A Branch-and-Cut Algorithm for the Single-Commodity, Uncapacitated, Fixed-Charge Network Flow Problem

Francisco Ortega
n-Side, Rue de la Longue Haie 17/001, 1348 Louvain-la-Neuve, Belgium

Laurence A. Wolsey
CORE and FSA, Université Catholique de Louvain, Voie du Roman Pays 34, 1348 Louvain-la-Neuve, Belgium

We present a branch-and-cut algorithm to solve the single-commodity, uncapacitated, fixed-charge network flow problem, which includes the Steiner tree problem, uncapacitated lot-sizing problems, and the fixed-charge transportation problem as special cases. The cuts used are simple *dicut* inequalities and their variants. A crucial problem when separating these inequalities is to find the right cut set on which to generate the inequalities. The prototype branch-and-cut system, `bc-nd`, includes a separation heuristic for the dicut inequalities and problem-specific primal heuristics, branching, and pruning rules. Computational results show that `bc-nd` is competitive compared to a variety of special purpose algorithms for problems with explicit flow costs. We also examine how general purpose MIP systems perform on such problems when provided with formulations that have been tightened *a priori* with dicut inequalities.

Keywords: network design; fixed charge; branch and cut; dicut inequalities; branching; heuristics; minimum-cost flow

1. INTRODUCTION

The single-commodity uncapacitated fixed-charge network flow problem (UFC) is one of a large class of network design problems. Specifically, given a digraph/network $D = (V, A)$, demands at the nodes, and fixed and variable costs on the arcs, the problem is to select a set of arcs to be opened and to find a feasible flow in the resulting network such that the sum of the fixed arc costs plus the variable flow costs is minimized.

Received March 2001; accepted January 2003
Correspondence to: L. A. Wolsey; email: wolsey@core.ucl.ac.be
This work was carried out at CORE as part of a doctoral dissertation at l'Université Catholique de Louvain

Until recently, the commercial mixed-integer programming solvers just used linear programming-based branch and bound and did not perform at all well on most instances of the UFC. In the last 4 years, these solvers have improved remarkably and now generate cutting planes such as flow-cover inequalities designed for simple fixed-charge flow problems. However they still do not take full advantage of the structure of the UFC.

This explains, in part, why much previous work has been on the development of specialized algorithms for specific variants of UFC. Chopra et al. [11] and Koch and Martin [27] developed branch-and-cut codes for the Steiner tree problem which can be viewed as a special case of the UFC in which the flow costs are zero. Several branch-and-bound algorithms have been described for the uncapacitated fixed-charge transportation problem in which the underlying network is bipartite (see, e.g., [5, 10, 34, 38]). Some production planning problems such as the single-level and (series) multilevel uncapacitated lot-sizing problem can be formulated as a UFC, but here, again, specialized algorithms have been developed (see, e.g., [7, 40]). Finally, for the single-source UFC, Hochbaum and Segev in [23] presented a Lagrangian relaxation algorithm and primal heuristics, and very recently Cruz et al. [14] reported results obtained with a branch-and-bound algorithm also based on a Lagrangian relaxation.

Another standard approach for single-source UFCs is a multicommodity reformulation. This was tested by Rardin and Choe [36] and used to obtain tight formulations for trees and Steiner trees [27, 29] and lot-sizing problems [35], and its projection into the original space of variables was analyzed in Rardin and Wolsey [37]. Although this approach leads to very tight linear programming formulations, the linear programs are known to be very hard to solve (see, for instance, [8]).

The original goal of this study was to test whether the effectiveness of cutting planes in solving uncapacitated lot-

sizing problems [4, 42] extended to the more general and more difficult UFC. In particular, we were interested in how the prototype branch-and-cut system `bc-opt` [13] (which, among others, generates path inequalities for fixed-charge path networks generalizing the lot-sizing inequalities) behaved on such problems. Another question was whether the UFC, lying somewhere between a general mixed-integer program and highly structured problems such as the fixed-charge transportation or Steiner problems, was at an appropriate level of generality for the study of cutting planes and the development of algorithms.

One result of our study was that, whereas cuts based on paths (as implemented in `bc-opt`) are fundamental for lot-sizing problems, simple "dicut" inequalities [2] and their variants are crucial for UFC and form the basis of the branch-and-cut system `bc-nd` implemented here. Our test set contains 31 "hard" instances that are not solved by either Cplex [24], `mp-opt` [16], or `bc-opt` within 30 minutes. `bc-nd` solves six of these instances to optimality, and the average final duality gap for the other 25 instances is 4.4%, while it is 30.40, 23.55, and 18.31% for the three systems just cited. Additionally, all the "medium" instances (those that are solved by exactly one of the general systems) are solved with `bc-nd`.

The outline of the article is as follows: In Section 2, we formulate the problem and give some definitions used throughout the paper. In Section 3, we describe the dicut inequality and its variants and consider the complexity of the separation problem. Section 4 is devoted to a description of `bc-nd`. The test instances are described in Section 5, and the computational results, in Section 6. The latter includes a brief report on the use of the multicommodity reformulation mentioned above for single-source UFCs. Although seven relatively small instances are solved just by linear programming, none of the linear programming relaxations of the other 13 instances tested was solved within 30 minutes.

Some conclusions and extensions are discussed in Section 7.

2. PROBLEM FORMULATION

The UFC can be formulated as follows: Given a directed graph $D = (V, A)$, a demand vector $b = (b_i)$ for $i \in V$, and fixed and variable costs f_{ij} and c_{ij} for $(i, j) \in A$, find a set of arcs and a feasible flow in the resulting network that minimizes the total cost. This problem is NP-hard, as it generalizes the Steiner tree problem [17].

To describe the UFC as a mixed-integer program, define x_{ij} to be the flow on arc (i, j) and $y_{ij} = 1$ if the arc (i, j) is used ($x_{ij} > 0$) and $y_{ij} = 0$ otherwise. A resulting formulation is

$$(UFC)\begin{cases} \min \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{(i,j)\in A} f_{ij}y_{ij} & (1)\\ s.t. & \\ \sum_{j\in V_i^-} x_{ji} - \sum_{j\in V_i^+} x_{ij} = b_i \ \forall\, i \in V & (2)\\ x_{ij} \le U y_{ij} \ \forall\, (i,j) \in A & (3)\\ x_{ij} \ge 0 \ \forall\, (i,j) \in A, \quad y_{ij} \in \{0,1\} \ \forall\, (i,j) \in A, & (4) \end{cases}$$

where $V_i^+ = \{j \in V : (i, j) \in A\}$ and $V_i^- = \{j \in V : (j, i) \in A\}$, U is a large positive integer, and $\Sigma_{i\in V} b_i = 0$. Constraints (2) are the well-known *conservation constraints*, and constraints (3) are the forcing constraints that guarantee that y_{ij} takes value one whenever x_{ij} is positive. Note that it suffices to take $U = \Sigma_{i\in V:b_i>0} b_i$.

As additional notation, we use $V_S = \{i \in V : b_i < 0\}$ to denote the set of *supply nodes*; $V_D = \{i \in V : b_i > 0\}$, the set of *demand nodes*; and $V_0 = \{i \in V : b_i = 0\}$, the set of *transshipment nodes*. Let X be the set of vectors (x, y) satisfying (2)–(4). The next proposition characterizes the extreme points of $conv(X)$ (see, e.g., Ahuja et al. [1]).

Proposition 2.1. *Given* (x, y) *in* X, *let*

$$F(x, y) = \{a \in A/0 < x_a < U,\ y_a = 1\}$$

$$L(x, y) = \{a \in A/x_a = 0,\ y_a \in \{0, 1\}\}$$

$$U(x, y) = \{a \in A/x_a = U,\ y_a = 1\}.$$

Then, (x, y) *is an extreme point of* $conv(X)$ *if and only if the graph* $D_{x,y} = (V, F(x, y))$ *contains no cycles.*

This characterization will be used in Section 4 to devise branching rules and a pruning criterion and also to fix variables in the enumeration tree.

3. VALID INEQUALITIES

In this section, we describe the dicut inequality and its variants. To present examples of the different inequalities, we use the instance shown in Figure 1.

Consider a proper subset S of nodes for which the net demand is positive, that is, $b(S) = \Sigma_{i\in S} b_i > 0$. The set X_S, obtained by summing up the conservation constraints (2) over all nodes in S, is known as a single-node flow set. Mathematically, it takes the form

$$X_S = \left\{ (x, y) \in \mathbb{R}^{|A|} \times \mathbb{R}^{|A|} : \begin{array}{ll} \sum_{(i,j)\in\delta^-(S)} x_{ij} - \sum_{(i,j)\in\delta^+(S)} x_{ij} = \sum_{i\in S} b_i & (5)\\ x_{ij} \le U y_{ij} \quad \text{for } (i,j) \in A & (6)\\ x_{ij} \ge 0, \quad y_{ij} \in \{0,1\} \quad \text{for } (i,j) \in A, & (7) \end{array} \right\}$$

$$
\begin{aligned}
x_{21} - x_{14} - x_{15} &= -1 \\
-x_{21} - x_{23} &= -3 \\
x_{23} + x_{43} - x_{37} &= 0 \\
x_{14} + x_{54} + x_{74} - x_{43} - x_{46} &= 2 \\
x_{15} - x_{54} - x_{56} &= -1 \\
x_{56} + x_{46} - x_{67} &= 0 \\
x_{37} + x_{67} - x_{74} &= 3
\end{aligned}
$$

$$x_{ij} \leq 5\, y_{ij}$$

$$x_{ij} \geq 0$$

$$y_{ij} \in \{0, 1\}$$

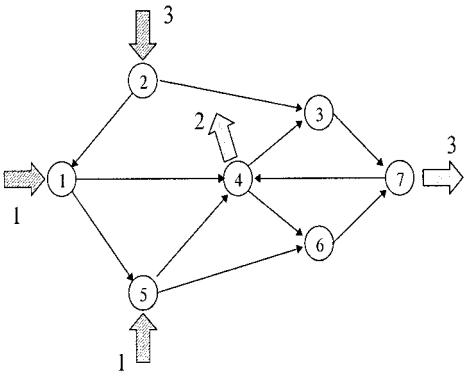

FIG. 1. Fixed-charge network flow example.

where $\delta^+(S) = \{(i, j) \in A : i \in S, j \notin S\}$ is the set of arcs leaving S, and $\delta^-(S) = \{(i, j) \in A : i \notin S, j \in S\}$ is the set of arcs entering S.

