An Improved Branch and Bound Algorithm for Minimum Concave Cost Network Flow Problems

BRUCE W. LAMAR
Department of Management, University of Canterbury, Christchurch, New Zealand

(Received: 1 July 1991; accepted: October 1992)

Abstract. This paper formulates the minimum concave cost network flow (MCCNF) problem as a mixed integer program and solves this program using a new branch and bound algorithm. The algorithm combines Driebeek's "up and down" penalties with a new technique referred to as the simple bound improvement (SBI) procedure. An efficient numerical method for the SBI procedure is described and computational results are presented which show that the SBI procedure reduces both the in-core storage and the CPU time required to solve the MCCNF problem. In fact, for problems with over 200 binary decision variables, the SBI procedure reduced the in-core storage by more than one-third and the CPU time by more than 40 percent.

Key words. Concave costs, network flows, simple bound improvement.

1. Introduction

This paper examines minimum concave cost network flow (MCCNF) problems; that is, problems wherein the marginal cost of carrying flow on an arc decreases as the volume of flow on that arc increases (see Figure 1). This type of cost function – representing quantity discounting, volume-based price incentives, and other forms of scale economies – is a salient feature of many problems involving the transport of people, commodities, or information [17]. Moreover, facility location, network design, and other types of network flow problems involving fixed charges can also be modelled as MCCNF problems.

It is well-known that the general MCCNF problem is NP-hard [14]. The complexity of the problem arises from the fact that – although a minimum cost solution (if one exists) always occurs at an extreme point of the feasible region [36] – identification of the optimal point requires, in the worst case, a complete enumeration of all extreme points in the feasible region. Thus, except in special cases, exact algorithms for the MCCNF problem run in exponential time.

Very efficient methods for solving network flow problems with *constant* marginal costs have been available since the early 1960's [12]. However, MCCNF problems, being much harder to solve, require specialized solution procedures. Guisewite and Pardalos [18, 19, 20, 21] discuss recent algorithmic developments for MCCNF problems. In general, solution methods for MCCNF problems can be grouped into three categories: (1) heuristic procedures; (2) dynamic programming methods; and (3) branch and bound procedures. Heuristic procedures are applicable to MCCNF problems with an arbitrary network topology, but they do not

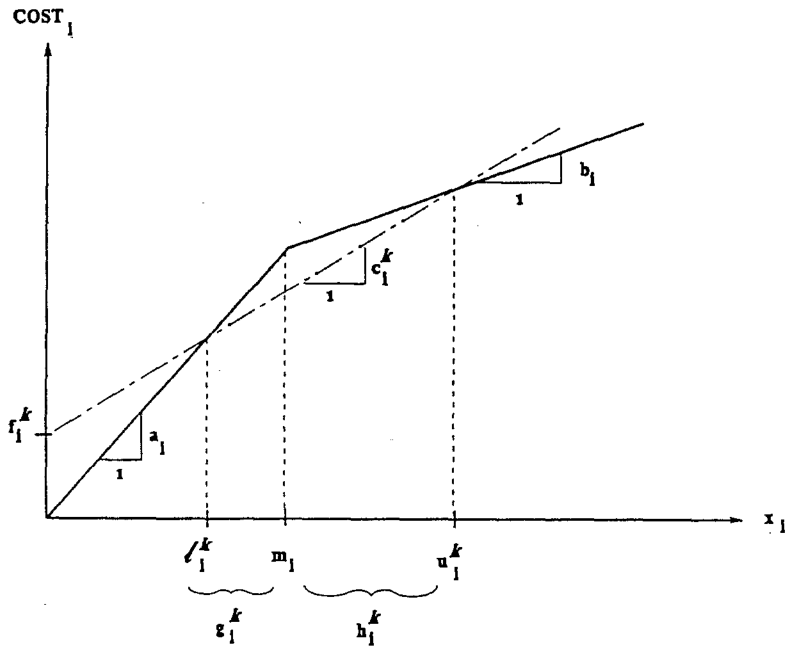


Fig. 1. Typical arc cost function.

provide a measure of quality (i.e., near-optimality) of the solution obtained. Just the reverse is true for dynamic programming approaches. They provide exact results, but are efficient only for applications involving a limited number of supply and/or demand points (e.g., economic lot-sizing problems). Finally, branch and bound procedures have the dual advantage of providing exact results and being applicable to MCCNF problems with arbitrary network topologies. But, because the enumeration tree grows exponentially with problem size, very efficient bounding techniques are required when branch and bound procedures are used to solve MCCNF problems.

This paper presents a new bounding technique, referred to as the "simple bound improvement" (SBI) procedure, for the MCCNF problems comprised of piecewise-linear-concave arc cost functions. The SBI procedure is of theoretical as well as practical interest because it can reduce both the time and the in-core memory required to solve MCCNF problems using a branch and bound procedure. The majority of this paper is devoted to a formal description of the SBI procedure and its role in solving MCCNF problems. The concept underlying this method, however, can be stated very simply. Each arc in a network has a lower (possibly zero) and upper (possibility infinity) bound on the flow that can be carried on that arc. If these bounds can be made tighter, then the feasible solution space of the problem is reduced. The "trick" to the SBI procedure is to tighten these bounds as much as possible while, at the same time, ensuring that these improved bounds do not "cut off" the optimal flow on any of the arcs in the network. By using this method, larger MCCNF problems can be solved more efficiently.

This paper is organized as follows. Section 2 formulates a family of mixed integer programs representing a generic subprogram in a branch and bound enumeration tree. The linear programming relaxation of a subprogram is characterized in Section 3. This section also discusses postoptimality analysis of a subprogram relaxation. Sections 4 through 6 discuss a trilogy of algorithms used to solve the MCCNF problem. Section 4 specializes Driebeek's [9] "up and down" penalties for a generic subprogram of the MCCNF problem. These penalties form part of the SBI procedure presented in Section 5. In this section, conceptual as well as computational aspects of the SBI method are discussed. The SBI procedure, in turn, forms part of the branch and bound algorithm outlined in Section 6. Section 7 presents some computational results demonstrating that the SBI procedure accelerates the branch and bound algorithm. Finally, Section 8 summarizes the paper and suggests several possible extensions of the SBI procedure.

2. Mixed Integer Programming Subprogram

The mixed integer programming formulation developed in this section represents a generic subprogram in a branch and bound procedures that is used to solve the minimum concave cost network flow (MCCNF) problem. However, in contrast to the standard branch and bound procedure (where there is a one-to-one correspondence between subprograms and nodes in an enumeration tree), the branch and bound method presented in this paper associates an entire *family* of subprograms with each node in the enumeration tree. This family of mixed integer programming subprograms is denoted by MIP^k where the superscript k ($k = 0, 1, 2, \dots$) is an index number that identifies an individual subprogram within this family. For each family of subprograms, we refer to MIP^0 as the "initial" subprogram. The initial subprogram at the root node in the enumeration tree is the original problem, MCCNF.

Below, we define notation and formulate a generic subprogram, MIP^k.

To describe the network topology used with each subprogram, let NODE denote the node set (with generic element n) and let ARC denote the arc set (with generic element i). For each $i \in \text{ARC}$, let TAIL_i and HEAD_i denote, respectively, the tail and head nodes for arc i. For each $n \in \text{NODE}$, let LEAVE_n denote the set of arcs whose tail node is n, let ENTER_n denote the set of arcs whose head node is n, and let d_n denote the net demand (i.e., $d_n < 0$) or supply (i.e., $d_n > 0$) at node n. Note that the network topology descriptors defined above are *not* indexed by k.

For each $i \in \text{ARC}$, COST_i denotes the cost function for arc i. We assume that COST_i is a concave nondecreasing function consisting of two piecewise linear segments with marginal costs a_i and b_i, respectively (see Figure 1). [Although the

MCCNF problem formulated in this paper consists of arc cost functions with, at most, two piecewise linear segments, generalization to more than two segments is straightforward.] We assume that $a_i \geqslant b_i \geqslant 0$. In addition, we let m_i denote the flow "breakpoint" at which the marginal cost of COST_i switches from a_i to b_i. We assume that m_i is a nonnegative integer. Note that the cost function parameters given above also are *not* indexed by k.

The lower and upper flow bounds for arc i in program MIP^k, however, *are* indexed by k. These bounds are denoted by l_i^k and u_i^k, respectively. We assume that these bounds are integers and that $0 \leqslant l_i^k \leqslant m_i \leqslant u_i^k$. [Note that this latter assumption is not restrictive because if $l_i^k > m_i$ or $u_i^k < m_i$, then arc i will have a constant marginal cost in program MIP^k so m_i can be reset to l_i^k or u_i^k in COST_i.] It is also useful to define the following two flow interval coefficients (see Figure 1):

$$g_i^k = m_i - l_i^k \quad \forall i \in \text{ARC}\,, \tag{1a}$$

$$h_i^k = u_i^k - m_i \quad \forall i \in \text{ARC}\,. \tag{1b}$$

By construction, g_i^k and h_i^k are nonnegative integers.

For each arc $i \in \text{ARC}$ in program MIP^k, one binary and three continuous decision variables are defined. The binary variable, denoted y_i, specifies whether the applicable marginal cost of function COST_i is a_i (if $y_i = 0$) or b_i (if $y_i = 1$). We also partition the arc set ARC into three disjoint, collectively exhaustive subsets: ZERO, ONE, and FREE. The value of y_i depends, in part, upon which subset arc i is a member. If $i \in \text{ZERO}$, then y_i is fixed at zero; if $i \in \text{ONE}$, then y_i is fixed at one; and if $i \in \text{FREE}$, then y_i may assume either value.

The continuous variables for $i \in \text{ARC}$, denoted x_i, v_i, and w_i, measure the level of flow carried on arc i. Decision variable x_i gives the total flow on arc i. This total flow is the sum of two components: (1) decision variable v_i (representing the flow carried on arc i at a marginal cost of a_i); and (2) decision variable w_i (representing the flow carried on arc i at a marginal cost of b_i). For purposes of formulating the problem, we represent each arc $i \in \text{ARC}$ as a pair of parallel arcs, one for each of the flow components defined above (see Figure 2). Note that $i \in \text{ZERO}$ implies that $x_i \leqslant m_i$, and that $i \in \text{ONE}$ implies that $x_i \geqslant m_i$.

