Systems
Optimization
Laboratory

(4)

An Exact Ceiling Point Algorithm for General Integer Linear Programming

by
Robert M. Saltzman
and Frederick S. Hillier

TECHNICAL REPORT SOL 88-20

November 1988

Department of Operations Research
Stanford University
Stanford, CA 94305

(4)

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-4022

An Exact Ceiling Point Algorithm for General Integer Linear Programming

by
Robert M. Saltzman
and Frederick S. Hillier

TECHNICAL REPORT SOL 88-20

November 1988

Research and reproduction of this report were partially supported by the Office of Naval Research Contract N00014-85-K-0343.

Abstract

An Exact Ceiling Point Algorithm for General Integer Linear Programming

Robert M. Saltzman and Frederick S. Hillier

Stanford University, 1988

This report describes an exact algorithm for the pure, general integer linear programming problem (ILP). Common applications of this model occur in capital budgeting (project selection), resource allocation and fixed-charge (plant location) problems. The central theme of our algorithm is to enumerate a subset of all solutions called "feasible 1-ceiling points." A feasible 1-ceiling point may be thought of as an integer solution lying on or near the boundary of the feasible region for the LP-relaxation associated with (ILP). Precise definitions of 1-ceiling points and the role they play in an integer linear program are presented in a recent report by the authors. One key theorem therein demonstrates that all optimal solutions for an (ILP) whose feasible region is non-empty and bounded are feasible 1-ceiling points. Consequently, such a problem may be solved by enumerating just its feasible 1-ceiling points. Our approach is to implicitly enumerate 1-ceiling points with respect to one constraint at a time while simultaneously considering feasibility. Computational results from applying this incumbent-improving Exact Ceiling Point Algorithm to 48 test problems taken from the literature indicate that this enumeration scheme may hold potential as a practical approach for solving problems with certain types of structure.

Subject Classification for OR/MS Index: Programming – Integer Algorithms; Branch-and-bound

Key Words: integer linear programming; general integer variables; exact algorithm; ceiling points; implicit enumeration; linear programming relaxation

1. Introduction

This report describes an exact algorithm for solving the pure, general integer linear programming problem in m constraints and n variables x_j, $j = 1, ..., n$, whose form is

$$\begin{aligned} &\text{Maximize } c^T x = z \\ &\text{subject to } Ax \leq b \qquad (ILP) \\ &x \geq 0, \quad x \text{ integer,} \end{aligned}$$

where $A \in \Re^{m \times n}$, $b \in \Re^m$ and $c \in \Re^n$. All the data $\{A, b, c\}$ are assumed to be rational numbers, but they are unrestricted in sign. The problem is "pure" in that all of the variables are required to take on nonnegative integer values. It is "general" in the sense that the variables may take on any nonnegative integer values permitted by $Ax \leq b$, as opposed to being restricted to 0 or 1 (the binary case). An important additional assumption is that no implicit or explicit equality constraints are used to define the feasible region $FR \equiv \{x \geq 0 | \ Ax \leq b\}$ for (LP_R), the linear programming relaxation associated with (ILP). Common applications of this model occur in capital budgeting (project selection), resource allocation and fixed-charge (plant location) problems. A further discussion of application areas for (ILP) may be found in [11] or [26].

Another report by the authors [24] describes a procedure for approximately solving (ILP) called the Heuristic Ceiling Point Algorithm. Like all heuristic algorithms, it offers no guarantee of finding an optimal solution. In fact, while the Heuristic Ceiling Point Algorithm usually finds solutions of very hiqh quality, the algorithm is not even guaranteed to find a *feasible* solution. Furthermore, even with an optimal integer solution at hand, the Heuristic Ceiling Point Algorithm can only rarely verify (prove) that such a solution is indeed optimal. For these reasons, an exact algorithm was developed based on a more systematic search for feasible "1-ceiling points." To understand why this might be a reasonable approach, we briefly review some of the key concepts of [23].