Taking $S = \{3, 5, 6, 7\}$ in the example, constraints (5) and (6) take the form

$$x_{23} + x_{43} + x_{46} + x_{15} - x_{54} - x_{74} = 2$$

$$x_{23} \leq 5y_{23}, \quad x_{43} \leq 5y_{43}, \quad x_{46} \leq 5y_{46}, \quad x_{15} \leq 5y_{15}, \quad x_{54} \leq 5y_{54}, \quad x_{74} \leq 5y_{74}.$$

Because S has a positive demand of 2 units, every feasible flow must contain at least 2 units entering S. Thus, at least one arc of $\delta^-(S) = \{(2, 3), (4, 3), (4, 6), (1, 5)\}$, the set of arcs entering S, must be open. This is expressed in the inequality

$$y_{23} + y_{43} + y_{46} + y_{15} \geq 1.$$

We now make a more general statement:

Proposition 3.1. *For $S \subset V$ with $\Sigma_{i \in S}\, b_i > 0$, the* basic dicut inequality

$$\sum_{(i,j) \in \delta^-(S)} y_{ij} \geq 1$$

is valid for X.

These inequalities have been used to formulate the directed Steiner tree problem [29].

Consider, again, the same subset $S = \{3, 5, 6, 7\}$. A relaxation of the single-node flow set X_S is obtained if x_{23} and x_{15} are replaced by their upper bounds and x_{54} and x_{74} are replaced by their lower bounds of zero. The resulting set is

$$5y_{23} + x_{43} + x_{46} + 5y_{15} \geq 2,$$
$$y_{23}, y_{15} \in \{0, 1\}, x_{43}, x_{46} \geq 0.$$

Applying coefficient reduction [31] or the mixed-integer rounding (MIR) procedure [33] gives the inequality

$$2y_{23} + x_{43} + x_{46} + 2y_{15} \geq 2.$$

In general, we have the following result:

Proposition 3.2. *For $S \subset V$ with $\Sigma_{i \in S}\, b_i > 0$, the* mixed dicut inequality

$$\sum_{(i,j) \in \delta^-(S) \setminus C} x_{ij} + \sum_{(i,j) \in C} b(S)\, y_{ij} \geq b(S)$$

is valid for X for all $C \subseteq \delta^-(S)$.

Further, taking $S = \{4, 5, 6, 7\}$, one such mixed dicut inequality is

$$x_{14} + 4y_{37} + x_{15} \geq 4.$$

Now, the flow going from $V\backslash S = \{1, 2, 3\}$ to satisfy demand in S passing through the arc (3, 7) is at most 3 units, the supply of node 2. This observation allows us to tighten the coefficient associated with y_{37} giving the valid inequality:

$$x_{14} + 3y_{37} + x_{15} \geq 4.$$

In general, for $e \in C \subseteq \delta^-(S)$, let $E(S) = \{(i, j) \in A : i, j \in S\}$, $V^+_{e,S} = \{i \in S : b_i > 0$ and there exists a dipath in $G_S = (S, E(S))$ between the head node of e and the node $i\}$, $V^-_{e,S} = \{i \in V\backslash S : b_i < 0$ and there exists a dipath in $G_{V\backslash S} = (V\backslash S, E(V\backslash S))$ between the node i and the tail node of $e\}$ and $\alpha_e(S) = \min\{\sum_{i \in V^+_{e,S}} b_i, \sum_{i \in V^-_{e,S}} |b_i|\}$.

Proposition 3.3. *For $S \subset V$ with $\sum_{i \in S} b_i > 0$, the* simple inflow–outflow inequality

$$\sum_{(i,j) \in \delta^-(S)\backslash C} x_{ij} + \sum_{(i,j) \in C} \alpha_{ij}(S)\, y_{ij} \geq b(S)$$

is valid for X for all $C \subset \delta^-(S)$.

This modification is important when the underlying graph is sparse. For the single-item uncapacitated lot-sizing problem, all the inequalities required to give a complete description of the convex hull are of this type.

With $S = \{3, 4, 6, 7\}$, we obtain the following mixed dicut inequality:

$$x_{23} + x_{14} + 5y_{54} + 5y_{56} \geq 5.$$

Now suppose that the contribution of the flow in arc (7, 4) to satisfy the demand $b(S)$ is measured separately. In this case, the maximum flow that can pass through arc (5, 6) using the arcs of $E(S)$ other than arc (7, 4) in order to satisfy the demand $b(S)$ is 3 units, the demand of node 7. So, the inequality

$$x_{23} + x_{14} + 5y_{54} + 3y_{56} + x_{74} \geq 5$$

is valid.

In general, given $R \subset E(S)$, define $V^R_{e,S} = \{i \in S : b_i > 0$ and there exists a dipath in $G_S = (S, E(S)\backslash R)$ between the head node of e and the node $i\}$ and $\alpha^R_e(S) = \sum_{i \in V^R_{e,S}} b_i$.

Proposition 3.4. *For $S \subset V$ with $\sum_{i \in S} b_i > 0$, the* inflow–outflow inequality

$$\sum_{(i,j) \in (\delta^-(S)\backslash C) \cup R} x_{ij} + \sum_{(i,j) \in C} \alpha^R_{ij}(S)\, y_{ij} \geq b(S)$$

is valid for X for all $C \subset \delta^-(S)$, $R \subseteq E(S)$.

The above inequality is a particular case of the network inequalities of van Roy and Wolsey [41].

With $S = \{3, 5, 6, 7\}$, another possible relaxation of X_S is given by

$$5y_{23} + 5y_{43} + x_{46} + 5y_{15} \geq 2 + x_{74}, \quad x_{74} \leq 5y_{74}.$$

Letting, $\bar{x}_{74} = 5y_{74} - x_{74} \geq 0$ and $\bar{y}_{74} = 1 - y_{74}$, we get

$$5y_{23} + 5y_{43} + x_{46} + 5y_{15} + \bar{x}_{74} + 5\bar{y}_{74} \geq 7.$$

Now applying the MIR procedure, we obtain

$$2y_{23} + 2y_{43} + x_{46} + 2y_{15} + \bar{x}_{74} + 2\bar{y}_{74} \geq 4,$$

and reintroducing the original variables, we get the following valid inequality:

$$2y_{23} + 2y_{43} + x_{46} + 2y_{15} \geq 2 + (x_{74} - 3y_{74}).$$

The general expression is given in the next proposition:

Proposition 3.5. *For $S \subset V$ with $\sum_{i \in S} b_i > 0$, the* mixed dicut with outflow inequality

$$\sum_{(i,j) \in \delta^-(S)\backslash C^-} x_{ij} + \sum_{(i,j) \in C^-} b(S)\, y_{ij} \geq b(S) + \sum_{(i,j) \in C^+} \{x_{ij} - r(S)\, y_{ij}\}$$

is valid for X for all $C^- \subseteq \delta^-(S)$ and $C^+ \subseteq \delta^+(S)$, with $r(S) = U - b(S)$.

Proof: The proof is a direct generalization of the procedure used in the example. Consider the following relaxation of X_S:

$$\sum_{(i,j) \in \delta^-(S)\backslash C^-} x_{ij} + U \sum_{(i,j) \in C^-} y_{ij} \geq b(S) + \sum_{(i,j) \in C^+} x_{ij}$$

plus the constraints $x_{ij} \leq U y_{ij}$ for all $(i, j) \in \delta^-(S) \cup C^+$ and all y_{ij} binary. Defining the variables $\bar{x}_{ij} = U y_{ij} - x_{ij}$, and $\bar{y}_{ij} = 1 - y_{ij}$, we can make the substitution $x_{ij} = U - U\bar{y}_{ij} - \bar{x}_{ij}$ for $(i, j) \in C^+$. Now the previous constraint can be rewritten as

$$\sum_{(i,j) \in \delta^-(S)\backslash C^-} x_{ij} + U \sum_{(i,j) \in C^-} y_{ij} + \sum_{(i,j) \in C^+} \bar{x}_{ij} + U \sum_{(i,j) \in C^+} \bar{y}_{ij} \geq b(S) + U|C^+|.$$

Applying the MIR procedure, we get

$$\sum_{(i,j)\in\delta^-(S)\setminus C^-} x_{ij} + \sum_{(i,j)\in C^+} \bar{x}_{ij} \geq b(S)(1 + |C^+| - \sum_{(i,j)\in C^-} y_{ij} - \sum_{(i,j)\in C^+} \bar{y}_{ij}),$$

which, after substitution, back gives the required inequality. ∎

At least two other classes of inequalities can potentially be used in solving UFC. First, it is possible to mix dicut inequalities for different sets S using the mixing procedure of Günlük and Pochet [21]. A second class are the *multi-dicut* inequalities. They were initially presented by Rardin and Wolsey in [37] for the single-source case. A recent version for the multiple-source case can be found in [30].