The formulation also uses an arc flow set, denoted by FLOW^k, which represents the set of arc flow *vectors*, $\mathbf{x} = (\ldots, x_i, \ldots)$, that conform to the following capacitated flow balance equations:

$$\sum_{i \in \text{LEAVE}_n} x_i - \sum_{i \in \text{ENTER}_n} x_i = d_n \quad \forall n \in \text{NODE}\,, \tag{2a}$$

$$l_i^k \leqslant x_i \leqslant m_i \quad \forall i \in \text{ZERO}\,, \tag{2b}$$

$$m_i \leqslant x_i \leqslant u_i^k \quad \forall i \in \text{ONE}\,, \tag{2c}$$

$$l_i^k \leqslant x_i \leqslant u_i^k \quad \forall i \in \text{FREE}\,. \tag{2d}$$

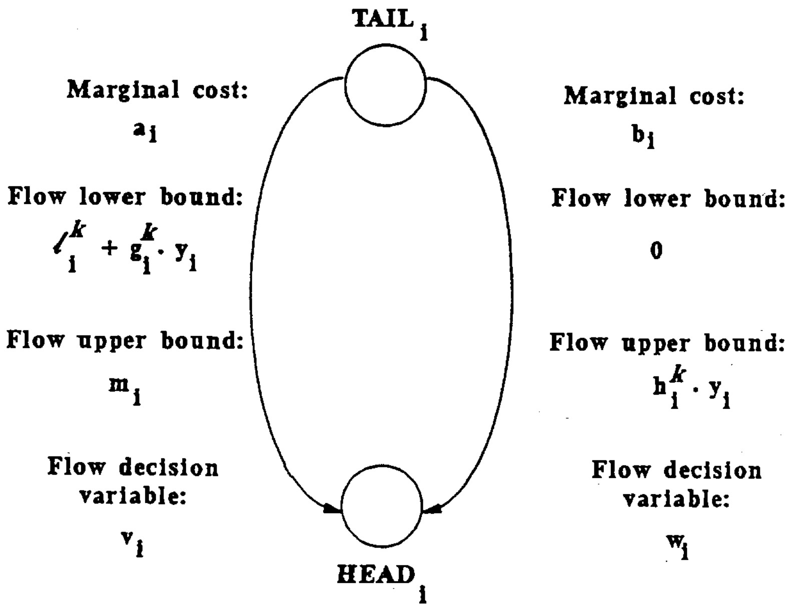

TAIL$_i$
Marginal cost: a_i
Flow lower bound: $l_i^k + g_i^k \cdot y_i$
Flow upper bound: m_i
Flow decision variable: v_i
Marginal cost: b_i
Flow lower bound: 0
Flow upper bound: $h_i^k \cdot y_i$
Flow decision variable: w_i
HEAD$_i$

Fig. 2. Parallel arc representation.

We now formulate MIP^k, a generic subprogram of the minimum concave cost network flow problem considered in this paper.

Program MIP^k:

$$\min_{x \in \mathrm{FLOW}^k} \sum_{i \in \mathrm{ARC}} (a_i \cdot v_i + b_i \cdot w_i) \tag{3a}$$

Subject to:

$$v_i + w_i = x_i \qquad \forall i \in \mathrm{ARC} \tag{3b}$$

$$l_i^k + g_i^k \cdot y_i \leq v_i \leq m_i \qquad \forall i \in \mathrm{ARC} \tag{3c}$$

$$0 \leq w_i \leq h_i^k \cdot y_i \qquad \forall i \in \mathrm{ARC} \tag{3d}$$

$$y_i = 0 \qquad \forall i \in \mathrm{ZERO} \tag{3e}$$

$$y_i = 1 \qquad \forall i \in \mathrm{ONE} \tag{3f}$$

$$y_i \in \{0, 1\} \qquad \forall i \in \mathrm{FREE}\,. \tag{3g}$$

Objective function (3a) minimizes the total cost of carrying flow in the network. Performing this minimization over the set FLOW^k ensures that only feasible flow patterns are considered. Constraint (3b) establishes the conservation of flow for the parallel arc representation for each arc $i \in \mathrm{ARC}$ (see Figure 2). Because COST$_i$ is a concave function, the marginal cost of arc i must be nonincreasing as the flow on arc i increases. This requirement is enforced by constraints (3c) and

(3d), which guarantee that $w_i = 0$ if $v_i < m_i$. Finally, constraints (3e) through (3g) identify the restrictions placed on the binary decision variables in subprogram MIP^k. [The nonnegativity conditions for the continuous decision variables are implicitly enforced by constraints (2b) through (2d), (3c), and (3d).]

For any program P, let FEASIBLE[P], SOLUTION[P], and OBJFCTN[P] denote, respectively, the feasible region, the optimal solution vector, and the optimal objective function value for that program. Thus, for subprogram MIP^k, this information is denoted by FEASIBLE[MIP^k], SOLUTION[MIP^k], and OBJFCTN[MIP^k]. Also, let the optimal value of an individual decision variable be denoted by the symbol for that variable followed by the name of the program (in square brackets) for which that decision variable is optimal. Thus, for each arc i in subprogram MIP^k, the optimal value of the decision variables is denoted by y_i[MIP^k], x_i[MIP^k], v_i[MIP^k], and w_i[MIP^k].

Mixed integer programming formulations used in implicit enumeration procedures for concave cost networks with arbitrary topologies have also been developed by Balakrishnan [2], Florian and Robillard [11], Lamar [24], Lamar and Sheffi [26], and Los and Lardinois [29]. Formulations and branch bound algorithms for fixed charge transportation (i.e., bipartite) network flow problems – originally formulated by Balinski [3] – have been developed by Barr *et al.* [5], Cabot and Erenguc [7], Gray [16], and Kennington and Unger [23]. Florian and Robillard [11] and Malek-Zavarei and Frisch [30] have shown the equivalence between bipartite and arbitrary network topologies for MCCNF problems (also see [10]). Gallo *et al.* [13], and Guisewite and Pardalos [18, 19] considered networks with a single source node; Afentakis *et al.* [1] applied a branch and bound algorithm to a lot sizing problem involving concave costs; and Sa [32] and Soland [33] used branch and bound methods for plant location problems with concave production, holding, and/or transportation costs. See Guisewite and Pardalos [17] for a comprehensive survey.

The structure of the linear programming relaxation of subprogram MIP^k is examined next.

3. Linear Programming Relaxation

It is well known (see, for example, [35]) that a lower bound to OBJFCTN[MIP^k] can be obtained by solving a linear program in which, for each arc $i \in$ ARC, COST$_i$ is replaced with a the linear underestimator of COST$_i$ (see Figure 1). Moreover, because this linear program is a minimum (linear) cost network flow problem, it can be solved very efficiently. The contribution of this section is to show that the linear program described above is equivalent to the linear program formed by relaxing the integrality requirements in subprogram MIP^k, thereby establishing the tightness of the lower bound obtained by using a linear underestimator of a concave cost function. This work extends Balinski's [3] results for fixed charge problems to a more general class of problems.

The material in this section is divided into two parts. The first part shows the equivalence between alternative linear programming representations and the second part discusses postoptimality analysis. This analysis is used later in this paper to develop tight bounds to OBJFCTN[MIP^k].

3.1. EQUIVALENT FORMULATIONS

We begin by formulating the linear programming relaxation of subprogram MIP^k. This relaxation, denoted LP^k, is formed simply by replacing the binary constraints (3g) with the following nonnegativity conditions:

$$0 \leq y_i \leq 1 \quad \forall i \in \text{FREE}\,. \tag{4}$$

Let FEASIBLE[LP^k], SOLUTION[LP^k], and OBJFCTN[LP^k] denote, respectively, the feasible region, optimal solution vector, and the optimal objective function value for program LP^k. Also, for each arc i, let y_i[LP^k], x_i[LP^k], v_i[LP^k], and w_i[LP^k] denote the optimal value of the decision variables in program LP^k.

Relaxation LP^k has the property that, if the total flow on each arc $i \in$ ARC is *given* (i.e., fixed), then program LP^k separates by arc and the optimal value of the other decision variables associated with each arc i can be obtained by inspection. Thus, we can conceptualize a solution procedure for program LP^k as follows: for each feasible flow vector in FLOW^k, solve a separate linear program for each arc in ARC; then select, from among this (possibly infinite) set of flow vectors, the one that minimizes the total cost. As shown below, this solution method is equivalent to solving a linear cost network flow problem in which the cost for arc i is taken as the linear underestimator of COST$_i$.

The concept outlined in the preceding paragraph can be expressed algebraically by representing the linear programming relaxation of subprogram MIP^k in the following equivalent form:

Program LP^k:

$$\underset{x \in \text{FLOW}^k}{\text{Min}} \sum_{i \in \text{ARC}} (b_i \cdot x_i + \text{OBJFCTN}[\text{FIXLP}_i^k])\,. \tag{5}$$

Here, OBJFCTN[FIXLP$_i^k$] is the optimal objective function value of the following linear program for arc i in which x_i (the total flow on arc i) is considered fixed:

Program FIXLP$_i^k$:

$$\text{Min}\,(a_i - b_i) \cdot v_i \tag{6a}$$

Subject to:

$$v_i \leq m_i \tag{6b}$$

$$-g_i^k \cdot y_i + v_i \geq l_i^k \tag{6c}$$

$$h_i^k \cdot y_i + v_i \geq x_i \tag{6d}$$

$$v_i \leqslant x_i \tag{6e}$$

$$y_i \geqslant p_i \tag{6f}$$

$$y_i \leqslant q_i \tag{6g}$$

where p_i and q_i are parameters such that $p_i = 0$ if $i \in \text{ZERO} \cup \text{FREE}$; $p_i = 1$ if $i \in \text{ONE}$; $q_i = 0$ if $i \in \text{ZERO}$; and $q_i = 1$ if $i \in \text{ONE} \cup \text{FREE}$. We refer to program (6) as the "fixed flow" linear program. Note that there is a separate program for each arc $i \in \text{ARC}$ and that x_i is considered a constant in each of these programs. The fixed flow program for arc i is composed of the constraints and objective function terms involving arc i in program LP^k. Constraints (6b) and (6c) correspond to constraint (3c); constraints (6d) and (6e) correspond to constraint (3d); and constraints (6f) and (6g) correspond to constraints (3e) through (3g). Note, however, that decision variable w_i has been eliminated from the fixed flow program by substituting $x_i - v_i$ for w_i (see eq. (3b)). Thus, w_i can also be interpreted as the slack variable associated with constraint (6e). Because w_i has been eliminated and x_i is considered a constant, each fixed flow linear program involves only two decision variables: y_i and v_i.

