N_2(k)$ only if an incident tight arc, say, $(i,j) \in A'(k)$ has zero slack; and this arc must lie on at least one of the shortest paths from node i to D(k).) Labeling the origin O(k), therefore, signals the creation (or existence, in degenerate cases) of a shortest O-D path for commodity k (using the modified arc lengths $\hat{c}_{ij}^{k}$), all of whose arcs have zero slack. Since this path contains only arcs with zero slack, w-values for arcs on this path cannot be increased. Since this path is also a shortest O-D path, increasing w-values corresponding to commodity k on other arcs does not increase the shortest O-D path length for commodity k or, therefore, the dual objective function value. Consequently, when the algorithm terminates, i.e., when all origins are labeled, any further increases in w-values, with respect to the final dual solution, cannot improve the final lower bound Z_D. Further, since every commodity has a zero-slack shortest O-D path, the design consisting of all zero slack arcs is feasible, i.e., every origin-destination pair is connected in this design. We use this solution as the starting point for a heuristic procedure that determines a locally optimal upper bound (see Section 3.3).

Property 3 (Minimum Allocation Property):

After every ascent step, the w-values satisfy the condition

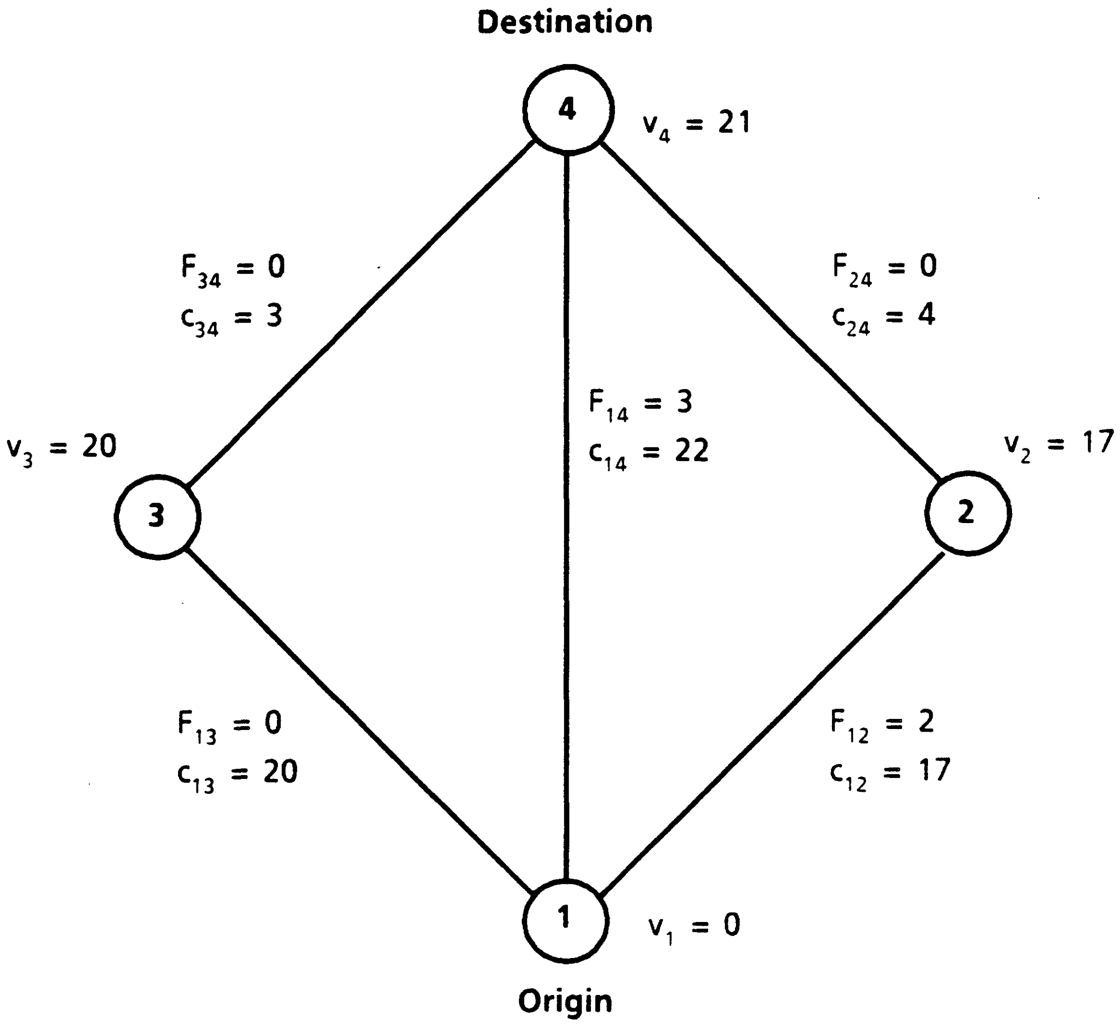
$$\begin{aligned} w_{ij}^k &= \max\{0, v_j^k - v_i^k - c_{ij}^k\} && \textit{for all } \{i,j\} \in A, \textit{ and} \\ w_{ji}^k &= \max\{0, v_i^k - v_j^k - c_{ji}^k\} && \textit{all } k \in K. \end{aligned} \qquad (2.8)$$

This property implies that w_{ij}^k is positive only if arc (i,j) is tight, that is $v_j^k - v_i^k - c_{ij}^k \geq 0$ (and hence is a shortest path arc with respect to the modified costs) for commodity k. Thus, the ascent procedure is parsimonious in allocating the slacks s_{ij} toward increasing the w_{ij}^k values. (Indeed, for a given set of v-values, the expression on the right-hand side of (2.8) is the minimum permissible value of w_{ij}^k needed to ensure feasibility in constraints (2.2) of DP_1.) Also, because of this characteristic, every intermediate dual solution is completely specified by the current values of v_i^k; thus, the algorithm need not explicitly store and update the values of w_{ij}^k.

2.4 Dual Ascent Example using the Labeling Method

To illustrate the operation of the Labeling method, consider the example in Figure 2.1. For this example, $F_{14} = 3$ and $F_{12} = 2$; all other fixed costs are zero. There is only one commodity, with origin O(1) = 1 and destination D(1) = 4. The variable costs are the same in both directions and are shown in Figure 2.1. (For simplicity, we have eliminated the superscripts referring to

Figure 2.1
Dual Ascent Labeling Method Example



commodity 1 in the figure.) We wish to increase v_4, starting with all w-values equal to zero and v-values as shown in the figure.

Initially, $N_2(1) = \{4\}$, $N_1(1) = \{1,2,3\}$, $s_{14} = F_{14} = 3$, $s_{12} = F_{12} = 2$, $s_{24} = s_{13} = s_{34} = 0$ and $Z_D = 21$. The first execution of Step 1 yields $A'(1) = \{(2,4)\}$, $\delta = s_{24} = 0$ and Z_D remains 21; $N_2(1)$ becomes $\{2,4\}$ and $N_1(1)$ becomes $\{1,3\}$. Node 2 is labeled in this iteration. The second pass through Step 1 gives $A'(1) = \{(1,2)\}$ and $\delta = \delta_2 = c_{14} - (v_4 - v_1) = 22 - (21 - 0) = 1$. The objective function $Z_D = v_4$ becomes 22, and the slack for arc $\{1,2\}$ decreases to $s_{12} = F_{12} - \delta = 2 - 1 = 1$. Node 2 is labeled in this iteration. The third iteration computes $A'(1) = \{(1,4),(1,2)\}$ and $\delta = \delta_1 = s_{12} = 1$. The objective function $Z_D = v_4$ increases to 23 and $N_2(1) = \{1,2,4\}$. Since node 1 is labeled in this iteration, i.e., $Q(1) = 1 \varepsilon N_2(1)$, the dual ascent procedure terminates with a dual solution value of 23. (For this example, the dual objective function value equals the optimal value of the design problem.)

2.5 Interpretations of the Dual Ascent Labeling Method

In Section 2.1, we interpreted dual ascent labeling method as a shortest path expansion technique. In fact, as the following discussion demonstrates, the method can be viewed as a generalization of Dijkstra's shortest path labeling technique. Suppose we wish to identify the shortest path from node 1 to node n in a given network. Consider an equivalent network design problem with a single commodity, say commodity 1, originating at node 1 and destined for node n. Set all routing costs c^k_{ij} equal to zero and fixed costs F_{ij} equal to the given arc lengths. Then the optimal network design solution consists of the shortest path from node 1 to node n.