Before finally leaving the example in Figure 1, it can be checked that all but one of the valid inequalities presented are facet-defining. The complete description of $conv(X)$, obtained with the code Porta [12], contains 7143 facet defining inequalities. Although only a small percentage of these are dicut inequalities, our results below suggest that they play an important role in closing the duality gap.

3.1. Difficulty of the Separation Problem

First, we formalize the separation problem for simple dicut inequalities. To find a violated simple dicut inequality, we look for a subset of nodes S such that $b(S) > 0$ and

$$\sum_{(i,j)\in\delta^-(S)} \bar{y}_{ij} < 1.$$

This can be seen as a minimum-cut problem with an additional constraint to ensure that $b(S) > 0$. For $i \in V$, define variable $z_i = 1$ if i belongs to S, and $z_i = 0$ otherwise. The separation problem reduces to solving the problem

$$\xi = \min\left\{ \sum_{(i,j)\in A} \bar{y}_{ij} z_j (1 - z_i) : \sum_{i\in V} b_i z_i > 0,\ z_i \in \{0,1\} \quad \text{for all } i \in V \right\}.$$

If $\xi < 1$, the set S defined by $\{i \in V : z_i = 1\}$ leads to a violated inequality.

By reduction from the exact partitioning problem, the separation problem associated with the simple dicut inequalities can be shown to be NP-complete [18]. However, for the single-source problem in which $|V_S| = 1$, the imposed constraint can be dropped. The separation problem can then be solved in polynomial time. It can be reduced to $|V_D|$ minimum $s - t$ cut problems, where s is the source and t varies over the set V_D.

The problem of finding a violated mixed dicut inequality can be stated as follows: Given a fractional point $(\bar{x}, \bar{y})$, we look for $S \subset V$ with $b(S) > 0$ and $C \subseteq \delta^-(S)$ such that

$$\sum_{(i,j)\in\delta^-(S)\setminus C} \bar{x}_{ij} + \sum_{(i,j)\in C} b(S)\bar{y}_{ij} < b(S).$$

For a given S, finding the most violated inequality is trivial. It suffices to set $C = \{i,j \in \delta^-(S) : \bar{x}_{ij} > b(S)\bar{y}_{ij}\}$. Therefore, the principal difficulty regarding the separation of dicut inequalities is to find the right set S.

For mixed dicut inequalities, the complexity of the separation problem is apparently still open (see [2]).

The above observations led us to consider using heuristics to separate the various dicut inequalities. The separation heuristic, based on searching for good candidate cut sets, is presented in the next section.

4. BC-ND: A BRANCH-AND-CUT SYSTEM FOR UFC

In this section, we describe the branch-and-cut system `bc-nd`. We begin with some basic implementation issues; then, we present the separation heuristic, the primal heuristic, and branching rules and, finally, pruning and variable fixing criteria.

4.1. The Basics

Our implementation is based on the Extended Modeling and Optimisation Subroutine Library (EMOSL) of Xpress [15]. This library implements a branch-and-bound algorithm with a series of "entry points" that allow users to include their own routines. Using these entry points, we can generate cuts and add them to the matrix, apply heuristics, and develop branching rules. At the top node, we used these entry points to generate cuts and to apply a heuristic. In the enumeration tree, we used the entry points to generate cuts, prune a node, choose branching variables, and implement a primal heuristic. Additionally, the library has routines to access the information contained in the model file. Such information is used to determine the digraph that defines the instance being solved. The data structure used to store the graph was borrowed from MCF [28].

Because several cuts can be generated from the same set S, we set up a set pool to store the candidate sets. The set pool is a dynamic doubly linked list. Sets are *active* until an associated *frequency parameter* falls below a certain value, at which point the set becomes inactive. How this parameter is updated and when a set is declared inactive are described later in the cut generation step.

4.2. Separation

The separation consists of the following steps: cut deletion, shrinking, set generation, cut generation, and reoptimization. One realization of all these steps is called a `pass`.

The default number of passes at the top node has been fixed at 30, whereas in the enumeration tree it has been fixed at 5. We now describe each step of a pass:

- *Cut deletion*. Because the number of violated dicuts can be large, keeping all of them in the matrix during all the passes can be too expensive. Therefore, we eliminate the nonbinding cuts from the matrix at the beginning of each pass. Because some of the deleted cuts may be violated later, we perform cut pool separation at the beginning of the cut generation step. No cuts are deleted from the cut pool.
- *Shrinking*. To reduce the size of the graph on which we search for "interesting" subsets, the graph is shrunk based on the current linear programming solution. Specifically, whenever $\bar{y}_{ij} > 0.99$ and $\bar{x}_{ij} > 10^{-6}$, the two end nodes i, j are contracted into one supernode. The demand of the new supernode is the sum of the demands. Only nodes are contracted, so the resulting reduced graph typically contains multiple arcs and loops. This shrinking procedure is heuristic. Arcs with $\bar{y}_{ij} > 0$ but $\bar{x}_{ij} = 0$ are not used for shrinking because the addition of the dicut inequalities often forces $y_{ij} > 0$ artificially in the linear programming relaxation, even when there is no flow in the corresponding arc.
- *Subset generation.* The dicut inequalities are based on "node subsets". Therefore, finding good subsets can reduce the number of iterations of the cut generation and also lead to a better top node reformulation. In our implementation, three greedy procedures are used to generate subsets for a given fractional solution $(\bar{x}, \bar{y})$. They differ in the choice of the initial node and in the quantity used to enlarge the current set. Below, we describe these choices:

  — Initialize $S = \{i_0\}$ for $i_0 \in V_S \cup V_D$. Enlarge S, using $\arg\max_i\{\bar{y}_{ij} : (i, j) \in \delta^-(S), \bar{y}_{ij} \in (0, 1)\}$.
  — Initialize $S = \{i_0\}$ for $i_0 \in V_S \cup V_D$. Enlarge S, using $\arg\max_i\{\bar{x}_{ij} - b(S)\bar{y}_{ij} : (i, j) \in \delta^-(S), \bar{y}_{ij} \in (0, 1)\}$.
  — Given an arc $a = (i_0, j_0)$ such that $\bar{y}_{i_0 j_0}$ is fractional, two candidate sets S are built. In the first case, we start with a set S such that $j_0 \in S$, $b(S) > 0$ and $a \in \delta^-(S)$, and we expand S using the criterion $\max_i\{\bar{x}_{ij} - b(S)\bar{y}_{ij} : (i, j) \in \delta^-(S), \bar{y}_{ij} \in (0, 1), b(S) + b_i > 0\}$. In the second case, we start with a set $\tilde{S} = V\backslash S$ such that $i_0 \in V\backslash S$, $b(V\backslash S) < 0$, and $a \in \delta^+(V\backslash S)$ and we expand $V\backslash S$ using the criterion $\max_j\{\bar{x}_{ij} - |b(V\backslash S)|\bar{y}_{ij} : (i, j) \in \delta^+(V\backslash S), \bar{y}_{ij} \in (0, 1), b(V\backslash S) + b_j < 0\}$. The procedure stops either when the maximum number of nodes allowed is reached or when a violated inequality can be generated.

  The three procedures are called sequentially. The sets generated are stored in the set pool with the status "active" and the frequency parameter initialized at zero.
- *Cut generation*. We start by performing cut pool separation. Then, for each active set S, three dicut inequalities can be generated:

  (a) *Simple dicut*. If $\sum_{a\in\delta^-(S)} \bar{y}_a < 1 - 0.015$, then the inequality

  $$\sum_{a\in\delta^-(S)} y_a \geq 1$$

  is added to the cut pool.

  (b) *Simple inflow–outflow inequality*. Define $C = \{a \in \delta^-(S) : \bar{x}_a > \alpha_a(S)\bar{y}_a\}$. If $\sum_{a\in\delta^-(S)\backslash C} \bar{x}_a + \sum_{a\in C} \alpha_a(S)\bar{y}_a < b(S) - 0.015$, then the inequality

  $$\sum_{a\in\delta^-(S)\backslash C} x_a + \sum_{a\in C} \alpha_a(S) y_a \geq b(S)$$

  is added to the cut pool. The coefficient $\alpha_{ij}(S)$ is computed using a breadth-first search to determine the sets $V^+_{ij,S}$ and $V^-_{ij,S}$ defined in Proposition 3.3. Then, $\alpha_{ij}(S) = \min\{b(S), \sum_{k\in V^+_{ij,S}} b_k, \sum_{k\in V^-_{ij,S}} |b_k|\}$.

  (c) *Mixed dicut with outflow*. Define $C^- = \{a \in \delta^-(S) : \bar{x}_a > b(S)\bar{y}_a\}$, $r(S) = U - b(S)$ and $C^+ = \{a \in \delta^+(S) : \bar{x}_a > r(S)\bar{y}_a\}$. If $\sum_{a\in\delta^-(S)\backslash C^-} \bar{x}_a + \sum_{a\in C^-} b(S)\bar{y}_a < b(S) + \sum_{a\in C^+} \{\bar{x}_a - r(S)\bar{y}_a\} - 0.015$, then the inequality

  $$\sum_{a\in\delta^-(S)\backslash C^-} x_a + \sum_{a\in C^-} b(S) y_a \geq b(S) + \sum_{a\in C^+} \{x_a - r(S) y_a\}$$

  is added to the cut pool.