Program FIXLP$_i^k$ has been expressed in the form given above because its optimal solution can be obtained by inspection. If $i \in \text{ZERO}$, then it is clear that the fixed flow program is feasible only if $l_i^k \leqslant x_i \leqslant m_i$; and if the program is feasible, then the optimal solution is

$$y_i[\text{FIXLP}_i^k] = 0 \tag{7a}$$

$$v_i[\text{FIXLP}_i^k] = x_i \tag{7b}$$

$$w_i[\text{FIXLP}_i^k] = 0 \tag{7c}$$

Similarly, if $i \in \text{ONE}$, then the fixed flow program is feasible only if $m_i \leqslant x_i \leqslant u_i^k$, in which case the optimal solution is

$$y_i[\text{FIXLP}_i^k] = 1 \tag{8a}$$

$$v_i[\text{FIXLP}_i^k] = m_i \tag{8b}$$

$$w_i[\text{FIXLP}_i^k] = x_i - m_i\,. \tag{8c}$$

Finally, if $i \in \text{FREE}$, then the fixed flow program will be feasible only if $l_i^k \leqslant x_i \leqslant u_i^k$. Figure 3 shows a typical feasible region for program FIXLP$_i^k$ for the case where $i \in \text{FREE}$. Observe that, because of the concavity assumption for the cost function COST$_i$, the coefficient $(a_i - b_i)$ in objective function (6a) is always nonnegative. This means that, if program FIXLP$_i^k$ is feasible, then an optimal solution *always* occurs at the extreme point in which constraints (6c) and (6d) are binding (see point 1 in Figure 3). Thus, these two constraints (taken as equalities), together with equality (3b), can be used to determine the optimal value of the decision variables. We consider two possible cases. On the one hand,

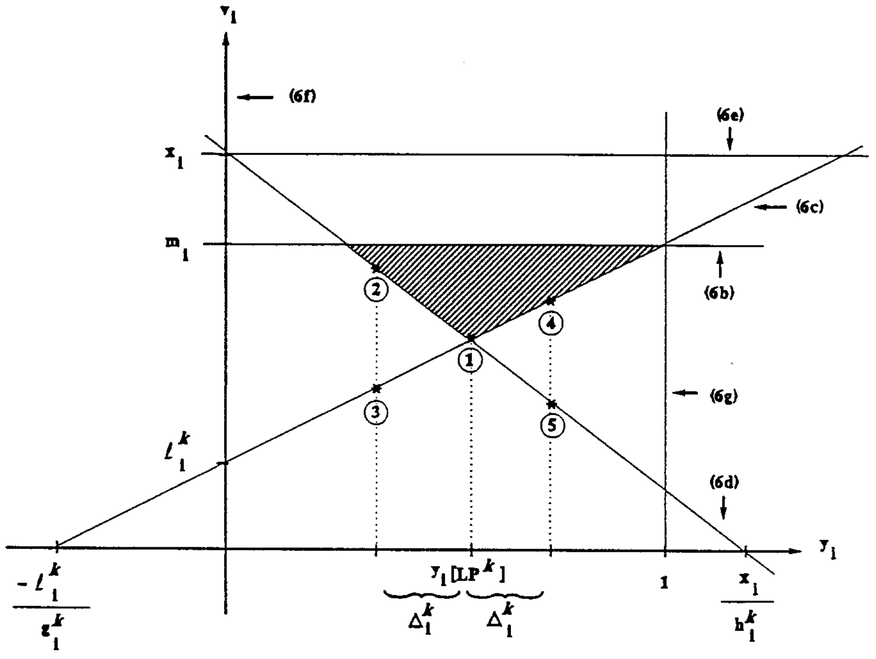


Fig. 3. Typical feasible region for fixed flow program.

if $g_i^k + h_i^k$ equals zero, then the fixed flow program is feasible only if $x_i = m_i$ and the value of y_i is arbitrary. Thus, in this case, the optimal solution to the program can be expressed using either (7) or (8). On the other hand, if $g_i^k + h_i^k$ is nonzero, then we can solve explicitly for y_i, v_i, and w_i. This yields,

$$y_i[\mathrm{FIXLP}_i^k] = \frac{x_i - l_i^k}{g_i^k + h_i^k} \tag{9a}$$

$$v_i[\mathrm{FIXLP}_i^k] = \frac{g_i^k \cdot x_i + h_i^k \cdot l_i^k}{g_i^k + h_i^k} \tag{9b}$$

$$w_i[\mathrm{FIXLP}_i^k] = \frac{h_i^k \cdot (x_i - l_i^k)}{g_i^k + h_i^k} \tag{9c}$$

Observe that, if the RHS of eq. (7b), (8b), or (9b) is substituted for v_i in eq. (6a), then eq. (5) can be expressed solely in terms of the total arc flow decision variables $\{x_i\}$. This substitution can be expressed compactly by defining the following new parameters:

$$c_i^k = \begin{cases} a_i & \text{if } i \in \text{ZERO or } g_i^k + h_i^k = 0 \\ b_i & \text{if } i \in \text{ONE and } g_i^k + h_i^k \neq 0 \\ \dfrac{a_i \cdot g_i^k + b_i \cdot h_i^k}{g_i^k + h_i^k} & \text{if } i \in \text{FREE and } g_i^k + h_i^k \neq 0 \end{cases} \tag{10a}$$

$$f_i^k = \begin{cases} 0 & \text{if } i \in \text{ZERO or } g_i^k + h_i^k = 0 \\ (a_i - b_i) \cdot m_i & \text{if } i \in \text{ONE and } g_i^k + h_i^k \neq 0 \\ (a_i - c_i^k) \cdot l_i^k & \text{if } i \in \text{FREE and } g_i^k + h_i^k \neq 0 . \end{cases} \tag{10b}$$

These two parameters can be interpreted as the slope and intercept, respectively, of the line representing the linear underestimator of function COST_i over the feasible domain of x_i (see Figure 1). Using these parameters, eq. (5) can be reexpressed as the following program:

Program LP^k:

$$\sum_{i\in\text{ARC}} f_i^k + \min_{x\in\text{FLOW}^k} \sum_{i\in\text{ARC}} c_i^k \cdot x_i . \tag{11}$$

Observe that three equivalent forms of the linear programming relaxation of subprogram MIP^k have been presented:

(1) Eq. (3a) through 3f and (4),

(2) Eq. (5) and (6), and

(3) Eq. (11).

The formulation given in eq. (11) is preferred, however, because it is clear from the form of this expression, that the linear programming relaxation of MIP^k is simply a minimum (linear) cost capacitated network flow model, plus an objective function constant term ($\Sigma\ f_i^k$). By using special tree labeling techniques (see, for example, [4]), program LP^k can be solved very efficiently. Moreover, eq. (11) establishes that the lower bound to $\text{OBJFCTN}[\text{MIP}^k]$ obtained by using the linear underestimator of COST_i is as tight as $\text{OBJFCTN}[\text{LP}^k]$.

In subsequent sections of this paper, two procedures are presented for obtaining a tighter lower bound to $\text{OBJFCTN}[\text{MIP}^k]$. Both of these methods are based on an incremental change of flow from the optimal solution of program LP^k. As shown in the next subsection, the effects of these changes can be determined directly from the solution to LP^k.

3.2. POSTOPTIMALITY FLOW ANALYSIS

To evaluate the effect of an incremental change of flow on arc i in program LP^k, we consider a new linear program, denoted POSTOPTLP_i^k, formed by adding the single constraint

$$x_i = x_i[\text{LP}^k] + \delta_i^k \tag{12}$$

to program LP^k. Here, $x_i[\text{LP}^k]$ is the optimal value of decision variable x_i in program LP^k and δ_i^k is a fixed number. Because program POSTOPTLP_i^k is of the form of a network flow program augmented by a single side constraint, it could be evaluated by the methods proposed by Belling-Seib *et al.* [6] or Glover *et al.* [15]. However, assuming no change in basis, the effect of constraint (12) can be determined directly from the optimal solution to program LP^k.

To describe this effect, let α_i^k denote the rate of increase in OBJFCTN

[POSTOPTLP$_i^k$] as δ_i^k changes from 0 to 0^- (i.e., to a small negative value) and let β_i^k denote the rate of increase in OBJFCTN[POSTOPTLP$_i^k$] as δ_i^k changes from 0 to 0^+ (i.e., to a small positive value). [Typical values of these rates of increase (i.e., slopes) are shown in Figure 4.] In addition, let π_n^k denote the optimal value of the dual variable in program LP^k associated with node n in the flow balance equation (2a), and let

$$r_i^k = c_i^k + \pi_{\text{TAIL}_i}^k - \pi_{\text{HEAD}_i}^k \tag{13}$$

denote the reduced cost associated with decision variable x_i in the optimal solution to program LP^k. The effect of constraint (12) depends on whether x_i is a basic or nonbasic decision variable. Thus, let BASIC^k denote the set of arcs $i \in$ ARC such that x_i is basic in the optimal solution in program LP^k and let NONBASIC^k = ARC − BASIC^k.

If $i \in$ BASIC^k, then arc i must be part of a basis-equivalent spanning tree representing the basic solution to program LP^k [22]. If arc i were omitted from this spanning tree, two disjoint subtrees would be created, one containing TAIL$_i$ (the tail node of arc i) and the other containing HEAD$_i$ (the head node of arc i). Let T_i^k and H_i^k denote, respectively, the subtrees containing node TAIL$_i$ and HEAD$_i$. As summarized in Table I, we now define four (disjoint but not collective exhaustive) subsets of the arcs contained in NONBASIC^k. Table I shows, for example, that THL$_i^k$ is the set of arcs $j \in$ NONBASIC^k such that node

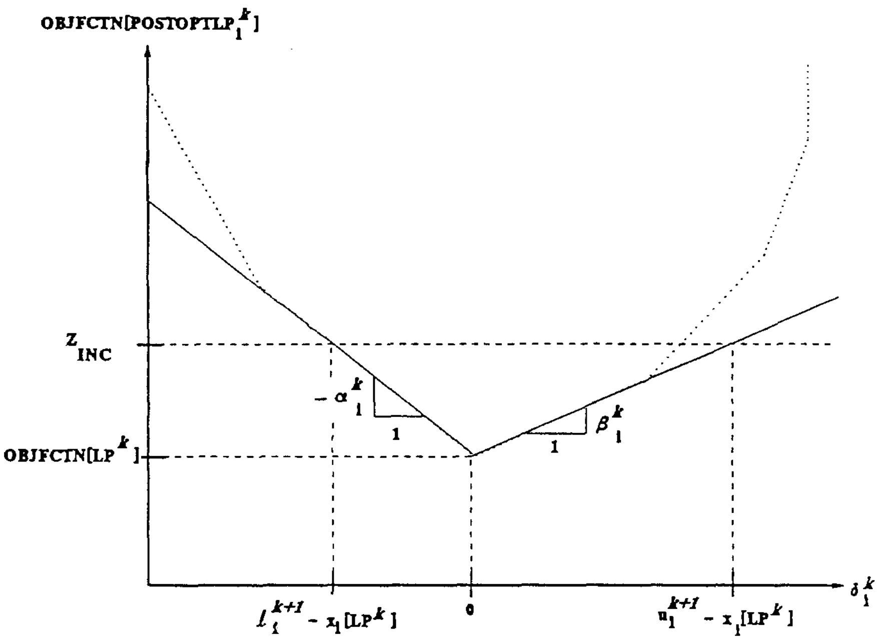


Fig. 4. Typical postoptimal flow analysis for linear relaxation.