An integer solution x is a *1-ceiling point with respect to the i^{th} constraint*, denoted $x = 1\text{-}CP(i)$, if (1) x satisfies this constraint, *i.e.*, $a_i^T x \leq b_i$ (where a_i is the i^{th} row of the constraint matrix A), and (2) modifying some component of x by $+1$ or -1 yields a

solution which violates this constraint, *i.e.*, $a_i^T x + |a_{ij}| > b_i$ for at least one j. Thus, $x =$ 1-$CP(i)$ means x narrowly satisfies the i^{th} constraint: taking a unit step from x toward the i^{th} constraining hyperplane in a direction parallel to some coordinate axis results in an infeasible point. Similarly, an integer solution x is defined to be a *1-ceiling point with respect to the feasible region* FR, denoted $x =$ 1-$CP(FR)$, if (1) x satisfies all constraints and (2) modifying some component of x by $+1$ or -1 leads to a solution which violates one or more constraints, *i.e.*, $\exists i : a_i^T x + |a_{ij}| > b_i$ for at least one j. It is demonstrated in [23] that all optimal solutions for an (ILP) whose feasible region is non-empty and bounded are feasible 1-$CP(i)$'s, *i.e.*, 1-$CP(FR)$'s. Consequently, one way to solve (ILP) is to enumerate its feasible 1-ceiling points. Our heuristic approach is based upon the idea that a feasible 1-ceiling point found relatively near $\bar{x}$ is apt to have a high (possibly even optimal) objective function value. On 48 test problems taken from the literature, searching for such 1-ceiling points usually did provide a very good solution with a moderate amount of computational effort.

The Exact Ceiling Point Algorithm begins by executing the Heuristic Ceiling Point Algorithm (which includes solving the linear programming relaxation), so Section 2 examines the assumptions needed for these steps. The goal at this stage is to be able to construct a small but sufficient search region. Section 3 discusses how this search region is divided up to permit the search to focus on finding 1-ceiling points with respect to one specific constraint. This leads to the calculation of unconditional variable bounds which correspond geometrically to a search hyperrectangle. The overall process of searching for 1-ceiling points within this search hyperrectangle is outlined at the beginning of Section 4 and then each of three subsections provide more detail on a particular aspect of the search. The first subsection describes the calculation of conditional variable bounds which are at least as strong as the unconditional variable bounds; the second subsection describes an efficient aspect of the enumeration procedure called "double backtracking" which allows subregions of the search rectangle to be skipped altogether; the third subsection describes what occurs when a new incumbent solution is found. An overview of the entire Exact Ceiling Point Algorithm is given is Section 5. Section 6 describes a preliminary search procedure on a constraint called an "intersection cut," where this procedure is designed

to speed up our algorithm by searching the region closest to $\bar{x}$ first. The choice of a search constraint in the Exact Ceiling Point Algorithm is discussed in Section 7, while a few other computational issues are examined in Section 8. Section 9 reports on our computational experience. Section 10 offers some conclusions about our approach and is followed by two appendices. The first appendix gives the variable bounds and options used in the GAMS/ZOOM runs reported in Section 9, while the second lists the Fortran code implementation of the Exact Ceiling Point Algorithm.

2. Assumptions

The Exact Ceiling Point Algorithm begins by finding an optimal solution $\bar{x}$ for the LP-relaxation (LP_R), the set $\bar{A}$ of constraints binding at $\bar{x}$, and the set of extreme directions emanating from $\bar{x}$ which form the cone $\overline{FR}$. The Exact Algorithm also uses an initial feasible integer solution, x_H, provided by the Heuristic Algorithm. The assumptions made here are very similar to those specified in [16].

Assumption 1: The set of feasible solutions for (LP_R) is non-empty and bounded.

Assumption 2: The optimal solution found for (LP_R) is not all-integer.

Assumption 3: The unique optimal solution for (LP_R) is $\bar{x}$.

Assumption 4: A feasible solution x_H for (ILP), with objective value $\underline{z}$, is known.

The first assumption implies that $\bar{x}$ exists. The second implies that (ILP) is not solved simply by solving (LP_R), so that we have a need for an exact algorithm. The third assumption is the most serious; however, when $\bar{x}$ is not unique, there are ways to perturb the data of (ILP) without altering its optimal solution(s) so that this condition does hold. Another alternative is to continue to append cutting planes to the problem until a unique $\bar{x}$ is found (see [16, pp. 670-673]). The last assumption is not really needed for the Exact Ceiling Point Algorithm to run, but vastly improves its performance since it provides another relatively strong constraint, $c^T x \geq \underline{z}$, on the solutions that need to

be considered. If no feasible integer solution for (ILP) has been identified prior to the start of the Exact Algorithm, then the algorithm starts with $\underline{z} = -\infty$. Alternatively, the value of some feasible non-integer solution could be used as a lower bound on the optimal objective value of (ILP). Then, if it is found that there do not exist any feasible integer solutions whose objective value exceeds or equals this bound, so the bound is not a valid one, the Exact Algorithm would have to be restarted with a smaller lower bound, and the previous lower bound would then become an upper bound [16, p. 673].