If a cut is added to the cut pool, the frequency parameter is increased by 1. Otherwise, it is decreased by 1, and the next set is inspected.

Whenever the frequency parameter of a set is less than -3, the set is declared inactive. Once, all the active sets have been visited, the cuts generated are added into the matrix.

The cut tolerance of 0.015 used above was selected from a small set of candidate values by running on a subset of the test instances.

- *Reoptimization*. If violated inequalities have been found, the linear program is reoptimized. If the number of passes is less than the maximum, go to the next pass. Otherwise, go to the enumeration phase.

4.3. Primal Heuristics

Given the structure of UFC, we examined the possibility of developing effective primal heuristics to find good feasible solutions rapidly. After several attempts, motivated by [14, 20, 22, 39], two heuristics were retained:

- *Slope scaling* [25]. This algorithm is based on the idea that there exists a linear program,

$$(P(\hat{c}))\min\left\{\sum_{(i,j)\in A}\hat{c}_{ij}x_{ij}:\sum_{j\in V_i^-}x_{ji}-\sum_{j\in V_i^+}x_{ij}=b_i\ \forall i\in V,\right.$$
$$\left.0\le x_{ij}\le U\ \forall (i,j)\in A\right\},$$

which has the same optimal solution as that of the original mixed-integer problem or, in other words, that there exists a cost vector $\hat{c}$ such that $v(UFC) = v(P(\hat{c}))$.

To find such a $\hat{c}$, a sequence $\{\bar{c}^k\}_{k=1}^K$ of slopes are constructed, such that $\bar{c}^K$ is not far from $\hat{c}$. Let x^k be an optimal solution of $P(\bar{c}^k)$. The slope at iteration $k + 1$ is computed as follows:

$$\bar{c}_{ij}^{k+1}=\begin{cases}c_{ij}+\dfrac{f_{ij}}{x_{ij}^k} & \text{if } x_{ij}^k>0\\ g(x^k, x^{k-1}, \ldots, x^1) & \text{otherwise,}\end{cases}$$

where c and f are the original variable and fixed costs and $g(\cdot)$ is a function that depends on the solutions of the previous iterations. In our implementation, the first objective function is computed from the solution obtained after the cut generation phase. Indeed, let $(\bar{x}, \bar{y})$ be such a solution. The first objective function is given by

$$\bar{c}_{ij}^1=\begin{cases}c_{ij}+\dfrac{f_{ij}}{\bar{x}_{ij}} & \text{if } \bar{x}_{ij}>0\\ c_{ij}+\dfrac{f_{ij}}{U} & \text{otherwise.}\end{cases}$$

Then, at iteration $k + 1$, we define the cost function as follows:

$$\bar{c}_{ij}^{k+1}=\begin{cases}c_{ij}+\dfrac{f_{ij}}{x_{ij}^k} & \text{if } x_{ij}^k>0\\ \lambda\bar{c}_{ij}^r+(1-\lambda)\left(c_{ij}+\dfrac{f_{ij}}{U}\right) & \text{otherwise,}\end{cases}$$

where $\lambda \in (0, 1)$, and $r \in \{1, \ldots, k - 1\}$ is the last iteration in which $x_{ij}^r > 0$ and the cost assigned was $\bar{c}_{ij}^r$.

If $x^{k+1} = x^k$, then stop. Otherwise, go to the next iteration. If the maximum number of iterations is attained, we stop. If no solution has been found, we apply a rounding heuristic. Let $A' = \{a \in A : \bar{y}_a > 0\}$. Find a feasible flow x^* using just the arcs of A'. Set $y_a^* = 1$ if $x_a^* > 0$. Finally, (x^*, y^*) is the heuristic solution.

- *Min cost flow*. Here, we solve a minimum-cost flow problem on the graph defined by the open arcs of the current fractional solution. In other words, given a fractional solution $(\bar{x}, \bar{y})$, we define the graph $G' = (V, A')$, where $A' = \{(i, j) \in A : \bar{y}_{ij} > 0\}$. The objective function is defined as

$$\bar{c}_{ij}(\bar{x})=\begin{cases}c_{ij}+\dfrac{f_{ij}}{\bar{x}_{ij}} & \text{if } \bar{x}_{ij}>0\\ c_{ij}+\dfrac{f_{ij}}{u_{ij}} & \text{otherwise}\end{cases}\quad \text{for all } (i,j)\in A'.$$

The solution to this problem is obtained with the network simplex implementation MCF [28]. The basis is stored and reused as the initial basis in the next call.

The slope scaling procedure is called at the top node, whereas the min-cost flow heuristic is called at 10 consecutive nodes every 100 nodes in the branch-and-bound tree.

4.4. Branching Rules

Whereas many specialized branch-and-cut codes use simple variable branching rules such as most fractional, most costly, etc., commercial MIP systems use pseudocosts based on dual variable estimates. As UFC falls somewhat between the general and the special purpose, we attempted to compare some of the simple branching rules.

Below, we briefly present the variable and constraint branching rules that have been tried:

- *Variable Branching*. The rules that we have studied are
  - — *Closest to integer*. Branch on the variable that maximizes $w_a = \max\{\bar{y}_a, 1 - \bar{y}_a\}$ over the set of arcs $a \in A$ such that $\bar{y}_a \in (0, 1)$.
  - — *Furthest from integer*. This criteria is similar to the previous one. Branch on the variable that minimizes $w_a = \max\{\bar{y}_a, 1 - \bar{y}_a\}$.
  - — *Maximum fixed charge*. Branch on the variable for which the fixed cost f_a is maximum among those with $\bar{y}_a$ fractional.
  - — *Maximum remaining fixed charge*. Branch on the variable that maximizes $w_a = (1 - \bar{y}_a) \cdot f_a$ among those with $\bar{y}_a$ fractional.
  - — 2-*strong branching* (2-*st*). Select the two arcs a^1, a^2 with the biggest and second biggest fixed charge for which $\bar{y}_{a^i}$ is fractional. Then, compute (using five iterations of dual simplex) a lower bound on the value of the possible successor nodes, namely, z_0^1, z_1^1 for a^1 and z_0^2, z_1^2 for a^2 when the variable is fixed to zero and one, respectively. Then, compute $w^1 = \min\{z_0^1, z_1^1\}$ and $w^2 = \min\{z_0^2, z_1^2\}$ and branch on the arc a^i for which w^i is larger.
- *Constraint Branching*. Branching can also be based on linear inequalities. Here, we consider the possibility of branching on subtour constraints, motivated by the fact that optimal solutions of uncapacitated problems do not contain cycles (Proposition 2.1). These inequalities have the following form:

$$\sum_{a\in E(S)} y_a \le |S| - 1 \quad \forall S \subset V.$$

Given the solution at the current node, determine a fractional subtour, that is, a set S for which there exists at least one arc $a \in E(S)$ with $\bar{y}_a$ fractional. Then,

- — If the subtour is encountered for the first time, compute its value, that is, $l(S) = \sum_{a\in E(S)} \bar{y}_a$. If this value is less than $|S| - 1$ and greater than or equal to one, then the branching constraints chosen are $\sum_{a\in E(S)} y_a \le \lfloor l(S) \rfloor$ and $\sum_{a\in E(S)} y_a \ge \lceil l(S) \rceil$. The

node to be solved in the next iteration is one of the successors of the current node.
— Otherwise, if the subtour has already been used, select the most fractional variable in $E(S)$ to branch on.

4.5. Pruning Criteria and Variable Fixing

When the variable and fixed costs are nonnegative, there exists an optimal solution that is cycle-free (Proposition 2.1). This allows us to prune some nodes of the enumeration tree and also provides a test allowing us to fix an arc variable y_a to zero, if arc a plus the arcs fixed to one form a cycle. Both these tests are carried out before solving the linear program at each node of the enumeration tree.

5. A SET OF TEST PROBLEMS

Several problem classes have been used to test our implementation. Here, we describe the test instances and how they have been generated. Some of the problems are from the literature and the others have been randomly generated. Problems are classified according to the number of source nodes and the structure of the underlying graph. The different classes of problems are shown below:

$$\text{single source}\begin{cases}\text{Grids}\\ K_n\\ K_n+1\\ \text{Steiner}\\ \text{Multisegment}\\ \text{Multilevel LS}\end{cases}\qquad \text{multisource}\begin{cases}\text{Grids}\\ \text{Series-parallel}\\ K_n\\ \text{Planar}\\ \text{Random.}\end{cases}$$

In all cases, the formulation that we have used is based on (1)–(4) from Section 2. For the single-source cases, the structure of the optimal extreme solutions says that for every node only one inflow arc can have a positive flow. Thus, the additional tree constraint $\sum_{j\in V_i^-} y_{ji} \le 1$ is valid for all $i \in V$.