Table I. Nonbasic arc subsets in postoptimal flow analysis

Arc subset name	Subtree containing tail node	Subtree containing head node	Arc flow
THL_i^k	T_i^k	H_i^k	lower bound
THU_i^k	T_i^k	H_i^k	upper bound
HTL_i^k	H_i^k	T_i^k	lower bound
HTU_i^k	H_i^k	T_i^k	upper bound

TAIL_j is contained in subtree T_i^k, node HEAD_j is contained in H_i^k, and $x_j[\mathrm{LP}^k]$ (the optimal flow on arc j in program LP^k) is equal to l_j^k.

Assuming no change in basis, observe that if $i \in \mathrm{BASIC}^k$, then a unit *decrease* in the flow on arc i requires either (1) a unit increase in flow on an arc contained in arc subset THL_i^k; or (2) a unit decrease in flow on an arc contained in arc subset HTU_i^k. So, the minimum cost way of forcing this change of flow on arc i is expressed by the minimum (absolute value) reduced cost of the arcs contained in subsets THL_i^k and HTU_i^k.

On the other hand, if $i \in \mathrm{NONBASIC}^k$, then it must be the case that either (1) $x_i[\mathrm{LP}^k] = l_i^k$; or (2) $x_i[\mathrm{LP}^k] = u_i^k$. If $x_i[\mathrm{LP}^k]$ is at its lower bound, then clearly the flow on arc i cannot be decreased; and if $x_i[\mathrm{LP}^k]$ is at its upper bound, then, assuming no change in basis, the minimum cost for a unit decrease in the flow on arc i is simply (the absolute value of) the reduced cost for arc i.

Thus, α_i^k, the rate of increase in $\mathrm{OBJFCTN}[\mathrm{POSTOPTLP}_i^k]$ as δ_i^k (in constraint (12)) changes from 0 to 0^-, is given by

$$\alpha_i^k = \begin{cases} \operatorname{Min}_j\{|r_j^k| : j \in \mathrm{THL}_i^k \cup \mathrm{HTU}_i^k\} & \text{if } i \in \mathrm{BASIC}^k \\ \infty \quad \text{if } i \in \mathrm{NONBASIC}^k \text{ and } x_i[\mathrm{LP}^k] = l_i^k \\ |r_i^k| \quad \text{if } i \in \mathrm{NONBASIC}^k \text{ and } x_i[\mathrm{LP}^k] = u_i^k\,. \end{cases} \tag{14}$$

Similarly, if $i \in \mathrm{BASIC}^k$ and no basis change occurs, then *increasing* the flow on arc i by one unit requires either (1) a unit decrease in flow on an arc contained in THU_i^k; or (2) a unit increase in flow on an arc contained in HTL_i^k; and the cost associated with this flow change is given by the reduced cost of the arcs in these two subsets. On the other hand, if $i \in \mathrm{NONBASIC}^k$, then the cost associated with a unit increase in the flow on arc i is the reduced cost of arc i if $x_i[\mathrm{LP}^k] = l_i^k$; and infinity if $x_i[\mathrm{LP}^k] = u_i^k$. Thus, β_i^k, the rate of increase in OBJFCTN [$\mathrm{POSTOPTLP}_i^k$] as δ_i^k changes from 0 to 0^+ is given by

$$\beta_i^k = \begin{cases} \operatorname{Min}_j\{|r_j^k| : j \in \mathrm{THU}_i^k \cup \mathrm{HTL}_i^k\} & \text{if } i \in \mathrm{BASIC}^k \\ r_i^k \quad \text{if } i \in \mathrm{NONBASIC}^k \text{ and } x_i[\mathrm{LP}^k] = l_i^k \\ \infty \quad \text{if } i \in \mathrm{NONBASIC}^k \text{ and } x_i[\mathrm{LP}^k] = u_i^k\,. \end{cases} \tag{15}$$

Note that, once program LP^k has been solved, then the rates α_i^k and β_i^k can be determined with very little additional computational effort. The next two sections discuss procedures that use these rates to obtain a lower bound to OBJFCTN[MIP^k] that can be significantly tighter than OBJFCTN[LP^k].

4. Up and Down Penalties

This section specializes, for program MIP^k, the "up and down" penalties developed by Driebeek [9]. This well-known procedure is briefly recapitulated here because it is incorporated into the stronger bounds developed in the next section.

The concept of Driebeek's penalty procedure applied to program MIP^k is as follows: if, for any arc $i \in$ FREE, the optimal value of decision variable, y_i, is frictional in program LP^k, then the value of OBJFCTN[LP^k] might be increased (i.e., "penalized") by forcing y_i to be zero or one; and this penalized objective function value will be a lower bound to OBJFCTN [MIP^k] that is at least as tight as OBJFCTN[LP^k]. Note that, although it was intended only for the initial subprogram MIP^0, Driebeek's penalty method can be applied to *any* subprogram MIP^k for $k = 0, 1, 2, \ldots$.

To analyze the penalty associated with an arc $i \in$ FREE whose decision variable y_i is fractional-valued in the optimal solution to LP^k, consider a new linear program, denoted PENLP$_i^k$, formed by adding the single constraint

$$y_i = y_i[\text{LP}^k] + \Delta_i^k \tag{16}$$

to program LP^k. In this constraint, $y_i[\text{LP}^k]$ is the optimal value of y_i in program LP^k and Δ_i^k is a fixed number. If Δ_i^k changes from 0 to 0^- (i.e., to a small negative value), then one of two possible changes to the solution of program LP^k will occur: (1) the flow on arc i will remain at $x_i[\text{LP}^k]$; or (2) the flow on arc i will change (i.e., decrease) by $\Delta_i^k \cdot (g_i^k + h_i^k)$. In the first case, the optimal solution of program PENLP$_i^k$ will shift from point 1 to point 2 in Figure 3 and the optimal objective function value of program PENLP$_i^k$ will change from OBJFCTN[LP^k] to OBJFCTN[LP^k] $- \Delta_i^k \cdot (g_i^k + h_i^k) \cdot (a_i - c_i^k)$. In the second case, the optimal solution of program PENLP$_i^k$ will shift from point 1 to point 3 in Figure 3 (with a corresponding shift in constraint (6d)) and the optimal objective function value will change from OBJFCTN[LP^k] to OBJFCTN[LP^k] $- \Delta_i^k \cdot (g_i^k + h_i^k) \cdot \alpha_i^k$ (where α_i^k is the marginal rate of change defined in eq. (14)). Thus, the penalty, denoted DOWN$_i^k$, for forcing y_i down to zero must be at least as great as

$$\text{DOWN}_i^k = y_i[\text{LP}^k] \cdot (g_i^k + h_i^k) \cdot \text{Min}\{a_i - c_i^k, \alpha_i^k\} \,. \tag{17}$$

Similarly, if Δ_i^k changes from 0 to 0^+ (i.e., to a small positive value), then either (1) the flow on arc i will remain at $x_i[\text{LP}^k]$; or (2) the flow will change (i.e., increase) by $\Delta_i^k \cdot (g_i^k + h_i^k)$. In the first case, the optimal solution of program PENLP$_i^k$ will shift from point 1 to point 4 in Figure 3 and the optimal objective function value of program PENLP$_i^k$ will change from OBJFCTN[LP^k] to OBJFCTN[LP^k] $+ \Delta_i^k \cdot (g_i^k + h_i^k) \cdot (c_i^k - b_i)$. In the second case, the optimal solution of program PENLP$_i^k$ will shift from point 1 to point 5 (with a corresponding shift in constraint (6d)) and the optimal objective function value will change from OBJFCTN[LP^k] to OBJFCTN[LP^k] + $\Delta_i^k \cdot (g_i^k + h_i^k) \cdot \beta_i^k$ (where β_i^k is the marginal rate of change defined in eq. (15)). Thus, the penalty, denoted UP_i^k, for forcing y_i up to one must be at least as great as

$$\mathrm{UP}_i^k = (1 - y_i[\mathrm{LP}^k]) \cdot (g_i^k + h_i^k) \cdot \mathrm{Min}\{c_i^k - b_i, \beta_i^k\} . \tag{18}$$

Because, in the optimal solution to MIP^k, every y_i must be either 0 or 1, the "penalty" lower bound, denoted z_{PEN}^k, to OBJFCTN[MIP^k] is given by

$$z_{\mathrm{PEN}}^k = \mathrm{OBJFCTN}[\mathrm{LP}^k] + \underset{i \in \mathrm{FREE}}{\mathrm{Max}} \{\mathrm{Min}\{\mathrm{DOWN}_i^k, \mathrm{UP}_i^k\}\} . \tag{19}$$

The next section describes a procedure, which can be used in conjunction with the penalty method described above, to obtain a tighter lower bound.

5. Simple Bound Improvement Procedure

In contrast to the "up and down" penalty procedure summarized in Section 4 (which concentrated on the binary decision variables $\{y_i\}$), the "simple bound improvement" (SBI) method presented here focuses on the continuous variables $\{x_i\}$. In this section, we generalize Lamar and Sheffi's [26] and Lamar *et al.* [27] work on fixed charge problems. We also introduce a simplified computational procedure.

The discussion below is divided into two parts. The first part develops the concept of the SBI procedure for MCCNF problems; and the second part shows that the parameters used in this procedure are easy to compute.

5.1. CONCEPT

Starting with the solution LP^0 (the relaxation of the initial subprogram), the SBI procedure evaluates the family of linear programming relaxations LP^k for $k = 1, 2, \ldots$ to obtain a successively tighter lower bound to OBJFCTN[MIP^0] (the optimal objective function value of the initial subprogram). The process uses z_{INC}, the objective function value of an incumbent (i.e., feasible but not necessarily optimal) solution to the original problem, MCCNF. [The determination of z_{INC} is discussed in Section 6.] Regardless of the value of z_{INC}, though, it must be the case that either (1) $z_{\mathrm{INC}} > \mathrm{OBJFCTN}[\mathrm{MIP}^0]$; or (2) $z_{\mathrm{INC}} \leq \mathrm{OBJFCTN}[\mathrm{MIP}^0]$. Below, we consider each of these cases separately and then summarize the lower bound implied by these two cases.