With one more definition, we will be able to define a volume which must be searched to find an optimal solution for (ILP). For $k = 1, ..., n$, let p^k be defined as the point of intersection between the k^{th} extreme direction d^k emanating from $\bar{x}$ and the objective constraint hyperplane $c^T x = \underline{z}$. These intersection points $\{p^k, k = 1, ..., n\}$ exist as long as $\bar{x}$ exists and the objective constraint hyperplane is not parallel to any edge or face of the cone $\overline{FR}$ (Assumptions 1 and 3). The point p^k has the form $\bar{x} + \lambda^k d^k$, where λ^k is such that

$$c^T(\bar{x} + \lambda^k d^k) = \underline{z}.$$

Solving for λ^k yields

$$\lambda^k = (\underline{z} - c^T\bar{x})/c^T d^k.$$

Therefore,

$$p^k = \bar{x} + (\underline{z} - c^T\bar{x})d^k/c^T d^k, \text{ for } k = 1, ..., n.$$

For notational convenience in what follows, define $p^0 \equiv \bar{x}$. Now we can restate Hillier's key theorem [16, Theorem 1] for this point in the analysis.

Theorem 1. Under Assumptions 1, 2, and 3, all optimal solutions for (ILP) are contained in the n-simplex S whose $(n+1)$ extreme points are $\{p^0, p^1, ..., p^n\}$.

An example in $\Re^3$ of an n-simplex S is shown in Figure 1, where the extreme directions emanating from $\bar{x}$ are $\{d^1, d^2, d^3\}$. Theorem 1 indicates that once $\underline{z}$ has been specified, our attention may be confined to S without fear of missing any optimal solutions for (ILP). Consequently, any non-binding constraint which does not intersect S may be disregarded henceforth. In the Exact Algorithm, any constraint which is not violated by

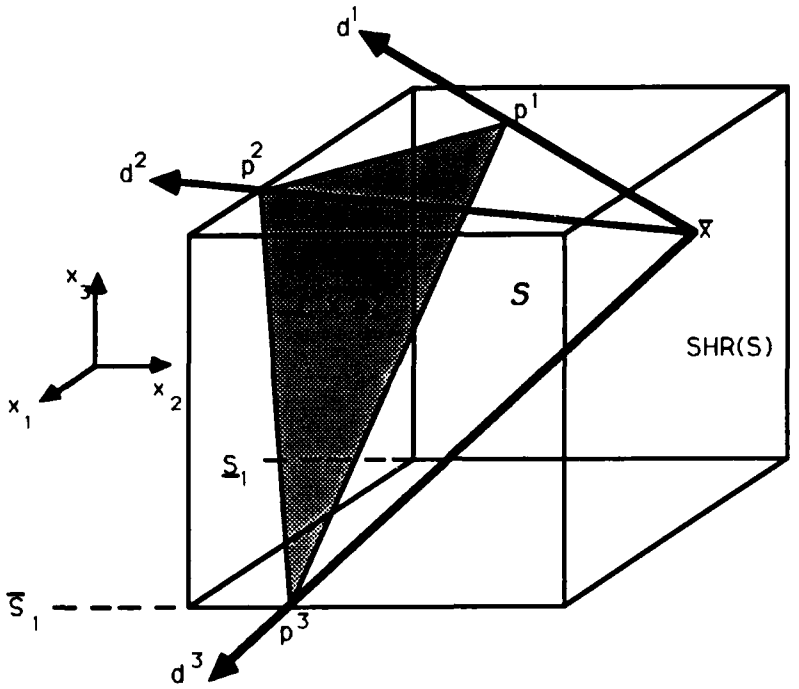

Figure 1. An n-simplex S in $\Re^3$ with vertices $\{\bar{x}, p^1, p^2, p^3\}$ and the corresponding search hyperrectangle $SHR(S)$.