Now, we explain how the different instances have been created or indicate their origin:

- *Grids*. Here, the graph is a two-dimensional rectangular grid. The parameters used to generate a grid problem are the width and the height (in number of nodes), the total demand, the number of source and demand nodes, and bounds on the costs. Demand and supply nodes are selected randomly as well as is the fraction of the demand/supply assigned to a node. The random number generator is that of NETGEN [26]. The variable and fixed costs are uniformly generated over the specified interval using the C/C++ random number generator.
- *Complete*, K_n. The graph is complete. The number of nodes, the bounds on the costs, and the total demand are the parameters needed to specify an instance. The source and demand nodes, the demands and supplies, and the costs are chosen in the same way as for grid graphs.
- *Random*. Here, the number of nodes, arcs, source nodes, demand nodes, total demand, and costs intervals are specified. The procedure iteratively selects an arc that has not already been chosen and then assigns costs. Finally, we define the source and demand nodes using the same strategy as before.
- *Planar*. To generate planar graphs, we used the planar graph generator included in LEDA [32]. The position and the number of the source and demand nodes are determined with the uniform random number generator from LEDA. The cost function is computed as for Grids.
- *Series-parallel*. Here, we use the series-parallel graph generator from LEDA. The rest of the parameters, number of source and demand nodes, and demands and costs are calculated as for planar graphs.
- *Hochbaum*, $K_n + 1$. These are single-source instances taken from [23]. To clarify the description, suppose that node 1 is the source node and $\{2, \ldots, n\}$ are the other nodes. The set of arcs consist of $(1, j)$ for every $j > 1$ and (i, j) for every $i, j > 1$, $i \neq j$. So, the nodes $\{2, \ldots, n\}$ induce a complete directed graph. The demand of each node is uniformly selected from $\{0, 1, 2, \ldots, 10\}$. The fixed cost is a multiple of the variable cost. We use the same factor 50 as in the paper. To generate the variable costs, we randomly select n points in the plane that are associated with each node. The variable cost of an arc is then calculated as the distance between the points corresponding to each end node. In other words, if (w_i, h_i) and (w_j, h_j) are the coordinates of the nodes i and j, the cost of the arc (i, j) is given by $c_{ij} = \sqrt{(w_i - w_j)^2 + (h_i - h_j)^2}$.
- *Steiner*. These undirected instances were obtained from the Steiner problem Library at ZIB `ftp://ftp.zib.de/pub/Packages/mp-testdata/index.html`. A complete description of all these problems can be found in [27]. In our tests, we selected a small subset of the instances:

  — *Beasley*. Problems 1, 2, and 3 of series C described in [6]. Instances from series B were not considered as they are too easy.
  — *X*: Instances brasil and berlin. These have complete graphs and Euclidean weights.
  — *Mc*7, *Mc*8, *and Mc*11. These instances were described in [27].

The formulation that we have used is again (1)–(4) from Section 2. An arbitrary node is defined as root node with supply equal to the number of terminal nodes minus one. Each terminal node has a demand of one unit. Undirected edges have been modeled with two directed arcs.

- *Multisegment*. These are single-source problems with a concave piecewise linear objective function (see [19] for a survey on this kind of model). One of the instances `beavma`, which comes from a practical harvesting problem in Chile, involves a planar graph and the objective function has at most two segments per arc. The others are randomly generated instances. `mtest4ma` has at most four segments. `g150x1100`, `k15x420`, and `p50x576`

TABLE 1. Instance classification: Summary of results.

	Hard (36 instances)			Medium (8 instances)			Easy (39 instances)		
Parameter	Cplex	bc-opt	mp-opt	Cplex	bc-opt	mp-opt	Cplex	bc-opt	mp-opt
# solved	0	0	0	0	2	6	37	34	38
$\langle gap \rangle$	29.97	19.43	18.57	15.54	2.66	7.64	0.06	0.00	0.01
$\langle \Delta LP \rangle$	45.42	69.50	67.45	41.47	87.43	81.10	69.02	90.73	93.98

have two segments. The other three instances have three segments. To model multiple segments, we use a directed multigraph. The formulation used is given by $x_{ij} = \sum_k x^k_{ij}$, $x^k_{ij} \leq u^k_{ij} y^k_{ij}$ and $y_{ij} = \sum_k y^k_{ij} \leq 1$.
In other words, each arc with more than one segment is repeated as many times as is the number of segments in the objective function. The constraint that at most one of the arcs can have a positive flow is then added.

- *fixnet*6 is a single source-problem from the MIPLIB [9].
- *Multilevel lot-sizing*. The production in series lot-sizing problem [35] can be formulated as a single-source UFC problem. The parameters required to generate these instances are the number of periods, the number of levels, the maximum demand, and the intervals for the costs. The fixed and variable costs are uniformly selected from the given interval, except that the fixed costs are zero on the stock arcs. The demand is uniformly selected from zero to the maximum demand. The supply is the negative sum of all the demands. Note that this problem is polynomialy solvable by dynamic programming. However, although many strong valid inequalities are known that generalize single-level inequalities, a complete description of the convex hull is not known.

An instance is named according to the graph type and its size or origin. In the first case, a typical instance name is `tnxm`, where `t` is the type of the graph; `n`, the number of nodes, and `m`, the number of arcs. Graph types are `g` grid, `p` planar, `k` complete, `sp` series-parallel, `r` random, `l` multilevel lot-sizing, and `h` hochbaum. Instances coming from the literature keep their original name.

Our test set consists of 39 multisource instances and 45 single-source instances. A complete list of the instances, some of their principal characteristics, and the data sets can be found in [43]. This site also contains two appendices containing solutions of all the instances from the computational experiments described in the next section.

6. COMPUTATIONAL EXPERIMENTS

The computational results reported below can be viewed as an attempt to answer the following questions:

Q1. How well do standard commercial mixed-integer programming systems perform on this class of problems?
Q2. What is the effect of tightening the formulation of UFC with dicut inequalities and their variants?
Q3. Can the specific structure of UFC be used to improve features of the branch-and-cut process such as primal heuristics, branching strategies, etc.?
Q4. Can anything be said about the complexity of different instances of UFC as a function of the graph structure, the number of source nodes, or the cost structure?
Q5. For specific classes of UFC, how effective is the general approach of reformulating with dicuts as compared with algorithms developed for the specific problem class?

Specifically, in Subsection 6.1, we report on the behavior of three MIP systems on our test set and use the results to obtain an initial classification of the difficulty of the 83 test instances. In Subsection 6.2, we first present the results of the preliminary tests carried out to define a default strategy for `bc-nd` and then the results on the complete test set are reported. In Subsection 6.3, we examine what happens when we add the dicut inequalities found by `bc-nd` *a priori* at the top node and then call the three systems. In Subsection 6.4, we look at the effect of variations in some of the parameters, such as the graph structure, the number of source nodes, and the cost ratio. Additionally, comments on specialized codes for the different special cases are presented at the end of this section. Finally, in Subsection 6.5, we test the use of the multicommodity reformulation.

6.1. Instance Classification

The three MIP systems used to solve the test instances were Cplex 6.6 [24] on a Sun Ultra 60, 250 MHz, 256 MB RAM running Sun Solaris 2.6, `mp-opt` v11.50o [16], and `bc-opt` [13] on a Pentium II, 400 MHz, 128 MB RAM running Windows NT 4.0. All the numerical tolerances were fixed to 10^{-6}. The maximum time per instance was fixed to be 1800 CPU seconds.

Based on the results obtained, we classified the test instances into three classes: `Easy`, `Medium`, and `Hard`. An instance is `Easy` when at least two of the three systems solve it to optimality. It is `Medium` when only one of the systems solves it to optimality and `Hard` when no system solves it. Among the 83 instances, 39 are `Easy`, 8 are `Medium`, and 36 are `Hard`. Table 1 summarizes the results for each system. The first line, # solved, is the number of instances solved to optimality; the second $\langle$gap$\rangle$ is the average gap of unsolved instances, where gap $= 100(BestIP - BestBound)/(BestIP)$; and the third $\langle \Delta$LP$\rangle$ is the aver-

age lower-bound improvement using the cuts of the system, where $\Delta\text{LP} = 100(XLP - LP)/(BestIP - LP)$. Here, LP is the value of the linear relaxation, XLP is the value of the LP relaxation after the addition of systems cuts at the top node, *BestBound* is the value of the lower bound at the end of the enumeration, and *BestIP* is the value of the best integer solution found.

6.2. *Solving UFC with* `bc-nd`

The first step when using a branch-and-cut system is to define the default strategy to be adopted. In this case, we had to decide the cut generation strategy, the branching strategy, and the option of if and when to use the primal heuristic. The results of these preliminary tests are presented in the next subsection. `bc-nd` was run on a Pentium II, 400 MHz, 128 MB RAM under Windows NT 4.0.

6.2.1. Choosing a Default Strategy. The first step is the cut generation strategy, namely, which families of inequalities to use, how to separate these families, which cuts to keep, and when and how often to separate.

To decide which variants of the dicut inequality to include in the final prototype, our first implementation included simple dicut, mixed dicut, simple inflow–outflow, and the mixed dicut with outflow inequalities. By evaluating each of them separately on the test set, the best results were obtained with just the simple dicut and the simple inflow–outflow inequalities.

In choosing which cuts to keep, the first observation, much as expected, is that simple dicut inequalities involving just the 0–1 arc variables typically produce a much more significant increase in the linear programming lower bound than do the various mixed dicut inequalities which include the flow variables. The second is that simultaneously adding simple and mixed dicut inequalities for the same node set seems to lead to LP degeneracy and increases running times. So, the default for each active set is to add a violated simple dicut inequality if possible and, if none is found, to try to add just a simple inflow–outflow (mixed dicut) inequality.

How often to separate, when to delete cuts, and when to make cut sets inactive were determined, in part, by testing and, in part, by earlier experience with `bc-opt`. The final strategy selected is as follows: Thirty passes are made, nonbinding cuts are deleted at the beginning of each pass, cut pool separation is performed, and a threshold of -3 is used to discard node sets. This strategy is based on a trade-off between the quality of the resulting lower bound and the execution time, in that the LP relaxation is not too difficult to solve and the memory requirements are reasonable.