- *Case 1: Incumbent Value Overestimates OBJFCTN*[*MIP^0*]

For the first case we assume that $z_{\mathrm{INC}} > \mathrm{OBJFCTN}[\mathrm{MIP}^0]$. For this case, we also assume that SOLUTION[MIP^0] (the optimal solution to the initial subprogram MIP^0) is contained in FEASIBLE[LP^k] (the feasible region of relaxation LP^k).

[As explained at the end of Case 1, this assumption will always be true if $z_{\mathrm{INC}} > \mathrm{OBJFCTN}[\mathrm{MIP}^0]$.] The incumbent objective function value can be used to seek tighter bounds for the flow on any arc $i \in \mathrm{ARC}$. Let l_i^{k+1} and u_i^{k+1} denote, respectively, these tighter lower and upper flow bounds for arc i. In order to determine the value of these tighter bounds, we once again consider the previously defined linear program, $\mathrm{POSTOPTLP}_i^k$, formed by adding constraint (12) to program LP^k. [In this subsection, however, we consider all possible values of δ_i^k rather than just those that are close to zero.]

Starting with $\delta_i^k = 0$, we first consider the effect of *decreasing* the value of δ_i^k. From parametric RHS analysis, we know that if $\delta_i^k = 0$ then OBJFCTN $[\mathrm{POSTOPTLP}_i^k] = \mathrm{OBJFCTN}[\mathrm{LP}^k]$; and that if $\delta_i^k < 0$, then OBJFCTN $[\mathrm{POSTOPTLP}_i^k] \geqslant \mathrm{OBJFCTN}[\mathrm{LP}^k]$. We continue decreasing the value of δ_i^k until either (1) $\delta_i^k = l_i^k - x_i[\mathrm{LP}^k]$; or (2) $\mathrm{OBJFCTN}[\mathrm{POSTOPTLP}_i^k] = z_{\mathrm{INC}}$. We then set $l_i^{k+1} = x_i[\mathrm{LP}^k] + \delta_i^k$. Observe that if $\delta_i^k = l_i^k - x_i[\mathrm{LP}^k]$, then l_i^{k+1} is simply l_i^k and since (by assumption) $x_i[\mathrm{MIP}^0] \geqslant l_i^k$, this means that $x_i[\mathrm{MIP}^0] \geqslant l_i^{k+1}$. On the other hand, if $\mathrm{OBJFCTN}[\mathrm{POSTOPTLP}_i^k] = z_{\mathrm{INC}}$ (and, by assumption, $z_{\mathrm{INC}} > \mathrm{OBJFCTN}[\mathrm{MIP}^0]$), then $x_i[\mathrm{MIP}^0]$ cannot be less than $x_i[\mathrm{LP}^k] + \delta_i^k$, so it must be true that $x_i[\mathrm{MIP}^0] \geqslant l_i^{k+1}$. Thus, in either case, l_i^{k+1} is a lower bound to $x_i[\mathrm{MIP}^0]$.

Once again starting with $\delta_i^k = 0$, we now consider the effect of *increasing* the value of δ_i^k. Clearly, if $\delta_i^k > 0$, then $\mathrm{OBJFCTN}[\mathrm{POSTOPTLP}_i^k] \geqslant \mathrm{OBJFCTN}[\mathrm{LP}^k]$. We continue increasing δ_i^k until either (1) $\delta_i^k = u_i^k - x_i[\mathrm{LP}^k]$; or (2) $\mathrm{OBJFCTN}[\mathrm{POSTOPTLP}_i^k] = z_{\mathrm{INC}}$. We then set $u_i^{k+1} = x_i[\mathrm{LP}^k] + \delta_i^k$. Observe that, if $\delta_i^k = u_i^k - x_i[\mathrm{LP}^k]$, then $u_i^{k+1} = u_i^k$, so $x_i[\mathrm{MIP}^0] \leqslant u_i^{k+1}$. Moreover, if $\mathrm{OBJFCTN}[\mathrm{POSTOPTLP}_i^k] = z_{\mathrm{INC}}$, then once again it must be true that $x_i[\mathrm{MIP}^0] \leqslant u_i^{k+1}$. Thus, in either case, u_i^{k+1} is an upper bound to $x_i[\mathrm{MIP}^0]$.

By performing the analysis outlined in the preceding paragraphs for each arc $i \in \mathrm{ARC}$, we obtain a set of tighter flow bounds $\{l_i^{k+1}\}$ and $\{u_i^{k+1}\}$. Using these bounds, we then solve program LP^{k+1}.

As mentioned at the beginning of Case 1, we assume that SOLUTION $[\mathrm{MIP}^0] \in \mathrm{FEASIBLE}[\mathrm{LP}^k]$. This will certainly be true for $k = 0$ because LP^0 is the linear programming relaxation of MIP^0. Moreover, we have shown above that if $z_{\mathrm{INC}} > \mathrm{OBJFCTN}[\mathrm{MIP}^0]$ and $l_i^k \leqslant x_i[\mathrm{MIP}^0] \leqslant u_i^k$ for all i, then it must be true that $l_i^{k+1} \leqslant x_i[\mathrm{MIP}^0] \leqslant u_i^{k+1}$ for all i. Thus, by mathematical induction, if $z_{\mathrm{INC}} > \mathrm{OBJFCTN}[\mathrm{MIP}^0]$, then it must be true that $\mathrm{SOLUTION}[\mathrm{MIP}^0] \in \mathrm{FEASIBLE}[\mathrm{LP}^k]$ for *all k*.

Driebeek's penalty lower bound can also be used in conjunction with the SBI procedure. Using eq. (19), we set $z_{\mathrm{PEN}}^{k+1} \leftarrow \mathrm{Max}\{z_{\mathrm{PEN}}^{k+1}, z_{\mathrm{PEN}}^k\}$. Observe that $\mathrm{FEASIBLE}[\mathrm{LP}^{k+1}] \subseteq \mathrm{FEASIBLE}[\mathrm{LP}^k]$ which, in turn, implies that $\mathrm{OBJFCTN}[\mathrm{LP}^{k+1}] \geqslant \mathrm{OBJFCTN}[\mathrm{LP}^k]$. Thus, in many cases, z_{PEN}^{k+1} will be strictly greater than z_{PEN}^k. Furthermore, because $\mathrm{SOLN}[\mathrm{MIP}^0] \in \mathrm{FEASIBLE}[\mathrm{LP}^k]$ and $\mathrm{SOLUTION}[\mathrm{MIP}^0] \in \mathrm{FEASIBLE}[\mathrm{LP}^{k+1}]$, it must be true that SOLUTION $[\mathrm{MIP}^0] \equiv \mathrm{SOLUTION}[\mathrm{MIP}^k] \equiv \mathrm{SOLUTION}[\mathrm{MIP}^{k+1}]$. So, not only is the initial penalty bound, z_{PEN}^0, a lower bound to $\mathrm{OBJFCTN}[\mathrm{MIP}^0]$, but so are z_{PEN}^k and

z_{PEN}^{k+1}. Thus, if $z_{\text{INC}} > \text{OBJFCTN}[\text{MIP}^0]$, then the following relationships hold:

$$z_{\text{PEN}}^{0} \leq z_{\text{PEN}}^{k} \leq z_{\text{PEN}}^{k+1} \leq \text{OBJFCTN}[\text{MIP}^0] \,. \tag{20}$$

This completes the discussion of the first case in which it is assumed that $z_{\text{INC}} > \text{OBJFCTN}[\text{MIP}^0]$.

- *Case 2: Incumbent Value Does Not Overestimate OBJFCTN*[*MIP*^{0}]

For completeness, we now consider the second case in which it is assumed that $z_{\text{INC}} \leq \text{OBJFCTN}[\text{MIP}^0]$. In this case, we simply note that z_{INC} itself is a lower bound to $\text{OBJFCTN}[\text{MIP}^0]$.

- *Lower Bound*

The two cases given above can be combined to develop a lower bound to $\text{OBJFCTN}[\text{MIP}^0]$. Observe that, for *any* incumbent objective function value, if z_{INC} itself is not a lower bound to $\text{OBJFCTN}[\text{MIP}^0]$, then z_{PEN}^{k+1} must be. Thus, we define a new lower bound, referred to as the "simple bound improvement" lower bound and denoted as z_{SBI}^{k+1}, to $\text{OBJFCTN}[\text{MIP}^0]$ as follows:

$$z_{\text{SBI}}^{k+1} = \text{Min}\{z_{\text{PEN}}^{k+1}, z_{\text{INC}}\} \,. \tag{21}$$

Note that, because of relationships (20) if z_{INC} is greater than z_{PEN}^{0}, then the SBI procedure produces a stronger lower bound to the optimal objective function value of the initial subprogram than Driebeek's penalty method used alone (i.e., z_{SBI}^{k+1} is tighter than z_{PEN}^{0}). For the SBI procedure to be useful, however, the lower and upper flow bounds, l_i^{k+1} and u_i^{k+1}, must be easy to compute. This result is shown next.

5.2. COMPUTATION OF IMPROVED FLOW BOUNDS

An important step in the SBI procedure is the efficient computation of the improved lower and upper flow bounds, $\{l_i^{k+1}\}$ and $\{u_i^{k+1}\}$. Note that although tighter flow bounds can be sought for any arc $i \in \text{ARC}$, the only arcs for which the linear underestimator strictly underestimates the concave cost function, COST_i, are the arcs contained in set IMPROVE^k, where

$$\text{IMPROVE}^k = \{i : i \in \text{FREE and } l_i^k < m_i < u_i^k\} \,. \tag{22}$$

Thus, if $i \notin \text{IMPROVE}^k$, then l_i^{k+1} and u_i^{k+1} can simply be set to l_i^k and u_i^k, respectively.