For the equivalent network design problem, let us examine the sequence that the Labeling method labels nodes. First, we note that, since all routing costs are zero, every arc (i,j) contained in the directed cutset A(k) corresponding to the node partition $N_1(k)$ and $N_2(k)$ is tight, i.e., if $i \varepsilon N_1(k)$ and $j \varepsilon N_2(k)$, then $(i,j) \varepsilon A'(k)$. At every iteration, therefore, $\delta = \delta_1$ and one node is labeled (assuming, for simplicity, that $\delta_1 = s_{ij}$ for a unique arc (i,j) in Step 1(a) of the labeling method). In particular, the node that is labeled at the p^{th} iteration is the p^{th} closest node to the destination n; and, the nodes in $N_2(k)$ correspond to those for which the shortest paths from the destination n have been definitively identified. Careful examination shows that the order in which nodes are added to $N_2(k)$ is exactly the same as the order in which Dijkstra's algorithm assigns permanent node labels when it starts from node n and seeks the shortest path to node 1.

The dual ascent labeling algorithm also generalizes several other procedures. When the method is applied to directed network design models, it can be viewed as a generalization of Wong's (1984) dual ascent technique for Steiner tree problems on a directed graph. Appendix 1 establishes the connection between the Steiner tree and network design dual ascent algorithms.

The ascent procedure also generalizes Edmonds' directed spanning tree algorithm and Erlenkotter's ascent procedure for uncapacitated facility location since Wong (1984) has shown that his procedure generalizes these two methods.

Thus, just as the network design model contains a number of well-known network optimization problems as special cases, the network design ascent procedure generalizes the algorithms proposed for a number of these special cases including the Steiner tree, uncapacitated facility location, shortest path and directed spanning tree problems.

Finally, the methods discussed in this section extend easily to the directed network design case as well. The primary difference for directed problems is that w^k_{ij} and w^k_{ji} 'use up' separate fixed charges F_{ij} and F_{ji}, respectively (assuming the given network contains both the directed arcs (i,j) and (j,i)), instead of competing for the same fixed charge F_{ij}.

In summary, we have outlined a general dual ascent framework for the network design problem and discussed two specific implementations of the general strategy. The two implementations differ primarily in the method used to identify arc-commodity combinations for increasing w-values. Balakrishnan (1984) discusses other alternative implementations suggested by this dual ascent framework including a complementary method for adjusting the v-values without decreasing the dual objective function value in order to increase the slack variables for selected arcs, thereby permitting more ascent.

In the next section, we discuss several enhancements for the labeling method to account for the stronger forcing constraints (1.6) and (1.7) discussed in Section 1.2, provide an initial solution for local improvement network design heuristics, and permit problem reduction by eliminating arcs.

3. Enhancements and Applications of the Dual Ascent Algorithm

In this section, we discuss several enhancements of the dual ascent labeling method presented in Section 2.3. After describing a modified ascent procedure that exploits the improved network design formulation discussed in Section 1.2, we outline an enhancement that utilizes information provided by a feasible integer design solution and offers an opportunity to further improve the lower bound.

A good dual solution naturally provides a good lower bound for the optimal design solution. The last two subsections discuss additional ways of exploiting a good dual solution; we discuss a way of constructing a feasible design solution from a given dual solution and also describe problem reduction tests that eliminate unnecessary variables. These tests are based upon information provided by the dual solution.

3.1 Modified Dual Ascent Procedure for an Improved Formulation

This section describes a modification that takes advantage of the improved network design formulation P_2 given in Section 1.2.

Recall that we can substitute (1.6) and (1.7) for the forcing constraints (1.3) and obtain a formulation with a stronger linear programming relaxation than the usual formulation (1.1) - (1.4). In order to simplify our implementation, whenever two commodities k and h have the same *origin*, we substitute only the following inequalities for (1.3).

$$x^k_{ij} + x^h_{ji} \le y_{ij} \qquad \text{for all } \{i,j\}\varepsilon A \text{ if } O(k) = O(h). \qquad (1.7)$$

(We do not add the corresponding inequalities (1.6) when two commodities have the same *destination*.) Replacing (1.3) by (1.7) gives a formulation P_3 (consisting of (1.1), (1.2), (1.7), and (1.4)) that has a tighter linear programming relaxation than P_1 (containing (1.1) - (1.4)).

The dual of the linear programming relaxation of P_3, denoted as DP_3, is

[DP$_3$]

$$\text{maximize} \quad \sum_{k} v^{k}_{D(k)} \tag{3.1}$$

subject to

$$v^{k}_{j} - v^{k}_{i} - \sum_{h \varepsilon S(k)} u^{kh}_{ij} \leq c^{k}_{ij} \qquad \text{for all } k \varepsilon K, \text{ and} \tag{3.2a}$$

$$v^{k}_{i} - v^{k}_{j} - \sum_{h \varepsilon S(k)} u^{kh}_{ij} \leq c^{k}_{ji} \qquad \text{all } \{i,j\} \varepsilon A, \tag{3.2b}$$

$$\sum_{k \varepsilon K} \sum_{h \varepsilon S(k)} u^{kh}_{ij} \leq F_{ij} \qquad \text{for all } \{i,j\} \varepsilon A, \tag{3.3}$$

$$u^{kh}_{ij} \geq 0 \qquad \text{for all } k \varepsilon K,\ h \varepsilon S(k), \text{ and all } \{i,j\} \varepsilon A, \tag{3.4}$$

where $S(k) = \{h\colon h \varepsilon K,\ h \neq k, \text{ and } O(h) = O(k)\}$,

v^{k}_{i} = dual variable corresponding to the flow conservation equation (1.2) at node i for commodity k, and

u^{kh}_{ij} = dual variable corresponding to the forcing constraint (1.7) for arc {i,j} and commodities k and h ε S(k).

The dual ascent strategy remains unchanged since we wish to modify the v^{k}_{i} values in order to maximize the dual objective function. The only difference between this procedure and the previous algorithm is that instead of changing the w^{k}_{ij} and w^{k}_{ji} variables we update the u^{kh}_{ij} variables. For this new dual problem, we can define the modified arc length (see Section 2.1) as

$$\tilde{c}^{k}_{ij} = c^{k}_{ij} + \sum_{h \varepsilon S(k)} u^{kh}_{ij}, \quad \text{or}$$

$$\tilde{c}^k_{ji} = c^k_{ji} + \sum_{h\varepsilon S(k)} u^{kh}_{ij}, \qquad \text{for all } \{i,j\} \varepsilon A. \qquad (3.5)$$

Recall from Section 2.1 that increasing variable w^k_{ij} increases the modified arc length $\hat{c}^k_{ij}$. For the current dual problem if we increase u^{kh}_{ij}, then the two modified arc lengths $\tilde{c}^k_{ij}$ and $\tilde{c}^h_{ij}$ increase. So increasing $\tilde{c}^k_{ij}$ also increases $\tilde{c}^h_{ij}$ for some $h \varepsilon S(k)$ without an additional decrease in the slack s_{ij}. Essentially, we are able to increase $\tilde{c}^h_{ij}$ for 'free'.

To increase $\tilde{c}^k_{ij}$ we must select a commodity $h \varepsilon S(k)$ and increase u^{kh}_{ij}. Our implementation delays this decision. Suppose we increase the modified arc length $\tilde{c}^k_{ij}$ by an amount Δ. We save this information until we need to increase an arc length $\tilde{c}^h_{ji}$, for some $h \varepsilon S(k)$. Then we can increase $\tilde{c}^h_{ji}$ up to an amount Δ without any decrease in the slack variables.

Note that this delay in selecting u^{kh}_{ij} allows us to exploit constraint (1.7) which is an 'expanded' version of the forcing constraint (1.3). The ascent procedure for the expanded dual problem is more flexible than the ascent procedure for the dual of LP_1 because the expanded version can increase some shortest path lengths with a possibly smaller decrease in the slacks; therefore, the enhancement should generally produce tighter lower bounds.