The second part concerns the branching rules: The rules tested are least fractional, most fractional, maximum fixed charge, and strong branching with a candidate list of two elements. Thirteen instances were selected for this experiment, of which three were Steiner tree instances and 10 multisource instances. The latter set contains four grid graphs, two complete graphs, two planar graphs, and two random graphs. The results are summarized in Table 2. The first column gives the rule; the second, the number of instances solved within 1800 seconds; the third, the average number of nodes evaluated; and the fourth, the average duality gap at the end of the enumeration for the unsolved problems.

TABLE 2. Branching rules.

Rule	Solved	Avg # nodes	⟨gap⟩
least frac	3	11,075	1.87
most frac	1	13,838	2.55
max f_a	7	7715	1.22
max 2-st	2	3137	1.80
subtour	3	7866	4.35
Xpress	7	17,253	1.94

From Table 2, we conclude that, given the time limit, the best strategy is to use either the branching selection rule provided by the Xpress library or the maximum fixed charge, max f_a. To decide between them, an additional test was carried out on the complete test set. The results showed that with the Xpress rule 53 instances were solved, with an average number of nodes visited of 4518.6 and an average time to find an optimal solution of 177.47 seconds. On the other hand, with the maximum fixed-charge rule, 52 instances were solved, with an average number of visited nodes of 4403.9 and an average solution time of 125.8 seconds. Somewhat arbitrarily, we selected the Xpress rule.

In addition, we chose best bound node selection, in the belief that it is effective when the duality gaps after adding cuts are very small.

The third part of this experiment is related to the effects produced by the min-cost flow heuristic within the enumeration phase of the algorithm. First, we select the instances that need enumeration to be solved or, in other words, those that are not solved at the top node by `bc-nd`. There are 59 such instances. Then, we use `bc-nd` without and with the heuristic. The rest of the parameters are the same in both runs. The results are summarized in Table 3. The first column is the class, the second is the number of instances falling within the class, the next four columns summarize the results without using the heuristic, and the last four summarize the results with the heuristic. The columns # sol and ⟨gap⟩ are defined as before: sol^0 is the average number of nodes for the instances that are solved and unsol is the average number of nodes for the instances that cannot be solved either with or without the heuristic. An * appearing as a superscript of ⟨gap⟩ means that no feasible solution has been found for one instance.

The results reported in Table 3 show that 30 instances are solved with the primal heuristic and 23 instances are solved without the primal heuristic. For the instances solved to optimality by both versions, the number of nodes needed to

TABLE 3. Results with and without the heuristic.

		Without heuristic				With heuristic			
				⟨nodes⟩				⟨nodes⟩	
Class	# Instances	# sol.	⟨gap⟩	sol^0	unsol	# sol.	⟨gap⟩	sol^0	unsol
grid	8	2	7.0*	621	8167	4	5.0	423	1399
K_n	6	3	12.0	455	14,342	4	3.8	182	37,913
plan	10	1	9.0	53	4660	1	4.1	17	7138
rand	5	1	5.0	5661	10,200	2	2.3	923	17,849
s–p	2	2	0	2073	—	2	0	6	—
stein	8	2	3.0	69	2120	2	2.3	16	986
m–s	5	3	5.0	35	1574	3	0.6	19	473
grid	3	2	0	18	—	3	0	9	—
$K_n + 1$	8	0	∞	—	385	3	12.9	769	342
K_n	4	4	0	28	—	3	0	18	—
l–s	1	1	0	3	—	1	0	23	—
Total	59	23				30			

prove optimality is significantly smaller when using the heuristic. Also, for the instances that cannot be solved by either version, the final gap is smaller when using the heuristic. So, the default choice is to use the heuristic at a frequency described in Subsection 4.3.

6.2.2. Using the Default Strategy: Results. Table 4 summarizes the results obtained by `bc-nd` with the default strategy for the set of test instances. The maximum time limit and the tolerances are fixed as before. The measures reported here are the same as in Table 1 with, in addition, $\langle \text{gap}^0 \rangle$, the average duality gap at the beginning of the enumeration, where $\text{gap}^0 = 100(\mathit{firstIP} - \mathit{LP})/\mathit{firstIP}$.

In comparing Table 4 with Table 1, we observe that, for the hard instances, the duality gap after 30 minutes is dramatically reduced. Moreover, the six medium instances can be solved within the limit time. The two unsolved medium instances are $K_n + 1$ instances that were solved by `bc-opt`. The average value of ΔLP, the improvement obtained by reformulating the problem using the dicuts, is 92.61%. In other words, a large part of the gap is directly closed by the dicut inequalities. In fact, 24 instances are solved without enumeration. It is also worth pointing out that the average duality gap at the top node for the 36 `Hard` instances is 20.08%. Table 5 shows $\langle \text{gap}^0 \rangle$ for each class of graph.

TABLE 4. `bc-nd` summary results.

Parameter	`Hard` (36 instances)	`Medium` (8 instances)	`Easy` (39 instances)
# solved	8	6	39
⟨gap⟩	6.67	1.18	0.00
$\langle \text{gap}^0 \rangle$	20.08	24.11	1.17
⟨ΔLP⟩	85.25	97.23	98.46

6.3. Effectiveness of Dicut Inequalities

Given the classification of Subsection 6.1, we would like to study how much the dicut inequalities alone contribute to the overall improvement. To that aim, the instances are first reformulated using `bc-nd` and then given to the three systems, Cplex 6.6, `mp-opt` 11.50o, and `bc-opt` under the same conditions as in Subsection 6.1, namely, with a time limit of 1800 CPU seconds, all tolerances fixed to 10^{-6}, and the system default strategy for the cut generation and the enumeration. The reformulation of each instance is obtained using the default strategy of `bc-nd`. The results are summarized in Table 6. The measures reported are the following: # solved is the number of instances solved and $\langle \text{gap}^i \rangle$ and $\langle \text{gap}^r \rangle$ are the average gap of the nonsolved instances using the original formulation and the improved formulation. $\langle \text{Nodes}^i \rangle$ and $\langle \text{Nodes}^r \rangle$ are the average number of nodes used to solve the original and the improved formulation. Finally, $\langle T^i \rangle$ and $\langle T^r \rangle$ are the average time needed to solve the original and the improved formulation. It turns out that five `Hard` instances are solved by exactly one system and nine `Hard` instances by the three systems.

We also observe that all the `Easy` and `Medium` instances can be solved to optimality with the three systems. Moreover, compared with Table 1, the number of nodes and the time required are reduced. Fourteen of the 31 `Hard` instances are now solved. For the other 22 instances, the final duality gap is considerably reduced.

6.4. Structure and Comparison with Specialized Codes

In this subsection, we look at the subclasses of the UFC appearing in our test set. In Table 7, we consider their difficulty based on the classification of Subsection 6.1. The first four columns give the results for the single-source instances, and the last four columns, for the multisource

TABLE 5. Initial duality gap.

Multisource					Single source						
grid	K_n	plan	rand	s–p	stein	m–s	grid	$K_n + 1$	K_n	l–s	plan
5.97	8.76	5.31	6.94	0.30	10.59	1.04	0.01	84.5	0.04	0	0

instances. In each case, the first three columns give the number of `Hard`, `Medium`, and `Easy` instances in the class, and the last column, the average improvement obtained with the dicut inequalities at the top node using `bc-nd`.

We see that of the single-source instances 14 are `Hard`, five `Medium`, and 26 `Easy`, while of the multisource instances, 22 are `Hard`, three `Medium`, and 13 `Easy`. Below we analyze the results by problem class:

Single-source instances. The average gap reduction for these instances is 92.10%, meaning that dicut inequalities are effective when solving single-source instances. We now discuss the results obtained for each specific subclass of UFC:

- *Steiner*. These are the only single-source instances for which `bc-nd` performs poorly in that the gap reduction obtained is only 94% as opposed to 99% on the other instances. One possible reason is that the UFC formulation (1)–(4) is unsuitable for the Steiner problem and, in addition, that the number of flow and setup variables is doubled so as to produce a directed network flow problem. However, compared to the three general systems, `bc-nd` is much more effective. In Table 9, we see that on the seven hard instances the final duality gap with `bc-nd` is 2.54%, while for Cplex, `bc-opt`, and `mp-opt`, it is 49.21, 43.14, and 24.94%, respectively. However, if the dicut-tightened formulation is given to the three systems, six of these seven instances are solved at least once.

  This is one of the most well-studied special cases of the UFC. Recently, Koch and Martin [27] developed a successful branch-and-cut code for the undirected Steiner tree problem. The Steiner instances in the test set are all solved to optimality without branching by their specialized code within 20 seconds. However, as the authors pointed out, these remarkable solution times are due, in part, to the specialized preprocessing that leads to significant reductions in the size of the network. For example, *brasil* initially has 58 nodes and 1653 edges, whereas after preprocessing it has 39 nodes and 113 edges and the number of terminal nodes is reduced from 25 to 10. Similar reductions are obtained for the other instances.
- $K_n + 1$. Among the eight instances in the test set, `bc-nd` can solve three of them: the easy instance and two hard instances. It is interesting to point out that the two medium instances of this class were solved by `bc-opt` using path inequalities. Moreover, the two hard instances solved by `bc-nd` were not solved by the commercial codes after the reformulation. However, after reformulation, all three general codes were able to solve the medium and easy instances.

  The main difficulty encountered is that the number of variables involved in each dicut is fairly large, and, thus, the LP relaxation becomes difficult to solve. In fact, the LP relaxation of these instances is quite degenerate even without adding the dicuts. These difficulties are only significant for instances with more than 50 nodes.