On the other hand, if arc $i \in \text{IMPROVE}^k$, then, as discussed in Subsection 5.1, the determination of l_i^{k+1} and u_i^{k+1} requires a RHS parametric analysis of eq. (12) in program POSTOPTLP_i^k. For instance, the dotted line in Figure 4 shows a typical change in $\text{OBJFCTN}[\text{POSTOPTLP}_i^k]$ as the parameter δ_i^k is varied. A complete parametric analysis of this constraint would, of course, be computationally burdensome. But, as explained in Subsection 3.2, α_i^k and β_i^k, the rate of change in OBJFCTN[POSTOPT$_i^k$] for an incremental change in δ_i^k, can be determined directly from the solution to LP^k (see eq. (14) and (15)). Thus, an underestimator of l_i^{k+1} and an overestimator of u_i^{k+1} can be obtained with very little additional computational effort.

Specifically, if $\alpha_i^k = 0$ then $l_i^{k+1} = l_i^k$, otherwise, as indicated in Figure 4,

$$l_i^{k+1} = \mathrm{Max}\left\{ l_i^k, x_i[\mathrm{LP}^k] - \left\lfloor \frac{z_{\mathrm{INC}} - \mathrm{OBJFCTN}[\mathrm{LP}^k]}{\alpha_i^k} \right\rfloor \right\} \tag{23}$$

where "$\lfloor e \rfloor$" denotes the "floor" function for any expression e (i.e., the largest integer less than or equal to e). Moreover, if $\beta_i^k = 0$, then $u_i^{k+1} = u_i^k$, otherwise

$$u_i^{k+1} = \mathrm{Min}\left\{ u_i^k, x_i[\mathrm{LP}^k] + \left\lfloor \frac{z_{\mathrm{INC}} - \mathrm{OBJFCTN}[\mathrm{LP}^k]}{\beta_i^k} \right\rfloor \right\}. \tag{24}$$

We assume, in eq. (23) and (24), that $z_{\mathrm{INC}} \geqslant \mathrm{OBJFCTN}[\mathrm{LP}^k]$. This is because, if $z_{\mathrm{INC}} < \mathrm{OBJFCTN}[\mathrm{LP}^k]$, then SOLUTION[MCCNF] cannot be contained in FEASIBLE[MIP^k] and hence there is no need to evaluate program MIP^k any further.

The tighter bounds developed in this section form an integral part of the branch and bound procedure outlined next.

6. Branch and Bound Procedure

The branch and bound procedure described in this section solves program MCCNF (or determines that the problem is infeasible). The distinguishing feature between the standard branch and bound method (see, for example, [31] – denoted BBSTANDARD – and the one presented in this section – denoted BBSBI – is the incorporation of the SBI procedure to generate a tighter bound to the current subprogram.

The following paragraphs comment on each of the steps in the branch and bound flowchart shown in Figure 5. [The actual implementations of BBSTANDARD and BBSBI (which are compared empirically in Section 7) are also discussed.]

Step 0 initializes the branch and bound algorithm. Here, the initial subprogram MIP^0 at the root node in the enumeration tree is taken as the original problem, MCCNF, in which all arcs $i \in$ ARC are members of FREE; and sets ZERO and ONE are empty. Step 0 places program MCCNF in the "candidate list", denoted CAND. This list contains the subprograms that are evaluated in the branch and bound procedure. As mentioned in Section 5, we let z_{INC} denote the objective function value of the current incumbent. Step 0 sets z_{INC} to infinity.

Steps 1 and 2 review the subprograms in CAND. If this list is empty, then the

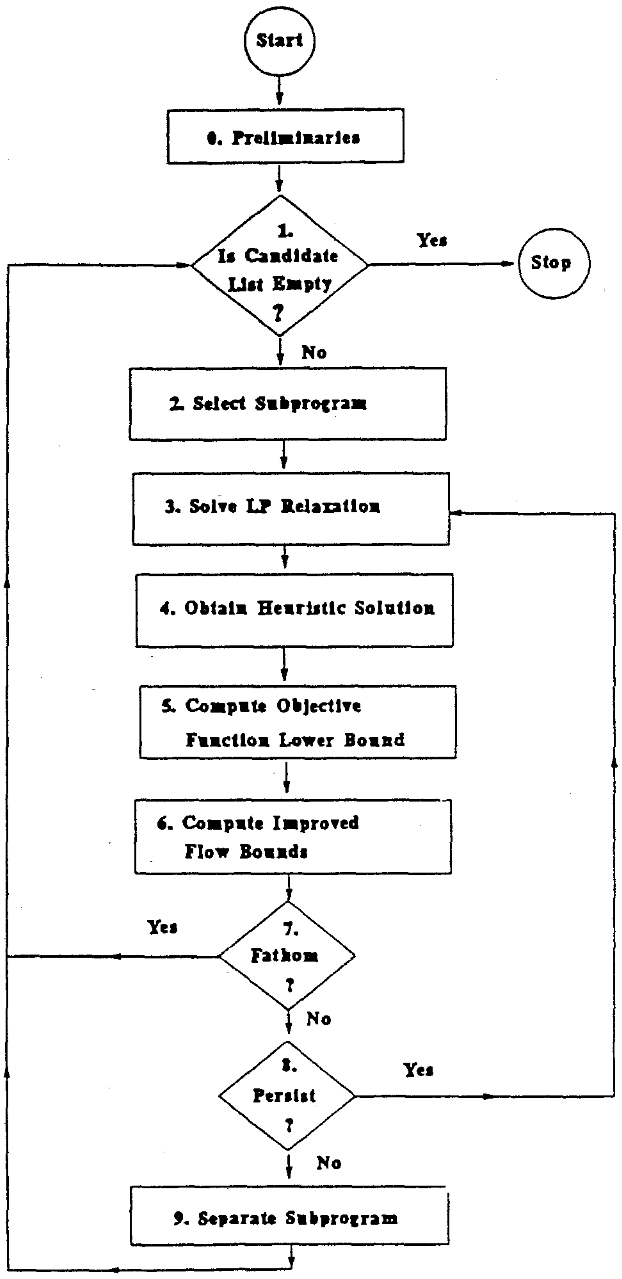

Start
0. Preliminaries
1. Is Candidate List Empty ?
Yes
Stop
No
2. Select Subprogram
3. Solve LP Relaxation
4. Obtain Heuristic Solution
5. Compute Objective Function Lower Bound
6. Compute Improved Flow Bounds
Yes
7. Fathom ?
No
8. Persist ?
Yes
No
9. Separate Subprogram

Fig. 5. Flowchart for branch and bound procedure.

branch and bound algorithm terminates and the current incumbent is the optimal solution to MCCNF. [If CAND is empty and there is no incumbent, then MCCNF is infeasible.] If CAND is nonempty, then a subprogram is selected to be the current initial subprogram, MIP^0. [In the computational tests conducted in Section 7, both BBSTANDARD and BBSBI use a LIFO (rather than a priority) subprogram selection rule because a LIFO selection rule minimizes the incore storage requirements for CAND. For BBSBI, if "backtracking" occurs in the enumeration tree in order to select the current subprogram, then the values of the initial lower and upper flow bounds, $\{l_i^0\}$ and $\{u_i^0\}$, for arcs $i \in \text{FREE}$ are taken from the subprogram at the root node in the enumeration tree; otherwise these bounds are taken from the final iteration of the previously evaluated subprogram (i.e., from the "parent" subprogram). In this way, BBSBI requires no additional storage for the improved flow bound parameters.]

In preparation for the SBI procedure, step 2 also sets the iteration index, k, to zero. [In Section 7, BBSTANDARD keeps k permanently set at zero.]

Step 3 first computes the flow interval parameters, $\{g_i^k\}$ and $\{h_i^k\}$, using eq. (1) and the objective function parameters, $\{c_i^k\}$ and $\{f_i^k\}$, using eq. (10). This step then solves LP^k, the linear programming relaxation of the current subprogram. Because this relaxation is a minimum (linear) cost network flow problem (see Section 3), program LP^k can be solved very efficiently. (See [4] for a discussion of solution methods for this class of problems.)

Step 4 seeks to find a new incumbent solution to MCCNF by obtaining a heuristic solution (see, for example, [35]) to the current subprogram, MIP^0. [In Section 7, both BBSTANDARD and BBSBI obtain a heuristic solution to MIP^0 with very little computational effort by simply setting y_i to zero (respectively, one) for any arc i such that $x_i[\text{LP}^k]$ is less than or equal to (respectively, greater than) m_i: This "rounding" technique always produces a feasible solution to MIP^0, and hence, a feasible solution to MCCNF]. If the objective function value of the heuristic solution obtained in step 4 is less than z_{INC} (the objective function value of the current incumbent), then this heuristic solution is retained as the incumbent solution and the value of z_{INC} is updated.

Step 5 computes a lower bound to $\text{OBJFCTN}[\text{MIP}^0]$ using the penalty and SBI procedures described in Sections 4 and 5. Here, the penalties, DOWN_i^k and UP_i^k, defined in eq. (17) and (18), are used to compute Driebeek's penalty lower bound, z_{PEN}^k, defined in eq. (19). Then, the SBI lower bound, z_{SBI}^k, defined in eq. (21), is calculated. [In Section 7, BBSTANDARD does not compute z_{SBI}^k.]

Step 6 computes the tighter lower and upper flow bounds, $\{l_i^{k+1}\}$ and $\{u_i^{k+1}\}$. If $i \notin \text{IMPROVE}^k$ (see eq. (22)), then step 6 sets $l_i^{k+1} \leftarrow l_i^k$ and $u_i^{k+1} \leftarrow u_i^k$. Otherwise, if $i \in \text{IMPROVE}^k$, then this step uses eq. (23) and (24) to compute the improved flow bounds. [In Section 7, BBSTANDARD omits this step.]

Step 7 tests whether or not the SBI lower bound – computed in step 5 – equals the incumbent objective function value, z_{INC}. If $z_{\text{SBI}}^k = z_{\text{INC}}$, then this means that $z_{\text{PEN}}^k \geq z_{\text{INC}}$ (see eq. (21)), so program MIP^k cannot contain a feasible solution to MCCNF that is better than the current incumbent. In other words, program MIP^k can be "fathomed". Therefore, if $z_{\text{SBI}}^k = z_{\text{INC}}$, then the branch and bound algorithm goes to step 1 to review the subprograms contained in CAND; otherwise the algorithm goes to step 8. [In Section 7, BBSTANDARD tests whether or not $z_{\text{PEN}}^0 \geq z_{\text{INC}}$ in this step.]