3.2 Dual Ascent and Feasible Network Design Solutions

Another way to improve the performance of the dual ascent algorithm is to use information provided by a good integer solution to the network design problem. Suppose we have an optimal solution to the design problem and the optimal integer solution has the same value as the optimal linear programming solution; then any optimal dual solution and optimal integer solution must satisfy the linear programming complementary slackness conditions. So we can use the optimal integer solution as a guide for constructing a dual solution by ensuring that the complementary slackness conditions are satisfied. Given a feasible integer solution (see Section 3.3 for methods to generate integer solutions) and a solution produced by the dual ascent procedure, we can find all complementary slackness violations. As shown in Section 2.3, when the dual ascent labeling procedure terminates, the design consisting of only zero-slack arcs is feasible. For this design, solving a shortest path problem for each commodity $k \varepsilon K$ gives the arc flows. With respect to this network design solution (with $y_{ij} = 1$ if $s_{ij} = 0$), the only complementary slackness condition that can be violated is

$$\begin{aligned} w_{ij}^{k}(x_{ij}^{k} - y_{ij}) &= 0 \qquad \text{for all } k \varepsilon K, \text{ and} \\ w_{ji}^{k}(x_{ji}^{k} - y_{ij}) &= 0 \qquad \text{all } \{i,j\} \varepsilon A. \end{aligned} \tag{3.6}$$

These constraints impose only one restriction: if $y_{ij} = 1$ and $x_{ij}^{k} = 0$ ($x_{ji}^{k} = 0$) then w_{ij}^{k} (w_{ji}^{k}) must be zero. Let IJK be the indices of all w_{ij}^{k} variables that must be zero in order to satisfy the complementary slackness condition (3.6). To enforce the complementary slackness conditions (3.6), we re-execute the dual ascent procedure with the added constraint that for all $(i,j,k) \varepsilon$ IJK, $w_{ij}^{k} = 0$. Incorporating these constraints into the ascent procedure is straightforward. Whenever attempting to increase w_{ij}^{k} for any $(i,j,k) \varepsilon$ IJK, we effectively assume that $F_{ij} = 0$; so w_{ij}^{k} will remain at value zero. After running this restricted dual ascent procedure, we take the final dual solution generated and use it as an initial solution to the regular unrestricted version of the dual ascent procedure.

Hopefully, the added restrictions imposed by the complementary slackness conditions will allow us to find a better dual solution. (However, the new dual solution is not guaranteed to provide a better lower bound than the previous dual solution.) Note that if the new dual solution generates another integer solution, we can repeat the proposed dual ascent enhancement. This iterative approach is similar to the one proposed by Prodon, Liebling, and Groflin (1985) for the Steiner tree problem.

3.3 Dual-based Heuristic Procedure

As noted in the last section, the dual solution generated by the ascent algorithm can be used to produce a feasible integer solution. This initial solution can be improved using add-drop type procedures (see Billheimer and

Gray or Los and Lardinois). The add (drop) procedure starts with an initial network design solution and at each iteration adds (drops) an arc to (from) the current design in order to improve the total design cost. The procedure continues until no arc can be added (dropped) from the solution without increasing the total design cost. Note that instead of constructing the initial design using the dual solution, we might consider other methods for generating initial feasible solutions for the local improvement heurisitic. For instance, an initial solution consisting of all possible arcs is often used to initialize the drop heuristic (see, Billheimer and Gray). Previous experience with various network design heuristics (see, Ellman (1983)) suggests that the cost of the networks produced by the local improvement procedure with different starting solutions is about the same. However, the computation time of the add-drop heuristic improves drastically when the initial design is produced from the dual solution. In addition, the dual solution provides a lower bound and hence a performance guarantee on the quality of the feasible solution generated by the heuristic.

Intuitively, this improvement in running time is not surprising. The dual solution takes into account all the design problem data. Thus, a design constructed from the dual solution utilizes a great deal of problem information. On the other hand, an initial solution consisting of all possible arcs does not utilize any special problem parameters. It is, therefore, reasonable to expect that the dual-based solution will provide a superior starting point for the local improvement procedure.

3.4 Dual-based Problem Reduction Methods

In addition to providing lower bounds and initializing local improvement heuristics, dual solutions also enable us to identify arcs of the given network that must or must not necessarily belong to the optimal design. Fixing the design variables (to 1 or 0, respectively) corresponding to these arcs not only reduces subsequent computational effort for solving the problem, but also improves the quality of the upper and lower bounds on the optimal value. This section describes two problem reduction tests that use any intermediate dual solution and a known upper bound $\hat{Z}$ (say, the cost of the current best heuristic solution). These preprocessing tests can be viewed as dual-based penalty methods similar to those employed for fathoming vertices in a branch-and-bound scheme (see, for example, Geoffrion (1974)).

Suppose we want to determine if the optimal design should contain some arc $\{i,j\}$. Consider the effect, on the current dual objective function value Z_D, of forcing arc $\{i,j\}$ to be in the design by adding the constraint

$$y_{ij} = 1 \tag{3.7}$$

to problem P_1. Let μ_{ij} be the dual variable corresponding to this constraint. Then, the dual constraint (2.3) corresponding to arc $\{i,j\}$ becomes

$$\sum_k w_{ij}^k + \sum_k w_{ji}^k + \mu_{ij} \leq F_{ij}, \tag{3.8}$$

and the revised dual objective function contains the additional term + μ_{ij}. Let s_{ij} denote the slack for the constraint (2.3) corresponding to arc $\{i,j\}$ in the final solution to DP_1 constructed by the dual ascent algorithm. Then, setting μ_{ij} equal to s_{ij} (and reducing the slack to zero) gives a feasible solution to the revised dual problem DP_3; this dual solution has an objective function value of $(Z_D + s_{ij})$. Therefore, the optimal value of the network design problem when constraint (3.7) is added to P_1 must be at least $(Z_D + s_{ij})$. Clearly, if $(Z_D + s_{ij})$ exceeds $\hat{Z}$, the current upper bound, then the optimal network design solution must violate constraint (3.7) implying that the optimal design cannot contain arc $\{i,j\}$. Thus, if

$$(\hat{Z} - Z_D) < s_{ij}, \tag{3.9}$$

then arc $\{i,j\}$ can be excluded, i.e., y_{ij} must be 0 in the optimal network design solution. We refer to this method as the Arc Exclusion Test.

We can similarly devise an Arc Inclusion test that identifies arcs necessarily belonging to the optimal design. For any arc $\{i,j\}$, let Δ_{ij}^k represent the increase in the shortest path length for commodity k (using $\hat{c}_{ij}^k = c_{ij}^k + w_{ij}^k$ as arc lengths) when arc $\{i,j\}$ is deleted from the network. If the sum $\Sigma_k \Delta_{ij}^k$ exceeds $(\hat{Z} - Z_D)$, then the optimal design must necessarily contain arc $\{i,j\}$. This test may be computationally expensive since, for each arc, it requires reevaluation of the shortest path lengths for all those commodities that currently use this arc in their shortest paths. Similar dual-based tests can be applied to determine whether or not a given commodity k must flow on a certain arc {i,j}.

To summarize, the basic dual ascent labeling method can be applied iteratively to generate good dual solutions and lower bounds. The dual solutions give feasible designs that serve as starting points for a local improvement heuristic. In turn, the upper bound thus generated, in conjunction with the dual lower bound, enables variable elimination. Thus, the enhancements discussed in this section lead to a *composite* algorithm, driven by the dual ascent labeling method, for identifying good network design solutions, for verifying the quality of these solutions through lower bounds, and for reducing the problem. In the next section, we discuss some implementation details and computational results for such a composite algorithm that incorporates the iterative dual ascent, heuristic and problem reduction features.

4. Computational Results

We implemented the dual ascent algorithm and the heuristic procedure in FORTRAN on an IBM 3083 (Model BX) and performed extensive computational tests using two types of problems ranging in size from 20 nodes, 80 arcs and 380 commodities (80 integer variables and 60,800 continuous variables and forcing constraints) to 45 nodes, 500 arcs and 1980 commodities (500 integer variables and 1,980,000 continuous variables and forcing constraints). In all, we tested 106 problem instances. We did not attempt to solve these problems to optimality (for example, by embedding the dual ascent algorithm in a branch-and-bound scheme); for almost all instances, the gap between our dual-based upper and lower bounds, expressed as a percentage of the lower bound, was less than 3 %.