  Note also that these instances are those for which the heuristics give the worst results. In Hochbaum and Segev [23], where these instances were introduced, two Lagrangian relaxations were studied. The first is obtained by relaxing the forcing constraints (3), and the second, by relaxing the flow balance constraints (2). They also incorporate three primal heuristics into their algorithm. They report a final average gap (defined as in Section 6.1) of 3%, with CPU times going from 10 to 80 seconds. The average gap, when the first primal solution is found, is around 8%.

  Additionally, they define a measure of the complexity of an instance depending on the cost function. The parameter is defined as the ratio between the maximum demand and twice the ratio between the fixed and variable cost. One of their empirical conclusions is that when the

TABLE 6. Reformulated instances: Summary of results.

	`Hard` (36 instances)			`Medium` (8 instances)			`Easy` (39 instances)		
Parameter	Cplex	`bc-opt`	`mp-opt`	Cplex	`bc-opt`	`mp-opt`	Cplex	`bc-opt`	`mp-opt`
# Solved	12	9	11	8	8	8	39	39	39
$\langle \text{gap}^i \rangle$	29.97	19.43	18.57	15.54	2.66	7.64	0.06	0.00	0.01
$\langle \text{gap}^r \rangle$	8.24	6.27	6.24	0.00	0.00	0.00	0.0	0.0	0.0
$\langle \text{Nodes}^i \rangle$	—	—	—	426,337	48,578	28,826	53,074	6620	10,687
$\langle \text{Nodes}^r \rangle$	—	—	—	667	62	1688	466	495	2061
$\langle T^i \rangle$	—	—	—	1800	840	582	203	169	83
$\langle T^r \rangle$	—	—	—	132	23	134	5	14	21

TABLE 7. Complexity depending on problem subclass.

	Single source				Multisource			
	Hard	Medium	Easy	ΔLP	Hard	Medium	Easy	ΔLP
grid	0	3	3	99.97	6	0	2	90.92
K_n	0	0	5	99.84	3	0	3	89.38
plan	0	0	5	99.99	9	0	1	92.47
rand	—	—	—	—	3	0	2	89.30
s–p	—	—	—	—	1	3	5	99.10
stein	7	1	0	94.83	—	—	—	—
m–s	2	0	6	98.91	—	—	—	—
K_n + 1	5	2	1	62.62	—	—	—	—
l–s	0	0	4	99.99	—	—	—	—

parameter lies in the interval [0.05, 0.1] the instances seems to be hard. For the instances in our test set, the value of this parameter is 0.1.

- *Lot-sizing*. All these instances turn out to be easy, with `mp-opt` being the fastest. We observe that the strengthened lower-bound XLP obtained with dicuts using `bc-nd` is better than that obtained with the path inequalities from `bc-opt`. One possible reason is that, for uncapacitated lot-sizing problems, appropriately chosen inflow–outflow inequalities resemble the path inequalities that give a complete description of the convex hull of solutions in the single-item case and also provide strong valid inequalities for multilevel problems.
- *Multisegment*. `bc-nd` seems to be highly effective on this class. For example, the two instances beavma and mtest4ma are both solved without enumeration. Neither could be solved with the versions of Cplex and `mp-opt` available 4 years ago, although `bc-opt` was able to solve both instances, making extensive use of the path inequalities. Now, `mp-opt` also solves them both at the top node, even more rapidly than does `bc-nd`.
- *Grids, planar and K_n*. Eight of the 16 instances are solved without enumeration, and for the other instances, the average gap reduction at the top node is 99.93%. For identical graph and cost structure, the single-source instances appear to be much easier than are the multisource instances. For such instances, no specialized code is known.
- *Other single-source instances*. During the development of `bc-nd`, we received a new set of 36 instances used by Cruz et al. [14], who implemented a Lagrangian relaxation to solve the single-source case of UFC. The instances are randomly generated in the plane with 16–32 nodes and 30–248 arcs. The objective function is defined using the Euclidean distance between node i and j, such that the ratio between the fixed and variable cost is fixed. The values of that ratio are 1/10, 1, and 10/1. Each instance is solved by `bc-nd` within 2 seconds, several of them without enumeration.

Multisource instances. Although 23 of the 38 instances can be solved to optimality by `bc-nd`, these instances are considerably more difficult to solve than are the single-source instances. In particular, the average gap reduction is only 90%. The exceptions are the instances defined on series-parallel graphs on which both `bc-nd` and `mp-opt` perform well.

We do not know of any other code developed for this type of model.

6.4.1. Importance of Fixed/Variable Cost Ratio. In this experiment, we took a subset of 27 instances of different kinds from our list and solved them with `bc-nd` (default strategy) for four ratios: 10, 100, 1000, and ∞ (that means $f_{ij} = 1$, $c_{ij} = 0$). A summary of the results is presented in Table 8, which is organized as follows: The first two columns give the class and the number of instances used in the experiment; the next four columns give in the first line # solved and in the second line ⟨gap⟩, both defined as before; the last four columns give the average value of ΔLP in the first line and the average solution time of the solved instances in the second line. The single-source instances were put all together in one line S-s, due to the homogeneity of the results obtained. A * appearing as superscript of ⟨gap⟩ means that one of the instances was not solved due to a numerical problem.

As we might expect, instances become harder when the fixed/variable cost ratio increases. For the single-source instances, the execution time increases when the ratio increases, but the gap reduction at the top node does not show a clear trend. In the multisource case, we observe that the gap reduction decreases when the cost ratio increases. However, some strange behavior is observed. For example, the time required for the grid graphs and planar graphs is much smaller when the cost ratio is 10^3 than when it is 10^2. The same occurs for series-parallel graphs for ratios 10^3 and ∞. Finally, the number of instances solved to optimality decreases when the cost ratio increases.

6.5. The Multicommodity Reformulation

Here, we first present the multicommodity reformulation [36] and then we describe our results. We introduce new variables z_{ij}^k to represent the flow in arc $(i, j) \in A$ with destination $k \in V_D$. The resulting formulation is then

TABLE 8. Modifying the cost ratio: Summary of results.

	# Instances	# Solved ⟨gap⟩				⟨ΔLP⟩ ⟨T⟩			
		10	10^2	10^3	∞	10	10^2	10^3	∞
S–s	9	8	8	8	7	99.43	88.9	91.54	97.70
		0*	0.12	0.73	2.1	13.7	277.1	531	873.7
grid	5	4	3	3	0	99.45	96.40	95.91	86.62
		5^{-2}	1.5	5.0	9.0	36.5	258.6	50	—
plan	5	5	4	3	0	99.95	99.17	99.21	85.96
		0	0.2	1.0	9.0	100.2	514.5	6	—
rand	5	4	4	4	0	97.59	97.16	95.99	75.89
		0.015	7.3	9.0	19.0	24.5	43.2	235	—
s–p	2	2	2	2	2	100	100	99.98	99.54
		0	0	0	0	3	16.5	445	51.5

$$\min \sum_{k \in V_D} \sum_{(i,j) \in A} c_{ij} x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij}$$

$$\sum_{j \in V_i^-} z_{ji}^k - \sum_{j \in V_i^+} z_{ij}^k = d_i^k \ \ \forall\, i \in V,\ k \in V_D$$

$$z_{ij}^k \le b_k y_{ij} \ \ \forall\, (i,j) \in A,\ k \in V_D$$

$$z_{ij}^k \ge 0 \ \ \forall\, i \in V,\ k \in V_D,\ y_{ij} \in \{0, 1\} \ \ \forall\, (i,j) \in A,$$

where $d_1^k = -b_k$, $d_k^k = b_k$, $d_i^k = 0$ for $i \in V \backslash \{1, k\}$ and all $k \in V_D$. In Table 9, we show which single-source instances are solved by Cplex 7.5 and/or `mp-opt12.29` within 1800 seconds. These results were obtained on a DELL 1 GhZ computer with 256 MB RAM.

For each of the instances in Table 9, the linear programming solution is integer. However, for all the other 13 instances tested, `g150x1650`, `g200x740d`, `g200x740e`, `h50x2450`, `h50x2450b`, `h50x2450c`, `h50x2450e`, `h80x6320`, `h80x6320b`, `h80x6320c`, `h80x6320d`, `mc8`, and `mc11`, neither code was able to solve the linear program within the allotted time of 1800 seconds.

It is not known how to obtain a tight multicommodity reformulation for multisource instances of the UFC. The obvious generalization is to use the formulation given above, but with d_k^i a variable for all sources $i \in V_S$ and demand nodes $k \in V_D$, along with the additional constraints

$$\sum_k d_k^i = -b_i \text{ for } i \in V_S.$$

For the four instances shown in Table 10, the linear programming bound is weak and the reformulation does not appear to be very effective. As classified above, `r30x160` is an `Easy` instance and `g200x740`, `g55x188`, and `k20x380` are `Hard` instances. However, when solved by `bc-nd` as in Section 6.2.2, `g200x740` ends with a gap of 0.33% and the other two `Hard` instances are solved.

7. CONCLUSIONS

In spite of the improvements brought about by the introduction of cutting planes into commercial mixed-integer programming systems, it appears that the solution of uncapacitated fixed-charge network design problems is significantly improved by the use of dicut inequalities either

TABLE 9. A multicommodity formulation: Single-source instances.