Step 8 tests whether or not additional effort should be expended on determining an improved lower bound for the current subprogram. Note that if there is an arc $i \in \text{IMPROVE}^k$ such that (1) $l_i^k < x_i^k < u_i^k$; (2) $l_i^{k+1} > l_i^k$ or $u_i^{k+1} < u_i^k$; and (3) $\alpha_i^k > 0$ and $\beta_i^k > 0$; then OBJFCTN[LP^{k+1}] will be strictly greater than OBJFCTN[LP^k]. Thus, if there is at least one arc that meets the conditions given above, then the iteration index, k, is incremented by one (i.e., $k \leftarrow k + 1$) and the algorithm goes to step 3 to resolve the relaxation of the current subprogram. [Note that SOLUTION[LP^k] $\in$ FEASIBLE[LP^{k+1}]. Thus SOLUTION[LP^k] can be used as an initial basic feasible solution in program LP^{k+1}. In many cases SOLUTION[LP^k] will be optimal or near-optimal in program LP^{k+1}, so program LP^{k+1} can be solved with very little additional computational effort.] On the other hand, if there are no arcs that satisfy the conditions given above, then the algorithm goes to step 9 to separate the current subprogram. [In Section 7, BBSTANDARD omits this step and proceeds to step 9.]

Finally, step 9 selects, from among the elements in arc set FREE, a "branching arc", denote $\hat{\imath}$. [In Section 7, both BBSTANDARD and BBSBI select, as arc $\hat{\imath}$, the arc with the maximum of $\min\{\text{DOWN}_i^k, \text{UP}_i^k\}$ (see eq. (17) and (18)).] Step 9 adds to new subprograms – one in which arc $\hat{\imath}$ is removed from FREE and added to ZERO, and the other in which arc $\hat{\imath}$ is removed from FREE and added to ONE – to the candidate list, CAND. [In Section 7, the subprogram associated with maximum up and down penalty for arc $\hat{\imath}$ is referred to as the "twin" problem. The twin problem is added first to CAND in order to seek "good" heuristic solutions with the LIFO subprogram selection rule [28]. In addition, z_{TWIN}^k, a lower bound to the optimal objective function of the twin problem, is also stored in CAND. Here, z_{TWIN}^k is given by

$$z_{\text{TWIN}}^k = \text{OBJFCTN}[\text{LP}^k] + \text{Max}\{\text{DOWN}_{\hat{\imath}}^k, \text{UP}_{\hat{\imath}}^k\} . \tag{25}$$

If, when the twin problem is selected from CAND, z_{INC} is less than or equal to z_{TWIN}^k, then the twin problem can be fathomed without any further evaluation of that problem.] After completing step 9, the branch and bound algorithm goes to step 1 to review the candidate list.

The next section illustrates the use of the branch and bound algorithm described above.

7. Computational Performance

In this section we demonstrate empirically that, by incorporating the SBI procedure into a conventional branch and bound algorithm, both solution time *and* in-core memory requirements can be reduced. We do this by solving a series of MCCNF problems. As in Section 6, we let BBSBI and BBSTANDARD denote, respectively, the branch and bound procedure with and without the SBI procedure. Both algorithms were programmed in Microsoft Fortran version 4.1 and run on a Micro Source International microcomputer (comparable to an IBM-AT). Solution time was measured by the total CPU-time exclusive of I/O operations;

and in-core storage was measured by the maximum depth of the branch and bound enumeration tree.

The material below is divided into two subsections. The first subsection uses a simple example to illustrate the effect that the SBI procedure has on computational performance; and the second subsection reports the computational results for a series of randomly generated test problems.

7.1. EXAMPLE

The four node, five arc MCCNF problem depicted in Figure 6 and Table II is taken from Florian and Robillard [11]. To illustrate the effect of the SBI procedure, we first solved this simple problem using BBSTANDARD, then resolved it using BBSBI.

The enumeration tree associated with BBSTANDARD for this problem is shown in Figure 7a. The node numbers in the tree indicate the order in which the subprograms were solved. At node 1 the linear programming relaxation of the original mixed integer program, MCCNF, was solved using a network simplex algorithm. The solution to this relaxation was 3 units of flow on arc (1, 2); 3 on

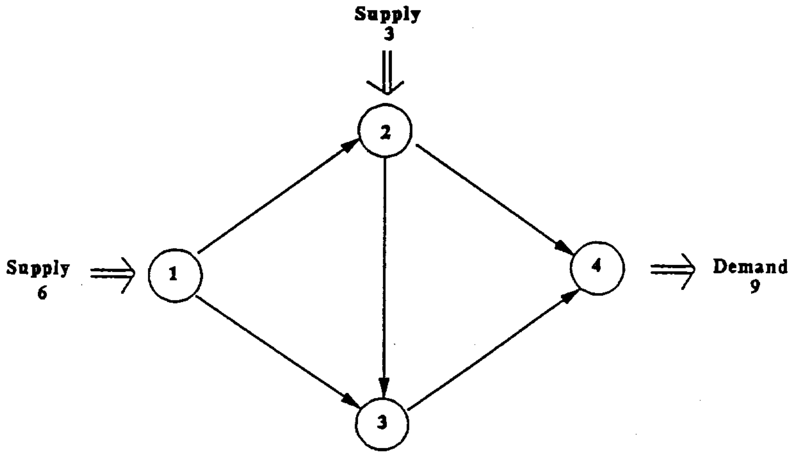
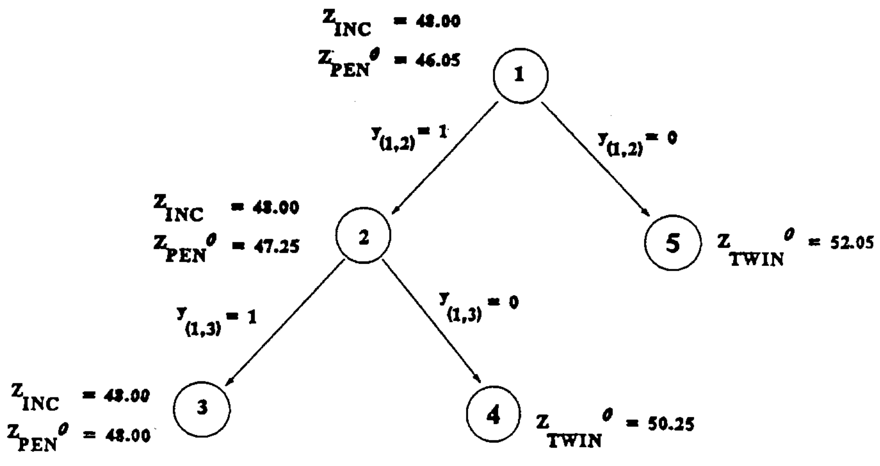


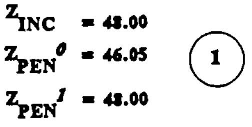
Fig. 6. Example network.

Table II. Example cost and flow coefficients

Arc i	Marginal costs		Flow bounds		
	a_i	b_i	l_i	m_i	u_i
(1,2)	3	0	0	1	7
(1,3)	4	1	0	1	5
(2,3)	5	2	0	1	3
(2,4)	6	3	0	1	6
(3,4)	8	5	0	1	4



(a)



(b)

Fig. 7. Example branch and bound enumeration trees: (a) using BBSTANDARD; (b) using BBSBI.

(1, 3); 0 on (2, 3); 6 on (2, 4); and 3 on (3, 4). "Rounding" this solution produced an incumbent solution with an objective function value of $z_{\text{INC}} = 48.00$. This incumbent solution was also the optimal solution to MCCNF. But, because the penalty lower bound was only $z^0_{\text{PEN}} = 46.05$, subprogram 1 could not be fathomed. Thus, using arc (1, 2) as the branching arc, two new subprograms – one with $(1,2) \in \text{ZERO}$ and the other with $(1,2) \in \text{ONE}$ – were created and the process was repeated.

In all, BBSTANDARD required the evaluation of three subprograms. [Note that subprograms 4 and 5 did not need evaluation because their stored penalty lower bound, z^0_{TWIN} (see eq. (25)), exceeded z_{INC}.] The CPU time for this algorithm was 0.11 seconds and the maximum depth for its enumeration tree was three.

In contrast, as shown in Figure 7b, the enumeration "tree" for BBSBI consisted solely of the root node representing the original mixed integer program, MCCNF. As with BBSTANDARD, LP^0, the linear programming relaxation of this subprogram, was solved using a network simplex algorithm; the incumbent solution was obtained by "rounding-up" the relaxation solution; and the penalty lower bound was computed. However, in BBSBI the SBI procedure was then performed to generate tighter lower and upper flow bounds, $\{l^k_i\}$ and $\{u^k_i\}$, and then program LP^1 was solved. This increased the penalty lower bound to $z^1_{PEN} = 48.00$ (which equals z_{INC}) so that the entire branch and bound enumeration tree was fathomed at the root node.

Thus, for BBSBI, the maximum depth in the enumeration tree was one. Two relaxations, LP^0 and LP^1, were solved, but the solution to LP^1 was trivial since the optimal solution to LP^0 was also optimal in LP^1. The total CPU time was 0.05 seconds.

The dominance of BBSBI over BBSTANDARD, brought out in this example, is next examined in a series of computational tests.

7.2. COMPUTATION TESTS

In order to evaluate the SBI procedure more fully, a series of test problems was solved with BBSTANDARD and BBSBI. Below, we describe how the problems were generated and comment on the results of these two branch and bound algorithms.

• *Problem Generation*

As summarized in Table III, three sizes of MCCNF problems were considered, each consisting of five randomly generated test networks. All networks were complete; i.e., there was a directed arc between every pair of nodes. Remember that, because the arc cost functions for our problems were piecewise-linear-concave, each arc in the network corresponded to a binary decision variable in program MCCNF. Thus, it is reasonable to characterize a network with over 200 such arcs as a "large" problem.

For each arc $i \in \text{ARC}$ in each test problem, the marginal costs a_i and b_i were randomly sampled from a uniform distribution UNIFORM[0, 100]. If $a_i > b_i$, then the values of a_i and b_i were interchanged. Also, for each arc i, the original flow bounds l_i, m_i, and u_i were sampled from UNIFORM[0, 100]. The values of l_i, m_i, and u_i were sorted so that $l_i \leq m_i \leq u_i$. For each test problem a node, denoted $\bar{n}$, was randomly selected. Then, for each node $n \in \text{NODE} - \{\bar{n}\}$, the demand/supply constant d_n was sampled from UNIFORM[−10, +10]; and the demand/supply constant for node $\bar{n}$ was set such that the total demand and supply in the network summed to zero.

Table III. Test problem characteristics

Problem number	Size	Number of nodes	Number of arcs
1 to 5	Small	5	20
6 to 10	Medium	10	90
11 to 15	Large	15	210

• *Results*

Each of the fifteen test problems identified in Table III was solved twice, first using BBSTANDARD, then using BBSBI. The results are shown in Tables IV and V. These tables show the average value and the range for the five test problems solved in each problem size. The results for BBSTANDARD and BBSBI are given; and the percent improvement of BBSBI over BBSTANDARD is reported.