The first set of test problems are defined over random undirected networks with "complete" demand (i.e., one commodity for every node pair) and arc costs (both fixed and variable) that are proportional to the Euclidean distance between the end points of the arc. For this problem type, Section 4.1 describes some salient features of our dual ascent implementation and presents computational results for various network sizes, arc densities and fixed-to-variable cost ratios.

The second class of test problems consist of some realistic directed, "Less-than-Truckload Consolidation" problems considered by Lamar, Sheffi and Powell (1984). Section 4.2 discusses the special characteristics of this problem type and presents computational results using the dual ascent method for Lamar et al.'s 7 test problems.

4.1 General Undirected Test Problems: Random, Undirected Networks with Complete Demand and Euclidean Costs

We used the following problem generating program to construct test problems in this category. The network generator first selects the required number of nodes (specified by the user) from a 100 x 100 grid in the plane and generates a random spanning tree over these nodes (to ensure that the problem is feasible). The required number of remaining arcs are then randomly selected using a method described in Knuth (1969). For every directed arc (i,j), the variable cost c_{ij}^{k} is the same for all commodities k and is set equal to the Euclidean length of arc {i,j} (rounded to the nearest integer). The fixed charge F_{ij} is some user-specified multiple of the variable cost. All problems have a complete demand pattern, i.e., one unit must be transported from each node to every other node. Transshipment is permitted at every node.

For our experiments we generated these problems in eleven sizes ranging from 20 nodes, 80 arcs and 380 commodities to 45 nodes, 500 arcs and 1980 commodities. Table I shows the dimensions of the network for each of the eleven problem sizes. Problems in categories 1 to 6 represent sparse networks with varying levels of arc densities. Problem sizes 7 to 11, on the other hand, correspond to complete networks.

For each problem size, we considered three realistic fixed-to-variable cost ratios of 2, 10 and 15; as the fixed charge increases in this range relative to the variable cost, we expect that the problems will become harder to solve. We generated three instances, using different random number seeds, for each of the 33 problem combinations.

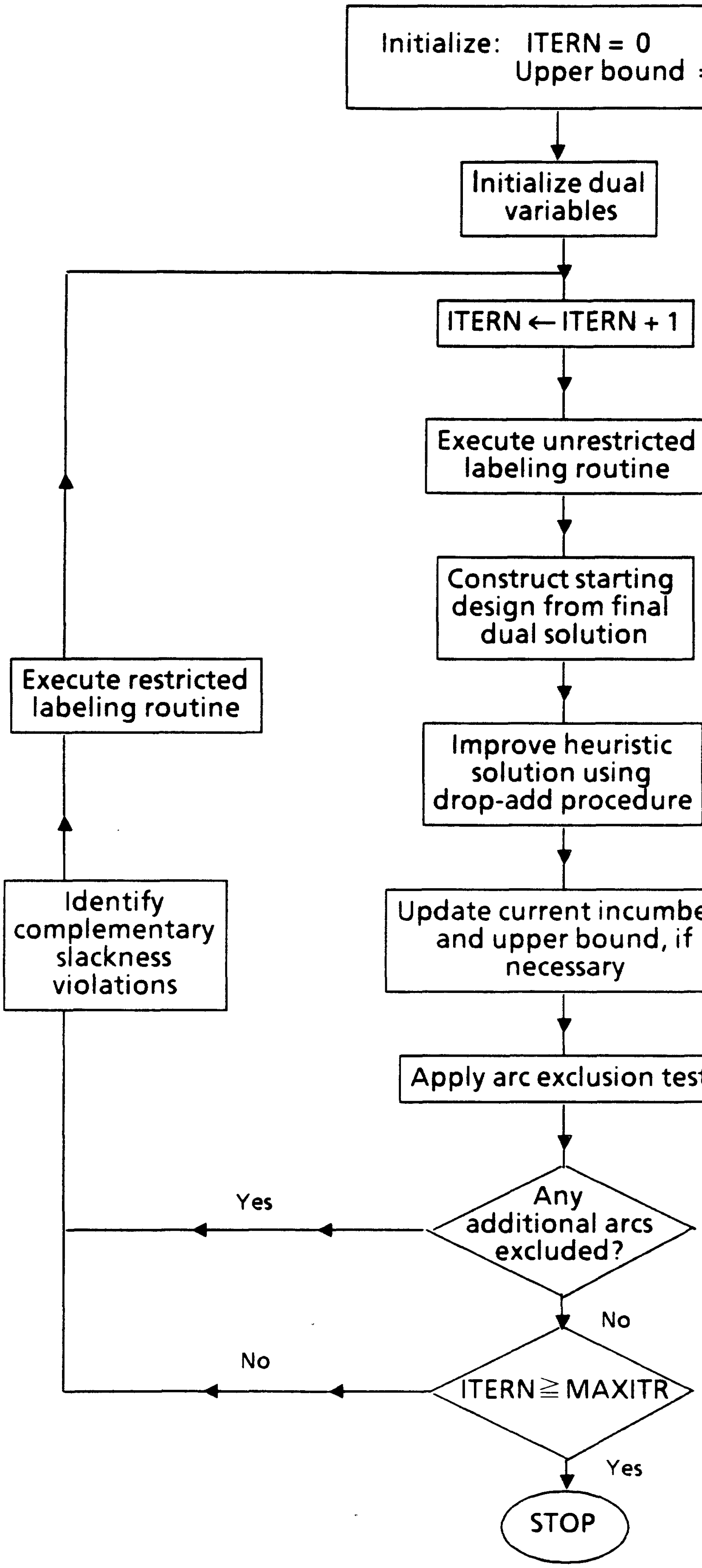
Implementation Details

Our FORTRAN implementation of the composite dual ascent-based algorithm to solve this class of undirected network design problems incorporates the enhancement and the iterative dual ascent scheme based on complementary slackness violations described in Sections 3.1 and 3.2, as well as a drop-add heuristic that uses an initial design derived from the dual solution. (The drop-add method first drops arcs successively as long as the total cost decreases before initiating the add phase.) The main subroutines in the program are (i) an "unrestricted" dual ascent routine, (ii) a "restricted" dual ascent routine that takes into account complementary slackness restrictions, and (iii) the drop-add heuristic procedure. The only problem reduction test that we implemented was the Arc Exclusion method to eliminate arcs {i,j} whose remaining slack s_{ij} exceeds the difference between the current best upper and lower bounds. Figure 4.1 contains a flowchart showing the interrelationships between the various segments of the composite program.

Table I
Network Dimensions for General, Undirected Test Problems

Problem number	No. of nodes	No. of arcs	No. of commodities	Number of variables in formulation P_1	
				Integer	Continuous
1	20	80	380	80	60,800
2	25	100	600	100	120,000
3	30	130	870	130	226,200
4	35	150	1190	150	357,000
5	40	400	1560	400	1,248,000
6	45	500	1980	500	1,980,000
		Complete Networks			
7	15	105	210	105	44, 100
8	20	190	380	190	144,400
9	25	300	600	300	360,000
10	30	435	870	435	756,900
11	35	595	1190	595	1,416,100

Figure 4.1
Flowchart for Dual-based Composite Algorithm


Initialize: ITERN = 0
Upper bound = ∞
Initialize dual variables
ITERN ← ITERN + 1
Execute unrestricted labeling routine
Construct starting design from final dual solution
Improve heuristic solution using drop-add procedure
Update current incumbent and upper bound, if necessary
Apply arc exclusion test
Any additional arcs excluded?
Yes
No
ITERN ≧ MAXITR
No
Yes
STOP
Identify complementary slackness violations
Execute restricted labeling routine

The program uses integer arithmetic and a forward-star representation (see, for example, Magnanti and Golden (1978)) of the network. Other data structures employed by the program include an array to flag the labeled nodes and a logical array to identify, for each commodity k, the set of tight cutset arcs A'(k). The dual ascent routines iteratively update the latter array rather than recomputing the set A'(k) every time (as indicated in Step 1 of the Labeling method). This set is updated in the following manner. In Step 1 of the Labeling method, a new arc becomes tight if $\delta = \delta_2$. In such a case, the status of the variable corresponding to this arc in the logical array for A'(k) is changed. This updating step adds at most only one arc to A'(k) at each iteration. Therefore, in some degenerate cases when two or more arcs become tight simultaneously, A'(k) may not contain all the tight cutset arcs; the other arcs are added to A'(k) successively in subsequent iterations. (This scheme is similar to the labeling strategy in Step 1(c) which labels at most one node in each iteration.)