	Single-source instances						
		Cplex			mp-opt		
Name	LP	BestIP	Time	# Nodes	BestIP	Time	# Nodes
beasleyC2	144	144	3	0	144	14	0
beasleyC3	754	754	74	0		1800^{lp}	
berlin	1044	1044	151	0	1044	70	0
brasil	13,655	13,655	409	0	13,655	242	0
`g150x1100`			1800^{lp}		71,816	1085	0
`g200x740c`	680,124	680,124	135	0	680,124	323	0
`mc7`	3417	3417	1594	0		1800^{lp}	

TABLE 10. A multicommodity formulation: Multisource instances.

Multisource instances											
		Cplex					mp–opt				
Name	LP	XLP	Best LB	BestIP	Time	# Nodes	XLP	Best LB	BestIP	Time	# Nodes
g200x740	36,432.8	40,844.9	40,897.3	76,468	1800	601	39,767.0	39,767.0		1800	100
g55x188	17,763.0	19,709.2	20,865.7	26,748	1800	2771	18,908.8	19,283.7	29,283	1800	600
k20x380	1339.9	1601.3	1756.5	2018	1800	2425	1516.7	1587.2	2156	1800	1900
r30x160	15,786.6	17,759.3	20,160.1	22,979	1800	7646	18,314	19,153.8	23,776	1800	1400

within a special-purpose system such as bc–nd or by giving a dicut-strengthened reformulation to an MPS system. Another important step in the development has been the use of a dynamic active node set list as part of the separation heuristic for dicut inequalities. This has been particularly important for certain single-source instances where the number of potential node sets has exploded rapidly.

Further work on the choice of dicut inequalities is certainly needed. It would be interesting to design and test separation routines for input–output inequalities, and preliminary tests indicate that the mixing (Günluk and Pochet [21]) of dicut inequalities may be valuable. Also, given that the multidicut inequalities arise from the projection of the multicommodity formulation for single-source problems [37], a good separation inequality for these inequalities would be very helpful in improving the bounds, and one possibility might be to generate some of these inequalities through mixing. There clearly is also a need to find new classes of valid inequalities and heuristics for multisource instances.

Another difficulty concerns actual MIP systems. One reason that bc–nd does not more significantly outperform "dicut reformulation + an MIP system" is that the MIP system uses its powerful preprocessor to reduce the tightened formulation, while bc–nd keeps the initial tightened formulation as it needs the network structure for cut generation and its primal heuristics. This handicap will only be overcome when the MPS subroutine libraries provide a two-way mapping between the original and preprocessed matrices.

Obvious developments are to extend the system to tackle single-commodity *capacitated* fixed-charge network design problems and to *multicommodity* problems. For the capacitated problem, the set X_S (5)–(7) modified with active capacity constraints is the natural starting point as a variety of valid inequalities (flow cover, mixed integer rounding, etc.) can be generated for the modified X_S, and the heuristics to generate good node sets S probably need only minor modifications. For multicommodity problems, nearly all the inequalities used to date are direct adaptations of single-commodity inequalities.

REFERENCES

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, Network flows, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] A. Atamtürk, On the network design cut set polyhedra, Draft, Department of Industrial Ingineering and Operations Research, University of California at Berkley, 1999.

[3] M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, "Network models," Handbooks in OR, Elsevier, Amsterdam, 1995, Vol. 7.

[4] I. Barany, T.J. van Roy, and L.A. Wolsey, Uncapacitated lot-sizing: The convex hull of solutions, Math Program Stud 22 (1984), 32–43.

[5] R.S. Barr, F. Glover, and D. Klingman, A new optimization method for large scale fixed charge transportation problems, Oper Res 29 (1981), 448–463.

[6] J.E. Beasley, An sst-based algorithm for the steiner problem in graphs, Networks 19 (1989), 1–16.

[7] G. Belvaux, Modelling and solving lot-sizing problems by mixed integer programming, Ph.D. thesis, F.S.A.—Université catholique de Louvain, 1998.

[8] D. Bienstock, Experiments with a network design algorithm using epsilon-approximate linear programs, Technical report TR-1999-4, Computational Optimization Research Center, Columbia University, New York, 1996.

[9] R.E. Bixby, S. Ceria, C.M. McZeal, and M.W.P. Savelsbergh, An updated mixed integer programming library: MIPLIB 3.0, Optima 58 (1998), 12–15. Problems available at http://www.caam.rice.edu/bixby/miplib/miplib.html.

[10] A.V. Cabot and S.S. Erenguc, Some branch and bound procedures for fixed-cost transportation problems, Nav Res Log Q 31 (1984), 145–154.

[11] S. Chopra, E. Gorres, and M.R. Rao, Solving a Steiner tree problem on a graph using branch and cut, ORSA J Comput 4 (1992), 320–335.

[12] T. Christof and A. Löbel, PORTA—A polyhedron representation and transformation algorithm, ZIB, Konrad-Zuse-Zentrum für Informationstechnik Berlin.

[13] C. Cordier, H. Marchand, R. Laundy, and L.A. Wolsey. bc–opt: A branch and cut code for mixed integer programs, Math Program 86 (1999), 335–353.

[14] F.R.B. Cruz, J. MacGregor Smith, and G.R. Mateus, Solving to optimality the uncapacitated fixed-charge network flow problem, Comput Oper Res 25 (1998), 67–81.

[15] Dash Associates, XPRESS-MP extended modelling and optimisation subroutine library, Reference manual, Release 11, Blisworth House, Blisworth, Northants NN73BX, U.K., 1999.

[16] Dash Associates, XPRESS-MP, Reference manual, Release 11, Blisworth House, Blisworth, Northants NN73BX, U.K., 1999.

[17] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W.H. Freeman, San Francisco, 1979.

[18] M.X. Goemans, personal communication, Dec. 1998.

[19] G.M. Guisewite and P.M. Pardalos, Minimum concave-cost network flow problems, Ann Oper Res 25 (1990), 75–100.

[20] O. Günlük, A branch-and-cut algorithm for capacitated network design problems, Math Program 86 (1999), 17–39.

[21] O. Günlük and Y. Pochet, Mixing mixed-integer inequalities, Math Program 90 (1998), 429–457.

[22] J.W. Herrman, G. Ioannou, I. Minis, R. Nagi, and J.M. Proth, Design of material flow networks in manufacturing facilities, Technical research report T.R.94-50, ISRI, University of Maryland, 1994.

[23] D. Hochbaum and A. Segev, Analysis of a flow problem with fixed charges, Networks 19 (1989), 291–312.

[24] ILOG, Cplex 6.5, User's manual, 2000.

[25] D. Kim and P. Pardalos, A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure, Oper Res Lett 24 (1998), 195–203.

[26] D. Klingman, A. Napier, and J. Stutz, Netgen: A program for generating large scale capacitated assignment, transportation, and minimum-cost flow network problems, Mgmt Sci 20 (1974), 814–820.

[27] T. Koch and A. Martin, Solving Steiner tree problems in graphs to optimality, Networks 32 (1998), 207–232.

[28] A. Löbel, MCF: A network simplex implementation, Konrad–Zuse–Zentrum für Informationstechnik Berlin, http://www.zib.de/Optimization/Software/Mcf/, 2000.

[29] T.L. Magnanti and L.A. Wolsey, "Optimal trees," Handbooks in OR, Elsevier, Amsterdam, 1995, Vol. 7, Chapter 9.

[30] H. Marchand, A. Martin, R. Weismantel, and L.A. Wolsey, Cutting planes in integer and mixed integer programming, DP 9953, CORE, Université catholique de Louvain-la-Neuve, 1999.

[31] R.K. Martin and L. Schrage, Subset coefficient reduction cuts for 0/1 mixed integer programing, Oper Res 33 (1985), 505–526.

[32] K. Mehlhorn, S. Näher, M. Seel, and C. Uhrig, The LEDA user manual, Version 4.1. http://www.mpi-sb.mpg.de/LEDA/MANUAL/MANUAL.html, 2000.

[33] G.L. Nemhauser and L.A. Wolsey, Integer and combinatorial optimization, Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley, New York, 1988.

[34] U.S. Palekar, M.K. Karwan, and S. Zionts, A branch and bound method for the fixed charge transportation problem, Mgmt Sci 36 (1990), 1092–1105.

[35] Y. Pochet and L.A. Wolsey, Algorithms and reformulations for lot sizing problems, DIMACS Ser DM TCS 20 (1995), 245–293.

[36] R.L. Rardin and U. Choe, Tighter relaxations of fixed charge network flow problems, Technical report J-79-18, Industrial and System Engineering, Georgia Institute of Technology, Atlanta, GA, 1979.

[37] R.L. Rardin and L.A. Wolsey, Valid inequalities and projecting the multicommodity extended formulation for uncapacitated fixed charge network flow problems, Eur J Oper Res 71 (1993), 95–109.

[38] J.E. Schaffer, Use of penalties in the branch and bound procedure for the fixed charge transportation problem, Eur J Oper Res 43 (1989), 305–312.

[39] M. Sun, J.E. Aronson, P.G. McKeown, and D. Drinka, A tabu search heuristic procedure for the fixed charge transportation problem, Eur J Oper Res 106 (1998), 441–456.

[40] S. van Hoesel, Models and algorithms fot the single item lot-sizing problem, Ph.D. thesis, Erasmus University, Rotterdam, 1991.

[41] T.J. van Roy and L.A. Wolsey, Valid inequalities and separation for uncapacitated fixed charge networks, Oper Res Lett 4 (1985), 105–112.

[42] T.J. van Roy and L.A. Wolsey, Solving mixed integer programming problems using automatic reformulation, Oper Res 33 (1987), 45–57.

[43] L.A. Wolsey, Personal webpage, http://www.core.ucl.ac.be/wolsey/default.htm.