Table IV, giving the maximum depth in the branch and bound enumeration tree, indicates the relative in-core storage requirements for the test problems. In all cases, the SBI procedure reduced the storage requirements for the branch and bound algorithm. Comparing BBSBI with BBSTANDARD, there was, on average, more than a two-thirds improvement for small problems, and more than a one-third improvement for large ones. Thus, although the amount of improvement decreased as the problem size increased, the overall reduction in in-core storage was still substantial.

Table V reports the CPU time required to solve the test problems. Once again, the SBI procedure increased the efficiency of the branch and bound procedure. Moreover, this improvement increased as the problem size increased. Thus, comparing BBSBI with BBSTANDARD, there was, on average, more than a forty percent reduction in the time required to solve the large test problems.

It should be pointed out, though, that for one test problem, problem 10, the CPU time for BBSBI was slightly greater than that for BBSTANDARD. For this particular test problem, BBSTANDARD required the evaluation of 65 subprograms whereas BBSBI required the evaluation of 27 families of subprograms.

Table IV. Test problem results: depth in enumeration tree

Problem size	Depth in enumeration tree				Percent improvement	
	BBSTANDARD		BBSBI			
	Range	Avg.	Range	Avg.	Range	Avg.
Small	2 to 5	4.5	1 to 2	1.2	+50 to +80	+68
Medium	11 to 15	13.0	1 to 10	5.6	+27 to +92	+57
Large	19 to 32	27.0	11 to 22	17.4	+29 to +41	+36

Table V. Test problem results: CPU time

Problem size	CPU seconds[a]				Percent improvement	
	BBSTANDARD		BBSBI			
	Range	Avg.	Range	Avg.	Range	Avg.
Small	0.2 to 0.9	0.5	0.2 to 0.7	0.4	+15 to +45	+21
Medium	6.2 to 47.2	27.5	4.1 to 38.6	21.8	−3 to +65	+27
Large	320.5 to 1863.6	923.4	151.7 to 862.5	479.0	+14 to +56	+44

[a] Using a Micro Source International microcomputer (comparable to an IBM-AT).

But, because in BBSBI each family consisted of an average of three subprograms, this problem had an overall solution time of 34 seconds using BBSBI (compared to 33 seconds using BBSTANDARD). Thus, we see from Table V, that although the SBI procedure cannot be guaranteed to reduce computation time of the branch and bound procedure, it will, on average, have a significant impact.

8. Conclusions and Extensions

This paper has presented a new branch and bound algorithm for minimum concave cost network flow problems with piecewise-linear arc cost functions. The distinctive feature of this algorithm is the incorporation of the simple bound improvement (SBI) method. As shown by the computational tests conducted in Section 7, the SBI procedure reduces both the CPU time and the in-core storage requirements of the branch and bound algorithm.

In closing, three alternative implementations of the SBI procedure are worth noting. First, although the SBI procedure discussed in this paper was used in conjunction with Driebeek's [9] penalty bounds, other penalty procedures (see, for example, [34] and [8]) could also be used instead. In this case, the SBI procedure would produce tighter bounds for each subprogram evaluated in the branch and bound, but with somewhat increased computational effort.

Second, when a program was selected by backtracking in the candidate list in the branch and bound procedure described in this paper, the lower and upper flow bounds for arcs in set FREE were set to the values determined in the root node subprogram. In this manner, no additional storage requirement for the improved arc flow bounds was required. An alternative implementation of the branch and bound algorithm would be to store the improved arc flow bounds along with each of the subprogram in the candidate list. Then, because the initial flow bounds of each subprogram would be tighter, less computational effort would be required for evaluating each subprogram, but at the cost of increased in-core storage.

Finally, this paper has focused on the SBI procedure for network flow problems with piecewise-liner-concave arc cost functions. We point out, however, that the SBI procedure is also applicable to problems with more general objective functions as well as to problems involving constraints other than network flow constraints (see [25]).

Acknowledgements

This work was supported, in part, by a research grant from the Institute of Transportation Studies, University of California, Irvine, CA. The author would like to thank MaryAnne Causino for her thoughtful reading of the manuscript, Barbara Skelly for her preparation of the illustrations and the referees for their constructive comments.

References

1. P. Afentakis, B. Gavish, and U. Karmarkar (1984), Computationally Efficient Optimal Solutions to the Lot-Sizing Problem in Multistage Assembly Systems, *Management Science* **30**(2), 222–239.
2. A. Balakrishnan (1984), Valid Inequalities and Algorithms for the Network Design Problem with Application to LTL Consolidation, Ph.D. dissertation, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA.
3. M. L. Balinski (1961), Fixed-Cost Transportation Problems, *Naval Research Logistics Quarterly* **8**(1), 41–54.
4. R. S. Barr, F. Glover, and D. Klingman (1979), Enhancements of Spanning Tree Labelling Procedures for Network Optimization, *INFOR* **17**(1) 16–34.
5. R. S. Barr, F. Glover, and D. Klingman (1981), A New Optimization Method for Large Scale Fixed Charge Transportation Problems, *Operations Research* **29**(3), 448–463.
6. K. Belling-Seib, P. Mever, and C. Müller (1988), Network Flow Problems with One Side Constraint: A Comparison of Three Solution Methods, *Computers and Operations Research* **15**(4), 381–394.
7. A. V. Cabot and S. S. Erenguc (1984), Some Branch-and-Bound Procedures for Fixed-Cost Transportation Problems, *Naval Research Logistics Quarterly* **31**, 145–154.
8. A. V. Cabot and S. S. Erenguc (1986), Improved Penalties for Fixed Cost Linear Programs Using Lagrangian Relaxation, *Management Science* **32**(7), 856–869.
9. N. Driebeek (1966), An algorithm for the Solution of Mixed Integer Programming Problems, *Management Science* **12**(7), 576–587.
10. R. E. Erickson, C. L. Monna, and A. F. Veinott, Jr. (1987), Send-and-Split Method for Minimum-Concave-Cost Network Flows, *Mathematics of Operations Research* **12**(4), 634–664.
11. M. Florian and P. Robillard (1971), An Implicit Enumeration Algorithm for the Concave Cost Network Flow Problem, *Management Science* **18**(3), 184–193.
12. L. R. Ford and D. R. Fulkerson (1962), *Flows in Networks*, Princeton University Press, Princeton, NJ.
13. G. Gallo, C. Sandi, and C. Sodini (1980), An Algorithm for the Min Concave Cost Flow Problem, *European Journal of Operational Research* **4**, 248–255.
14. M. R. Garey and D. S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, Inc., San Francisco, CA.
15. F. Glover, D. Karney, D. Klingman, R. Russell (1978), Solving Singly Constrained Transshipment Problems, *Transportation Science* **12**(4), 277–297.
16. P. Gray (1971), Exact Solution of the Fixed-Charge Transportation Problem, *Operations Research* **19**(6), 1529–1538.
17. G. Guisewite and P. M. Pardalos (1990), Minimum Concave Cost Network Flow Problems: Applications, Complexity, and Algorithms, *Annals of Operations Research* **25**, 75–100.
18. G. Guisewite and P. M. Pardalos (1991a), Single-Source Uncapacitated Minimum Concave Cost Network Flow Problems, in H. E. Bradley (ed.), *Operational Research '90*, Pergamon Press, Oxford, England, pp. 703–713.
19. G. Guisewite and P. M. Pardalos (1991b), Algorithms for the Single-Source Uncapacitated Minimum Concave-Cost Network Flow Problem, *Journal of Global Optimization* **1**, 245–265.
20. G. Guisewite and P. M. Pardalos (1991c), Global Search Algorithms for Minimum Concave Cost Network Flow Problems, *Journal of Global Optimization* **1**, 309–330.
21. G. Guisewite and P. M. Pardalos (1992), Performance of Local Search in Minimum Concave-Cost Network Flow Problems, in C. A. Floudas and P. M. Pardalos (eds.), *Recent Advances in Global Optimization*, Princeton University Press, Princeton, NJ, pp. 50–75.
22. E. L. Johnson (1966), Networks and Basic Solutions, *Operations Research* **14**(4), 619–623.
23. J. Kennington and E. Unger (1976), A New Branch-and-Bound Algorithm for the Fixed-Charge Transportation Problem, *Management Science* **22**(10), 1116–1126.
24. B. W. Lamar (1985), Network Design Algorithms with Applications to Freight Transportation, Ph.D. dissertation, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge, MA.

25. B. W. Lamar (1993), A Method for Solving Network Flow Problems with General Nonlinear Arc Costs, in D.-Z. Du and P. M. Pardalos (eds.), *Network Optimization Problems: Algorithms, Complexity, and Applications*, World Scientific, Singapore; Teaneck, NJ, pp. 147–168.
26. B. W. Lamar and Y. Sheffi (1988), An Implicit Enumeration Method for LTL Network Design, *Transportation Research Record* **1120**, 1–16.
27. B. W. Lamar, Y. Sheffi, and W. B. Powell (1990), A Capacity Improvement Lower Bound For Fixed Charge Network Design Problems, *Operations Research* **38**, 704–710.
28. J. D. C. Little, K. C. Murty, D. W. Sweeney, and C. Karel (1963), An Algorithm for the Travelling Salesman Problem, *Operations Research* **11**, 972–989.
29. M. Los and C. Lardinois (1982), Combinatorial Programming, Statistical Optimization and the Optimal Transportation Network Problem, *Transportation Research-B* **16B**(2), 89–124.
30. M. Malek-Zavarei and I. T. Frisch (1972), On the Fixed Cost Flow Problem, *International Journal of Control* **16**(5), 897–902.
31. R. G. Parker and R. L. Rardin (1988), *Discrete Optimization*, Academic Press, Inc., San Diego, CA.
32. G. Sa (1969), Branch-and-Bound and Approximate Solutions to the Capacitated Plant Location Problem, *Operations Research* **17**, 1005–1016.
33. R. M. Soland (1974), Optimal Facility Location with Concave Costs, *Operations Research* **22**, 373–382.
34. J. A. Tomlin (1971), An Improved Branch and Bound Method for Integer Programming, *Operations Research* **19**, 1070–1075.
35. B. Yaged, Jr. (1971), Minimum Cost Routing for Static Network Models, *Networks* **1**, 139–172.
36. W. I. Zangwill (1968), Minimum Concave Cost Flows in Certain Networks, *Management Science* **14**(7), 429–450.