As mentioned at the end of Section 2.3, at every stage the w-values are completely specified for given v-values (using equation (2.8)); therefore, our implementation does not explicitly store or update the w-values. Also, we evaluate the complementary slackness restrictions using the best solution derived by the drop-add heuristic rather than the initial design constructed from the final dual ascent solution.

The drop-add heuristic uses a modified version of Dijkstra's shortest path algorithm (suggested by Dionne and Florian) to efficiently update the commodity routings when arcs are dropped or added from the design. The method maintains and periodically updates a list of candidate arcs to be dropped (added), arranged in order of decreasing cost savings. The sequence in which arcs are dropped (added) follows the order in this list. (See Los and Lardinois for a similar technique.)

The program permits the user to specify a parameter MAXITR that influences the number of dual ascent cycles that are performed. By a dual ascent *cycle* we mean a complete execution of the dual ascent routine (until all commodity origins are labeled), heuristic procedure, and problem reduction test. The program initially executes the unrestricted dual ascent routine, constructs an initial design from the dual solution and identifies a heuristic solution using the drop-add procedure. From the second cycle onwards, the restricted dual ascent routine, with complementary slackness restrictions derived from the last cycle, precedes the unrestricted version. The drop-add procedure is rerun with an initial design derived from the latest dual solution and new complementary slackness restrictions are identified. The process continues for MAXITR cycles. The procedure continues with additional dual ascent cycles if, during the last cycle, the Arc Exclusion test eliminated one or more arcs.

The FORTRAN program was compiled with level 2 optimization using the VS FORTRAN compiler on the IBM 3083 (Model BX). Table II summarizes our results for 99 runs: for 11 problem sizes, with 3 fixed-to-variable cost ratios for each size, and 3 random replications for each ratio. After some initial testing, we decided to set the parameter MAXITR to 2 for each run (since additional 'required' dual ascent cycles did not seem to improve the bounds in most problem instances).

Interpretation of Computational Results

As expected, the algorithm performs much better for lower fixed-to-variable cost ratios both in terms of the quality of the bounds and required computation time. For problems with a ratio of 2, the average optimality measure ε is less than 0.38% for all problem sizes; when this ratio increases to 10, the average value of ε ranges from 1.00% to 2.24%. The effectiveness of the arc exclusion test is very sensitive to the magnitude of the gap between the upper and lower bounds. As the problem size and the fixed-to-variable cost ratio increase, this test deletes a smaller percentage of arcs. Also, for higher fixed-to-variable cost ratios, the proportion of arcs eliminated tends to be higher for dense networks (problem sizes 7 to 11).

In general, the CPU time increases with the problem size and the fixed-to-variable cost ratio. However, in some cases (for example, for problem sizes 2, 4, and 5) the average CPU time is lower for a higher ratio. This phenomenon is attributable to the fact that our program executes additional dual ascent cycles (beyond the specified limit of 2 dual ascent cycles) whenever the reduction test deletes one or more arcs. The CPU time per ascent cycle is larger for higher fixed-to-variable cost ratios in all cases. The

Table II
Summary of Dual Ascent Results for General Undirected Test Problems

Problem size	FC/VC ratio	Avg.% gap*+	Avg.* CPU time*&		Avg.* no. of arc deleted	Avg.* no. of ascent cycles
			Total§	%Ascent#		
	2.0	0.09	2.65	54.3	40.7	2.67
1	10.0	1.00	4.33	62.4	33.3	3.33
	15.0	1.94	5.04	64.5	21.3	3.00
	2.0	0.08	6.91	47.8	40.7	3.67
2	10.0	1.09	9.18	62.5	21.3	3.67
	15.0	1.97	7.78	57.8	11.0	2.67
	2.0	0.14	9.46	42.0	36.0	2.67
3	10.0	1.06	13.04	57.7	15.7	2.67
	15.0	1.68	14.21	61.1	3.3	2.67
	2.0	0.10	21.81	39.0	37.7	3.67
4	10.0	1.35	15.03	46.8	0	3.67
	15.0	2.12	19.69	48.9	0	2.00
	2.0	0.24	50.51	37.1	2	2.33
5	10.0	1.52	81.67	44.8	1	2.33
	15.0	2.33	79.80	42.0	0	2.00
	2.0	0.22	80.81	36.0	0.3	2.33
6	10.0	1.79	120.89	35.4	0	2.00
	15.0	2.30	150.26	35.1	0	2.00
	2.0	0.33	3.26	55.8	60	3.67
7	10.0	1.97	3.77	63.9	52	3.67
	15.0	2.29	4.08	64.2	57.3	3.3
	2.0	0.38	7.20	51.4	73.3	3
8	10.0	2.24	10.74	69.9	42	3
	15.0	3.12	14.72	60.9	45.3	3.33
	2.0	0.27	18.95	53.0	106	3.33
9	10.0	1.85	35.19	55.2	21	3.33
	15.0	2.44	37.19	62.1	52.7	3.33
	2.0	0.38	43.71	46.4	24.7	3
10	10.0	1.68	57.78	59.6	25	3
	15.0	2.34	68.95	62.5	17.7	3
	2.0	0.28	117.70	40.5	11	4
11	10.0	1.93	78.70	54.3	0	2
	15.0	2.49	87.40	57.4	0	2

* Average over 3 problem instances.

& CPU time in seconds on an IBM 3083 (Model BX).

§ Total CPU time includes time for initialization, dual ascent, heuristic, and input/output operations. The time for initialization and I/O operations was less than 2.5% of the total CPU time for all problems.

# Time required for dual ascent as a percentage of the Total CPU time.

+ Percentage gap = (Upper Bound - Lower Bound)/Lower Bound %.

breakdown of CPU time indicates that the ascent and heuristic phases require comparable computational effort. For smaller problems (problem sizes 1, 2, and 3), the CPU time per cycle for ascent increases faster with the fixed-to-variable cost ratio than the CPU time per cycle for the heuristic procedure. A detailed examination of the outputs revealed that the relative improvement in the upper bound was significantly lower for the Add phase than for the Drop phase.

In addition to these test results, Magnanti, Mireault, and Wong (1986) present computational experience for over 40 randomly generated, undirected networks of a similar type ranging in size from 10 to 33 nodes and 45 to 130 arcs. While Magnanti et al. focused on solving the network design problem optimally using Benders' decomposition, they used dual ascent in a preprocessing routine to generate lower and upper bounds for problem reduction. Their test networks differ from the problems considered here in the following three respects:

(i) Instead of a complete (or "all-to-all") demand pattern, the problems assume a sparser "two-to-all" demand pattern. One unit of demand is assumed from each of two designated sources to every other node.

(ii) The test problems model a variety of cost structures. Variable costs include both a component related to the Euclidean arc length and a uniform random component. Fixed costs are either directly or inversely proportional to the variable costs and for some problems contain an additional uniform random perturbation.

(iii) The problems belong to the class of *network improvement* models: a randomly selected subset of arcs is assigned zero fixed costs. These arcs represent links of the given network that already exist and that must be included in all candidate designs.

Magnanti et al. report that for all but 2 of their test problems, the ascent procedure gave upper and lower bounds that differed by at most 2.3%. These results complement and confirm our experience about the effectiveness of the dual-based procedures for solving general, undirected network design problems.

4.2 Directed LTL Test Problems: Less-than-Truckload Consolidation Problems

Many distribution contexts involve transporting relatively small quantities of material between numerous widely dispersed geographical locations. In such cases, the total transportation cost can often be reduced by exploiting the economies of scale in the cost structure. This saving is accomplished by consolidating the small less-than-truckload shipments close to the respective points of origin, dispatching full truckloads to downstream breakbulk centers and distributing the individual shipments locally. Lamar, Sheffi and Powell (1984) modeled this less-than-truckload (LTL) consolidation decision as a network design problem and proposed a method that successively strengthens the weak linear programming relaxation (with the aggregate forcing constraints (1.5) instead of (1.3) of formulation P_1) to generate lower bounds. In this section we discuss computational results for Lamar et al.'s 7 test problems using our dual ascent procedure.