at least one p^k is redundant and dropped from the further consideration. On the other hand, the n-simplex may be larger than necessary due to the presence of (non-redundant) non-binding constraints which chop off parts of the n-simplex. Notice that if we seek only solutions which are strictly better than x_H and all the c_j are integer, the objective constraint hyperplane to use in the determination of $\{p^k,\ k = 1, ..., n\}$ is $c^T x = \underline{z} + 1$. It seems worthwhile to recompute this set of intersection points every time a new (higher-valued) incumbent solution is found since this reduces the size of the n-simplex. Finally, we may calculate a set of lower and upper bounds $[\underline{S}_j, \overline{S}_j]$ for x_j by finding the minimum and maximum, respectively, over the j^{th} component of all vertices of the n-simplex and rounding appropriately. The lower bound $\underline{S}_j$ is not permitted to be less than 0:

$$\underline{S}_j \equiv \max\{\lceil \min_k \{p_j^k\} \rceil, 0\}, \text{ for } j = 1, \ldots, n,$$

and

$$\overline{S}_j \equiv \lfloor \max_k \{p_j^k\} \rfloor, \text{ for } j = 1, \ldots, n.$$

These bounds define what we shall refer to as $SHR(S)$, the search hyperrectangle for the n-simplex S, as illustrated in Figure 1. If, for any component j, we find that the upper bound $\overline{S}_j$ is strictly less than the the lower bound $\underline{S}_j$, then $SHR(S) = \emptyset$. In this case, the problem is solved because there are no feasible integer solutions better than the incumbent. Otherwise, we begin to search for 1-$CP(FR)$'s.

3. Unconditional Variable Bounds

The preceding section indicated that to solve (ILP) it is sufficient to enumerate the feasible 1-ceiling points contained in the region defined by $SHR(S)$. To improve efficiency, we split up the n-simplex and further confine our search to subhyperrectangles of $SHR(S)$. Like the Heuristic Ceiling Point Algorithm, the Exact Algorithm seeks 1-ceiling points with respect to one search constraint at a time. Therefore, a search hyperrectangle $SHR(i)$ for constraint (i) contained within $SHR(S)$ is constructed which is large enough to contain all feasible 1-$CP(i)$'s with objective value greater than $\underline{z}$. This subhyperrectangle is constructed once a promising search constraint has been identified, which is the topic of Section 7.

From Definition 3.4, a necessary and sufficient condition for an integer solution x to be a 1-ceiling point with respect to constraint (i) is

$$x = 1\text{-}CP(i) \iff 0 \leq b_i - a_i^T x \leq \max_j |a_{ij}| - 1 \tag{1}$$

where all coefficients a_{ij} are assumed to be integer. (If not all coefficients are integer, subtract a small positive quantity ϵ instead of 1 from the right hand side of (1). This will be the case when we search the intersection cut described in Section 6.) In other words, for x to be a 1-ceiling point with respect to constraint (i), it must satisfy the constraint

$$a_i^T x \leq b_i \tag{i}$$

but not be too far away from the i^{th} constraint hyperplane, *i.e.*, satisfy

$$a_i^T x \geq b_i - t_i \qquad (i')$$

where $t_i \equiv \max_j |a_{ij}| - 1$ may be thought of as the distance constraint (i) is translated in order to impose the ceiling point condition. Thus, only those integer solutions satisfying both constraints (i) and (i') are 1-$CP(i)$'s. These two constraints are central to the construction of the appropriate search region. An example in $\Re^2$ is given in Figure 2.