We first briefly review the distinctive characteristics of Lamar et al.'s test problems. The LTL network contains two types of nodes: end-of-line terminals that serve as origins and destinations, and transshipment (consolidation and break-bulk) points. Transshipment is not permitted at the terminal nodes. The network is directed and complete, i.e., it contains one directed arc for every pair of nodes. Both the fixed and variable costs are directly proportional to the Euclidean arc lengths. Every pair of terminal nodes defines a commodity whose demand is determined using a gravity-type model. Lamar et al. tested problems containing either 10 or 40 terminal nodes and either 2 or 6 transshipment points. Table III shows the problem dimensions for the 7 test problems labeled LSP1 to LSP7. Observe that problems LSP4 to LSP7 have the same dimensions; however, they differ in the demand level, with problems LSP4 and LSP6 corresponding to the medium demand case and problems LSP5 and LSP7 having low and high demand, respectively. (The demand parameters of problem LSP6 were divided (multiplied) by a factor of 50 (5) to obtain the low (high) demand instance LSP5 (LSP7).)

Since our network design model assumes unit demands for each commodity, we transform the LTL problem with commodity-dependent demands to the unit demand form by scaling the variable costs for each commodity (so that flow now measures percentage of demand). Observe that, unlike the first class of

Table III
Network Dimensions for LTL Consolidation Problems

Problem*	Number of				Average** FC/VC ratio	No. of variables in formulation P_1	
	Terminal nodes	Transshipment nodes	Directed arcs	Commodities		Integer	Continuous
LSP 1	10	2	132	90	2.38	42	630
LSP 2	10	6	240	90	2.25	150	3,870
LSP 3	40	2	1722	1560	3.26	162	10,920
LSP 4	40	6	2070	1560	3.20	510	67,080
LSP 5	40	6	2070	1560	114.43	510	67,080
LSP 6	40	6	2070	1560	2.29	510	67,080
LSP 7	40	6	2070	1560	0.46	510	67,080

* From Lamar, Sheffi and Powell (1984)

** For equivalent unit demand network design problem.

problems, variable costs are now different for different commodities. Also, in transforming the problem to an equivalent unit demand problem, decreasing the demand parameter effectively increases the fixed-to-variable cost ratio. The last column in Table 3 contains the average fixed-to-variable cost ratio for each of the 7 equivalent test problems. Finally, the integer programming formulation for the transformed LTL problem contains fewer integer variables than the total number of arcs in the network. This reduction in the number of integer variables is possible because each direct terminal-to-terminal arc (i,j) can carry only the commodity flowing from i to j (since transshipment is not permitted at terminal nodes). Effectively, therefore, for each of these arcs the fixed charge can be added to the flow cost (since the total flow on this arc will be only either 0 or 1). Consequently, only arcs incident to or from the transshipment nodes require design variables. For example, of the 2070 arcs in problem LSP4, 1560 (= 40 x 39) are direct terminal-to-terminal arcs that have effectively no fixed charge; hence, the formulation contains only 510 (= 2070 - 1560) design variables. Observe also that this special characteristic eliminates the need for forcing constraints corresponding to the direct terminal-to-terminal arcs.

Implementation Details

The LTL-consolidation problems are defined over directed networks, have commodity-dependent variable costs and prohibit transshipment at the origin/destination nodes. We, therefore, had to modify the dual ascent code used for the general, undirected problems in order to handle these special features. Some key differences in the implementation for LTL problems are the following: (i) the program uses floating point arithmetic; (ii) unlike the previous implementation, the LTL implementation executes only one dual ascent cycle, i.e., it does not perform the restricted ascent; (iii) the program does not incorporate the enhancement of Section 3.1 to handle the stronger forcing constraint (1.7); (iv) due to storage limitations, the program does not use a logical array to identify the tight cutset arcs belonging to the set A'(k); instead, A'(k) is recomputed at every iteration; and (v) local improvement is performed using only a drop procedure, instead of a drop-add procedure.

Interpretation of Computational Results

Table IV presents our computational results for the 7 LTL test problems. As before, the CPU execution times correspond to a FORTRAN implementation, compiled with the level 2 optimization option, on the IBM 3083. The computational results demonstrate that, for LTL problems, both the proportion of transshipment nodes and the demand level (or, equivalently, the fixed-to-variable cost ratio) have a strong influence on the number of dual ascent iterations and the quality of the dual-based bounds. For example, problem LSP2 has both the highest proportion of transshipment nodes and the highest percentage gap (4.57 %) among all the medium flow problems. Similarly, problem LSP5 has a relatively large gap (6.32 %) because of its exceptionally high fixed-to-variable cost ratio (114.43). The % gaps range from 0.32 to 1.23 for the remaining 5 out of the 7 problems.

The CPU time, on the other hand, depends more on the problem size than on the proportion of transshipment nodes. A comparison of Table IV with the dual ascent results for general, undirected problems of comparable size (Table II) suggests that the LTL-consolidation problems, because of their special structure, are easier to solve (though in 2 instances, the percentage gaps are larger). In particular, the CPU time required per ascent cycle is significantly smaller for the LTL problems. Another interesting difference between these two classes of problems is the much larger proportion of computation time required for the heuristic procedure in the LTL case. On analyzing the detailed outputs, we found that many more arcs had to be dropped from the initial LTL solution before reaching a local optimum. The fact that only one particular commodity can flow on each direct origin-to-destination arc possibly contributes to this behavior.

Table IV
Dual Ascent Results for LTL Consolidation Problems

Problem	(UBD-LBD)/LBD%	CPU secs on IBM 3083	
		Total*	Ascent %*
LSP1	0.67	0.27	59.3
LSP2	4.57	0.75	53.3
LSP 3	0.32	27.80	16.7
LSP 4	0.68	94.69	8.6
LSP 5	6.32	132.25	13.4
LSP 6	1.23	92.41	8.3
LSP 7	0.35	43.80	13.7

* Includes CPU time for input and initialization.

5. Conclusions

The results in this paper show that a combined dual ascent method and drop-add heuristic is capable of effectively solving large-scale uncapacitated network design problems. This approach solves large-scale problems (up to 595 design arcs in our computational experiments) quickly and consistently generates solutions guaranteed to be within 1 to 3 % of optimality. The application of the methodology to realistic trucking data also shows that it should be capable of solving large-scale problems met in practice. The most significant limitation to the method at present is not computation time, but rather storage requirements.

The effectiveness of the dual ascent method provides empirical confirmation that, for the disaggregate formulation (1.1) - (1.4), linear programming provides a good approximation to the integer programming polyhedron for the network design model. This experience adds to the growing evidence that for linear programming relaxations of integer programs "larger is better"; that is, when they can be solved, the relaxations are most effective when the linear programs are written in the most disaggregate form, with many constraints and variables. For the network design problem, this experience would be even more compelling if it were complemented by theoretical evidence (possibly probabilistic, average case analysis) that demonstrates the method's effectiveness. The work of Prodon, Liebling, and Groflin on the Steiner tree problem provides some insight in this direction. It seems, though, that much remains to be done.

Another major avenue for future research, that would considerably enhance the range of applications of the network design model, would be the addition of multiple choice and precedence constraints on the design variables and the introduction of arc capacities. Unfortunately, our preliminary experience suggests that a straightforward generalization of our linear programming-based dual-ascent methodology is not effective in solving capacitated problems. The linear programming relaxation no longer provides a good approximation to the integer program - the gap between the objective values of the linear and integer programs seems too large. This experience suggests the need for better linear programming formulations for the capacitated problems and the likely need for a better understanding of the structure (most particularly, the facial structure) of the integer programming polyhedron. In some complementary work, we have made some first steps toward meeting this objective. Balakrishnan (1985) and Balakrishnan and Magnanti (1987) have developed several results concerning the polyhedral structure of uncapacitated network design problems, as formulated with path instead of arc flows. Barany, Van Roy and Wolsey (1984a and 1984b) and Pochet and Wolsey (1986) have studied the polyhedral structure of the uncapacitated economic lot-sizing problem (this problem can be viewed as a special network design problem). Lemke and Wong (1987) described several facets of the k-median facility location problem and Leung and Magnanti (1986) have introduced new facets for the capacitated facility location problem. Leung, Magnanti and Vachani (1987) and Pochet (1986) have described facets for the capacitated economic lot-sizing problem and Magnanti and Vachani (1987) have discussed facets for production planning problems with changeover costs. Several of these contributions also contain encouraging computational experience. Since each of these problems can be viewed as special cases of the capacitated network design problem, their analysis sheds some insight on the polyhedral structure of the general version of this problem. Again, however, much remains to be done.

Acknowledgments: We would like to thank Professor Bruce W. Lamar for providing us the data for the Less-than-Truckload Consolidation test problems.

Appendix 1

Relating the Dual Ascent Algorithms for the Steiner Tree and Network Design Problems

The Steiner tree problem on a directed graph can be stated as follows: find a minimum cost set of directed arcs that connects a designated root node to a designated subset T of nodes. As indicated by Magnanti and Wong (1984), this problem can be modeled as a single source network design problem with |T| commodities, one for each node in T, and for every commodity t, the root node is the origin and node t is the destination. The fixed cost of an arc is equal to its cost in the Steiner problem and all flow routing costs are zero (i.e., all $c_{ij}^k = 0$). Then an optimal network design solution will also be an optimal Steiner problem solution.

Wong's (1984) dual ascent procedure for Steiner trees operates in the same manner as the dual ascent labeling method applied to the equivalent network design problem except that the two procedures process the commodities in a different order. (In this Appendix, we assume familiarity with Wong's algorithm.)

Recall that the dual ascent labeling method processes the commodities in a fixed order. On the other hand, the Steiner ascent method has a special priority scheme for processing the commodities: it processes commodities at certain times depending on the characteristics of the current dual solution. We will now show that the two ascent procedures are, in fact, equivalent in the following way. If the dual ascent labeling method processes a commodity that would not have been processed with the Steiner ascent method, the labeling method will not change the dual solution during that iteration. Thus, the commodities ignored in the course of the Steiner ascent method will also be effectively ignored by the labeling method. Since the two ascent procedures are otherwise identical, we will have shown that the dual ascent labeling method applied to the equivalent network design problem for Steiner trees is equivalent to the Steiner tree dual ascent method.

Wong's Steiner tree dual ascent procedure constructs an auxiliary directed graph G' with the same node set as the Steiner network and arc (i,j) is in the graph only if $s_{ij} = 0$. We shall say that a destination node $t_1 \varepsilon T$ belongs to a *root component* if whenever G' contains a path from any node $t_2 \varepsilon T$ to t_1, G' also contains a path from t_1 to t_2. (Thus if, for example, all arcs have positive slack then each node of T comprises a root component.) The Steiner ascent procedure selects a commodity k and increases $v^k_{D(k)}$ only if node D(k) belongs to a root component in G'. We now show that the dual ascent labeling method simulates this behavior when applied to the equivalent network design model of the Steiner tree problem.

We first note that when the dual ascent labeling method is applied to the network design model of Steiner tree problems, at each iteration, every arc (i,j) contained in the directed cutset corresponding to the node partition $N_1(k)$ and $N_2(k)$ is tight i.e., if $i \varepsilon N_1(k)$ and $j \varepsilon N_2(k)$, then $(i,j) \varepsilon A'(k)$. To establish this property observe that, since all $c^k_{ij} = 0$, and all w variables are initialized to zero, the initial values of all the v variables are zero. Therefore, originally

$$c^k_{ij} + w^k_{ij} - (v^k_j - v^k_i) = 0, \qquad \text{for all arcs } (i,j) \varepsilon A,$$

that is, all arcs are tight initially. Remark 1 following the statement of the algorithm shows that all the arcs in the directed cutset A(k) remain tight during each iteration.

This property implies that the minimization operator defining δ_2 in Step 1 never has any candidates. So δ will always be defined by δ_1 and each execution of Step 1 will label a new node and therefore increase $N_2(k)$ by one element. Throughout the following discussions we assume, for simplicity, that $\delta_1 = s_{ij}$ for a unique arc (i,j) in Step 1(a) of the labeling method (that is, the degeneracy described in Remark 4 following the statement of the method in Section 2.3 does not arise).

The dual ascent procedure executes Step 1 for every commodity $k \varepsilon K$ with $O(k) \varepsilon N_1(k)$. We will now show that if we are executing Step 1 with a commodity k, whose destination node D(k) does not belong to a root component, then $\delta = 0$ and the dual ascent procedure will not change the dual solution.

First we need to establish the following property.

Property (Labeling Property for Root Components)

Suppose we are executing Step 1 with commodity k whose destination D(k) does not belong to a root component of G'. Then, for some $l \in K$, there must a path of arcs with $s_{ij} = 0$ from $D(l)$ to D(k) but not from D(k) to $D(l)$. Also for at least one $h \in K$, $D(h) \in N_2(l)$ and $D(h) \notin N_2(k)$.

Proof:

The first part of this property is a restatement of the assumption that D(k) does not belong to a root component of G'. We will prove the second part by contradiction. Assume that no $h \in K$ satisfies $D(h) \in N_2(l)$ and $D(h) \notin N_2(k)$, and therefore $N_2(l)$ is a subset of $N_2(k)$. Then since no path of arcs with $s_{ij} = 0$ connects D(k) to $D(l)$, we must have $D(k) \notin N_2(l)$ and therefore $|N_2(k)| \geq |N_2(l)| + 1$.

The current execution of Step 1 with commodity k will increase $N_2(k)$ by one element. If commodity l is still in the set CANDIDATES then $N_2(l)$ will also increase by one element by the end of the entire execution of Step 1 for all candidates. Otherwise if commodity l is not in the set CANDIDATES, then $N_2(l)$ will not be increased during the current execution of Step 1. Therefore at the end of the entire execution of Step 1 we will still have $|N_2(k)| \geq |N_2(l)| + 1$.

However, each complete execution of Step 1 increases $N_2(j)$ by one element for every commodity j that satisfies the condition $O(j) \in N_1(j)$. Since $|N_2(k)| \geq |N_2(l)| + 1$ at the end of the current execution of Step 1, some previous execution of Step 1 must have processed commodity k but not commodity l (that is, during that iteration $O(l) \in N_2(l)$). Then the current execution of Step 1 with commodity k will also have $O(l) \in N_2(l)$. For the Steiner problem we have $O(l) = O(k)$, and since we are assuming that $N_2(l)$ is a subset of $N_2(k)$ we have $O(k) \in N_2(k)$. But we are executing Step 1 with commodity k which implies that $O(k) \in N_1(k)$ and contradicts the previous statement. So there must be at least one $h \in K$ such that $D(h) \in N_2(l)$ and $D(h) \notin N_2(k)$. ■

The labeling property for root components implies that δ must be zero during the current execution of Step 1 with commodity k for the following reason. We have assumed that $D(l) \varepsilon N_2(k)$ (since D(k) does not belong to a root component) which implies that there is a path of arcs with $s_{ij} = 0$ from D(*l*) to D(k). The labeling property implies that for some commodity h, $D(h) \varepsilon N_2(l)$ and the network must also contain a path of arcs with $s_{ij} = 0$ from D(h) to D(*l*). Hence there is a path of arcs with $s_{ij} = 0$ from D(h) to D(k). Since $D(h) \not\varepsilon N_2(k)$, at least one arc of this zero-slack path from D(h) to D(k) must belong to the directed cutset A(k). Hence, δ must be zero for the current execution of Step 1 with commodity k.

This argument shows that the Steiner tree ascent method priority scheme is, in effect, simulated by the labeling method (since the labeling method will not change the dual solution if it is processing a commodity that would not have been selected by the Steiner tree method). Thus, we have shown that the dual ascent labeling method for the network design problem is essentially a generalization of the Steiner tree dual ascent method.

References

BALAKRISHNAN, A. 1984. Valid Inequalities and Algorithms for the Network Design Problem with an Application to LTL Consolidation. Unpublished dissertation, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Massachusetts.

BALAKRISHNAN, A. 1985. LP Extreme Points and Cuts for the Fixed-Charge Network Design Problem. Working Paper, Krannert Graduate School of Management, Purdue University, West Lafayette, Indiana.