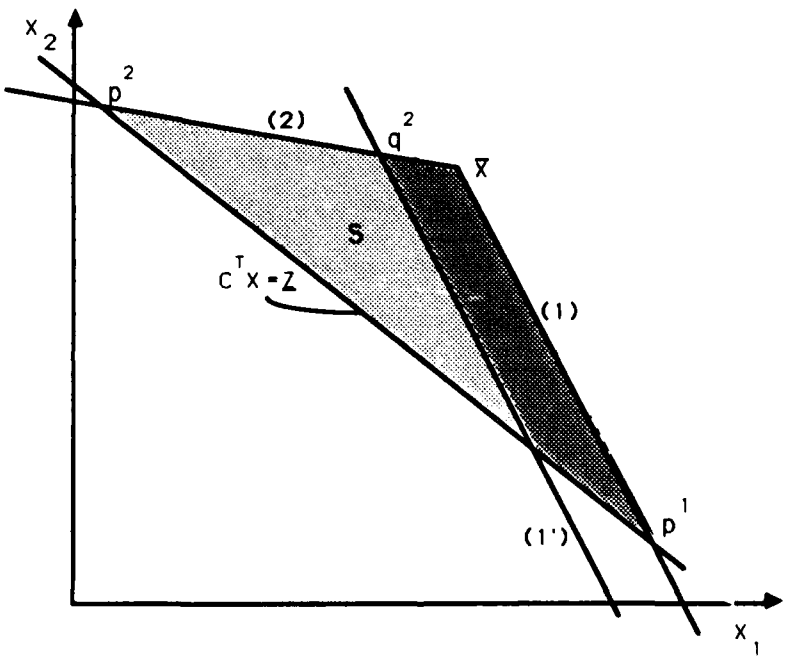

Figure 2. All feasible 1-$CP(1)$'s with objective value as large as $\underline{z}$ lie in the part of S between (1) and (1').

Now let E_i be the set of extreme rays emanating from $\bar{x}$ which lie on the i^{th} constraint hyperplane. Also let q^k be the point of intersection between the k^{th} extreme ray of $\overline{FR}$ and the translated constraint hyperplane (i'), for all $k \notin E_i$. In Figure 2, the point q^2 is shown as the intersection of constraint hyperplane (1') and extreme ray 2, where extreme ray 2 coincides with constraint hyperplane (2). When exactly n constraints are binding at

$\bar{x}$, each translated constraint hyperplane (i') intersects just one extreme ray because the other $(n-1)$ extreme rays lie on constraint hyperplane (i) which is parallel to (i'). In the case where $\bar{x}$ is overdetermined, more than one extreme ray will intersect the translated constraint hyperplane (i'). The point q^k has the form $\bar{x} + \gamma^k d^k$, where γ^k is such that

$$a_i^T(\bar{x} + \gamma^k d^k) = b_i - t_i.$$

Solving for γ^k yields

$$\gamma^k = -t_i / a_i^T d^k.$$

Therefore,

$$q^k = \bar{x} - (t_i / a_i^T d^k) d^k, \ \forall k \notin E_i.$$

The search hyperrectangle for this constraint, $SHR(i)$, is defined by the ranges $\{[l_j, u_j], j = 1, ..., n\}$, where the bounds for each x_j are formed by taking the minimum and maximum, respectively, over the j^{th} component of all of the intersection points required to define the hyperrectangle's vertices and rounding appropriately. Furthermore, these bounds should be no wider than those defining $SHR(S)$ because searching beyond the boundary of $SHR(S)$ is unproductive:

$$l_j \equiv \min\{\lceil \min\{\min_{k \in E_i}\{p_j^k\}, \ \min_{k \notin E_i}\{q_j^k\}\}\rceil, \ \underline{S}_j\} \tag{2l}$$

and

$$u_j \equiv \max\{\lfloor \max\{\max_{k \in E_i}\{p_j^k\}, \ \max_{k \notin E_i}\{q_j^k\}\}\rfloor, \ \overline{S}_j\}. \tag{2u}$$

If, for any component j, we find $u_j < l_j$, then $SHR(i) = \emptyset$, implying that there are no integer solutions at all in this search hyperrectangle, and a new search constraint is identified. Assuming this is not the case, we proceed to enumerate the integer solutions contained within the search hyperrectangle, as described in the next section.

4. Enumerating Solutions Within a Search Hyperrectangle

Once the search hyperrectangle $SHR(i)$ has been defined for search constraint (i), finding 1-$CP(i)$'s better than the incumbent amounts to examining $SHR(i)$ for solutions which are feasible with respect to all the relevant constraints, including (i'). Because this search hyperrectangle is contained within the n-simplex, *i.e.*, $SHR(i) \subseteq SHR(S)$, any feasible integer solution found will become the new incumbent solution. The overall enumeration process, shown as a flow diagram in Figure 3, contains three features which differentiate it from a simple exhaustive enumeration scheme: the use of conditional variable bounds, a potential for "double backtracking," and a tightening of bounds when a new incumbent is found. These are described in each of the next three subsections.