BALAKRISHNAN, A., AND T. L. MAGNANTI. 1987. Set Packing-Based Inequalities for the Fixed-Charge Network Design Problem. In preparation.

BARANY, I., T. J. VAN ROY AND L. A. WOLSEY. 1984a. Uncapacitated Lot Sizing: The Convex Hull of Solutions. *Math. Prog. Study* 22, 32-43.

BARANY, I., T. J. VAN ROY AND L. A. WOLSEY. 1984b. Strong Formulations for Multi-item Capacitated Lot Sizing. *Man. Sci.* 30, 1255-1261.

BEALE, E. M. L., AND J. A. TOMLIN. 1972. An Integer Programming Approach to a Class of Combinatorial Problems. *Math. Prog.* 3, 339-344.

BILDE, O., AND J. KRARUP. 1977. Sharp Lower Bounds and Efficient Algorithms for the Simple Plant Location Problem. *Annals of Disc. Math.* 1, 79-97.

BILLHEIMER, J., AND P. GRAY. 1973. Network Design with Fixed and Variable Cost Elements. *Trans. Sci.* 7, 49-74.

BOFFEY, T. B., AND A. I. HINXMAN. 1979. Solving for Optimal Network Problem. *Eur. J. Opnl. Res.* 3, 386-393.

BOYCE, D. E., A. FARHI AND R. WEISCHEDEL. 1973. Optimal Network Problem: A Branch-and-Bound Algorithm. *Environ. Plan.* 5, 519-533.

CHU, Y. J., AND T. H. LIU. 1965. On the Shortest Arborescences of a Directed Graph. *Scientia Sinica* 14, 1396-1400.

CORNUEJOLS, G., M. L. FISHER AND G. L. NEMHAUSER. 1977. Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms. *Mgmt. Sci.* 23, 789-810.

DAVIS, P. S., AND T. L. RAY. 1969. A Branch-and-Bound Algorithm for Capacitated Facilities Location Problems. *Nav. Res. Log. Quart.* 16, 331-344.

DIJKSTRA, E. W. 1959. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik* 1, 269-271.

DIONNE, R., AND M. FLORIAN. 1979. Exact and Approximate Algorithms for Optimal Network Design. *Networks* 9, 37-59.

EDMONDS, J. 1967. Optimum Branchings. *J. Res. Nat. Bur. Stan. - B. Math. and Math. Phy.* 71B, 233-240.

ELLMAN, J. 1983. Implementation of Solution Procedures to the General Network Design Problem. Computer Science Masters project, Rennselaer Polytechnic Institute, Troy, New York.

ERLENKOTTER, D. 1978. A Dual Based Procedure for Uncapacitated Facility Location. *Opns. Res.* 26, 992-1009.

FISHER, M. L., R. JAIKUMAR AND L. VAN WASSENHOVE. 1986. A Multiplier Adjustment Method for the Generalized Assignment Problem. *Man. Sci.* 32, 1095-1103.

FISHER, M. L., AND P. KEDIA. 1986. A Dual Algorithm for Large Scale Set Partitioning. Working Paper No. 894, Krannert Graduate School of Management, Purdue University, W. Lafayette, Indiana.

GALLO, G. 1981. A New Branch-and-Bound Algorithm for the Network Design Problem. Report L81-1, Instituto Di Elaborazione Dell' Informazione, Pisa, Italy.

GEOFFRION, A. M. 1974. Lagrangian Relaxation for Integer Programming. *Math. Prog. Study* 2, 82-114.

GEOFFRION, A. M., AND G. GRAVES. 1974. Multicommodity Distribution System Design by Benders Decomposition. *Mgmt. Sci.* 5, 822-844.

HOANG, H. H. 1973. A Computational Approach to the Selection of an Optimal Network. *Man. Sci.* 19, 488-498.

JOHNSON, D. S., J. K. LENSTRA AND A. H. G. RINNOOY KAN. 1978. The Complexity of the Network Design Problem. *Networks* 8, 279-285.

KEDIA, P., AND M. L. FISHER. 1986. Optimal Solution of Set Covering Problems Using Dual Heuristics. Working Paper No. 901, Krannert Graduate School of Management, W. Lafayette, Indiana.

KNUTH, D. 1969. *The Art of Computer Programming: Volume 2, Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts.

LAMAR, B. W., Y. SHEFFI AND W. B. POWELL. 1984. Bounding Procedures for Fixed Charge, Multicommodity Network Design Problems. Working Paper, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts.

LAMAR, B. W., Y. SHEFFI AND W. B. POWELL. 1986. A Lower Bound for Uncapacitated, Multicommodity Fixed Charge Network Design Problems. Working Paper, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts.

LEMKE, P., AND R. T. WONG. 1987. Characterization and Description of Facets for the K-Median Problem. In preparation.

LEUNG, J., AND T. L. MAGNANTI. 1986. Valid Inequalities and Facets of the Capacitated Plant Location Problem. Working Paper OR 149-86, Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts (to appear in *Math. Prog.*).

LEUNG, J., T. L. MAGNANTI AND R. VACHANI. 1987. Facets and Algorithms for Capacitated Lot Sizing. Working Paper, Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts.

LOS, M., AND C. LARDINOIS. 1980. Combinatorial Programming, Statistical Optimization and the Optimal Transportation Network Problem. *Trans. Res.* 16B, 89-124.

MAGNANTI, T. L., AND B. W. GOLDEN. 1978. Transportation Planning: Network Models and Their Implementation. In A. C. Hax (ed.). *Studies in Operations Management*. North-Holland, Amsterdam.

MAGNANTI, T. L., P. MIREAULT AND R. T. WONG. 1986. Tailoring Benders Decomposition for Network Design. *Math. Prog. Study* 26, 112-154.

MAGNANTI, T. L., AND R. VACHANI. 1987. A Strong Cutting Plane Approach to Production Planning with Changeover Costs. In preparation.

MAGNANTI, T. L., AND R. T. WONG. 1981. Accelerating Benders Decomposition: Algorithmic Enhancement and Model Selection Criteria. *Opns. Res.* 29, 464-484.

MAGNANTI, T. L., AND R. T. WONG. 1984. Network Design and Transportation Planning: Models and Algorithms. *Trans. Sci.* 18, 1-55.

MAIRS, T. G., G. W. WAKEFIELD, E. L. JOHNSON AND K. SPIELBERG. 1978. On a Production Allocation and Distribution Problem. *Man. Sci.* 24, 1622-1630.

POCHET, Y. 1986. Valid Inequalities and Separation for Capacitated Economic Lot Sizing. Discussion Paper, Center for Operations Research and Econometrics, Universite' Catholique de Louvain, Belgium.

POCHET, Y., AND L. A. WOLSEY. 1986. Lot-Size Models with Backlogging: Strong Relaxations and Cutting Planes. Discussion Paper, Center for Operations Research and Econometrics, Universite' Catholique de Louvain, Belgium.

PRODON, A., T. LIEBLING AND H. GROFLIN. 1985. Steiner's Problem on Two-Trees. Working Paper, Departement de Mathematiques, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland.

RARDIN, R. L. 1982. Tight Relaxations of Fixed Charge Network Flow Problems. Technical Report, No. J-82-3, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta.

RARDIN, R. L., AND U. CHOE. 1979. Tighter Relaxations of Fixed Charge Network Flow Problems. Technical Report, No. J-79-18, School of

Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta.

VAN ROY, T. J., AND E. ERLENKOTTER. 1982. A Dual-Based Procedure for Dynamic Facility Location. *Man. Sci.* 28, 1091-1105.

WILLIAMS, H. P. 1974. Experiments in the Formulation of Integer Programming Problems. *Math. Prog. Study* 2, 180-197.

WONG, R. T. 1980. Worst-Case Analysis of Network Design Problem Heuristics. *SIAM J. Alg. Disc. Meth.* 1, 51-63.

WONG, R. T. 1984. Dual Ascent Approach for Steiner Tree Problems on a Directed Graph. *Math. Prog.* 28, 271-287.

WONG, R. T. 1985. Probabilistic Analysis of an Optimal Network Problem Heuristic. *Networks* 15, 347-363.