4.1. Conditional Variable Bounds

Suppose the variables are rearranged so that x_1 is fixed first, x_2 is fixed second, and so forth. Given a "partial solution" $(x_1, x_2, ..., x_{j-1})$ whose components are fixed in value, a "completion" of the partial solution refers to an assignment of values to the remaining free components. Fixing the value of the first variable x_1 to an integer value between l_1 and u_1 may reduce the range of values for some or all of the free variables $(x_2, ..., x_n)$. Similarly, fixing the first $(j-1)$ variables may tighten the bounds on the remaining variables. In other words, it is possible to calculate conditional bounds on the free variables given the value of the fixed variables.

As in the Heuristic Ceiling Point Algorithm, we assume that all constraints are in $\leq$ form, having multiplied any $\geq$ constraints through by -1 if necessary. The following procedure develops conditional bounds $[L_j|x_{j-1}, U_j|x_{j-1}]$ for the variable x_j, given that variables $x_1, ..., x_{j-1}$ are fixed in value and $x_j, ..., x_n$ are free. (In Figure 3 these conditional bounds are abbreviated as $[L_j, U_j]$.) First, some useful notation is introduced. Let

$$g_{ij} \equiv b_i - \sum_{k=1}^{j-1} a_{ij} x_k$$

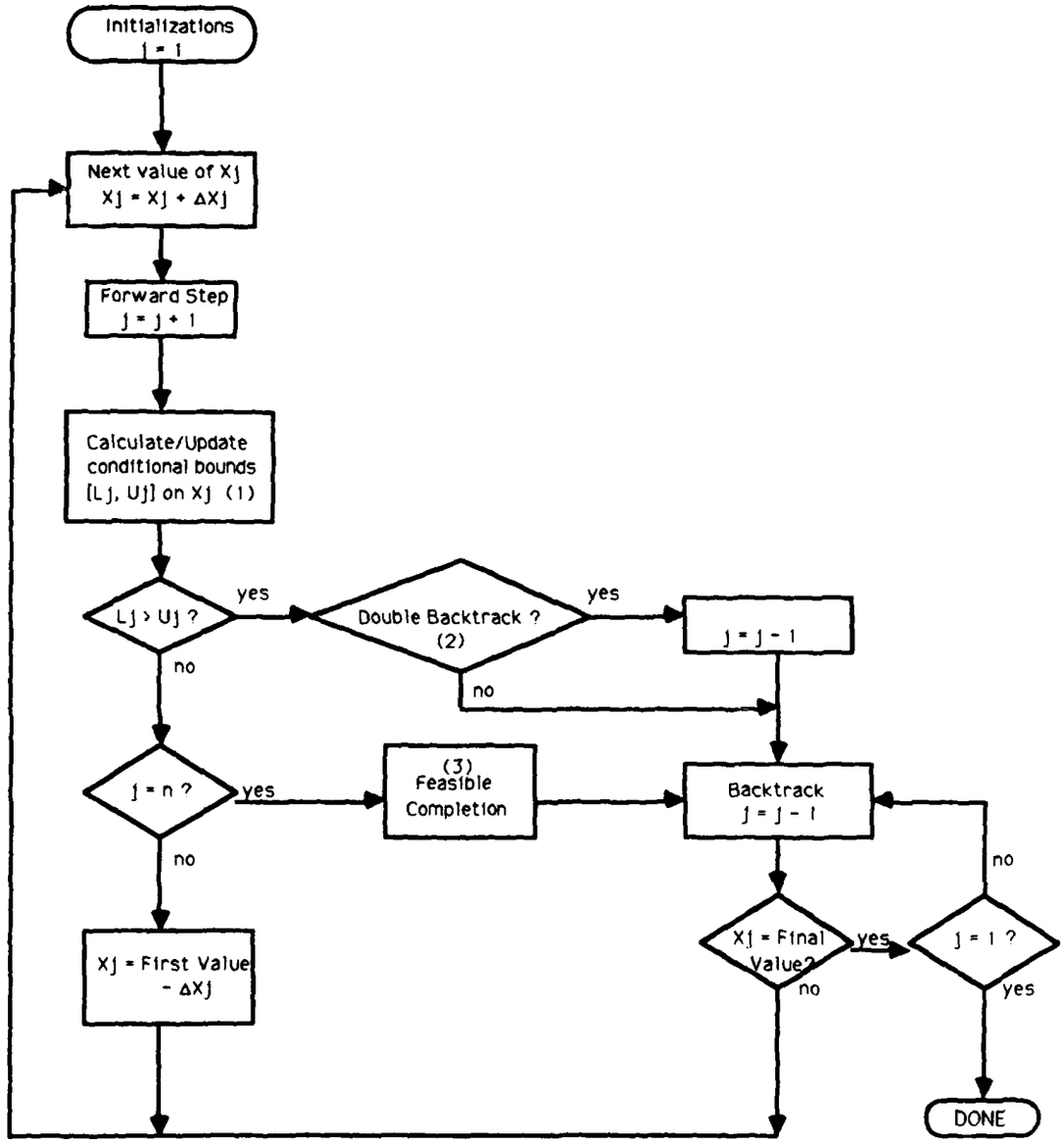

Figure 3. Flow diagram of the Exact Algorithm's enumeration scheme.

(1) See Section 4.1 for further explanation.

(2) See Section 4.2.

(3) See Section 4.3.

be the gap or slack remaining in constraint (i) after having fixed $x_1, ..., x_{j-1}$. Also, let

$$w_{i,j+1} \equiv \sum_{k=j+1}^{n} \min\{a_{ik}l_k, a_{ik}u_k\} \tag{3}$$

be the minimum amount of the gap g_{ij} that may be used up by fixing variables $x_{j+1}, ..., x_n$ within their unconditional bounds. Feasibility of a complete solution x with respect to the i^{th} constraint requires

$$\sum_{k=1}^{j-1} a_{ik}x_k + a_{ij}x_j + \sum_{k=j+1}^{n} a_{ik}x_k \leq b_i$$

$$\Rightarrow a_{ij}x_j \leq b_i - \sum_{k=1}^{j-1} a_{ik}x_k - \sum_{k=j+1}^{n} a_{ik}x_k$$

$$\Rightarrow a_{ij}x_j \leq g_{ij} - \sum_{k=j+1}^{n} a_{ik}x_k$$

$$\Rightarrow a_{ij}x_j \leq g_{ij} - w_{i,j+1} \tag{4}$$

If a_{ij} is positive, then (4) yields an upper conditional bound for x_j when just constraint (i) is considered:

$$x_j \leq (g_{ij} - w_{i,j+1})/a_{ij}. \tag{5}$$

On the other hand, if a_{ij} is negative, (4) yields a lower conditional bound for x_j when just constraint (i) is considered:

$$x_j \geq (g_{ij} - w_{i,j+1})/a_{ij}. \tag{6}$$

Note that the numerator may be negative, so that the lower conditional bound is not necessarily negative. If a_{ij} is positive for all (i), (4) yields only upper conditional bounds. Finally, if $a_{ij} = 0$, constraint (i) cannot be used to determine a conditional bound on x_j. Letting $L_{ij}|x_{j-1}$ and $U_{ij}|x_{j-1}$ denote the lower and upper conditional bounds, respectively, on x_j when constraint (i) is considered alone after fixing variables $(x_1, ..., x_{j-1})$, we have

$$L_{ij}|x_{j-1} \equiv \begin{cases} (g_{ij} - w_{i,j+1})/a_{ij}, & \text{if } a_{ij} < 0; \\ l_j, & \text{if } a_{ij} \geq 0, \end{cases}$$

and

$$U_{ij}|x_{j-1} \equiv \begin{cases} u_j, & \text{if } a_{ij} \leq 0; \\ (g_{ij} - w_{i,j+1})/a_{ij}, & \text{if } a_{ij} > 0, \end{cases}$$

where l_j and u_j are the unconditional lower and upper bounds on x_j defined earlier. To maintain feasibility with respect to *all* constraints, the conditional lower and upper bounds $[L_j|x_{j-1}, U_j|x_{j-1}]$ on x_j must be the tightest among all those conditional bounds which consider only one constraint at a time. Therefore, the desired conditional lower and upper bounds are

$$L_j|x_{j-1} \equiv \lceil \max_i \{L_{ij}|x_{j-1}\} \rceil \tag{7}$$

and

$$U_j|x_{j-1} \equiv \lfloor \min_i \{U_{ij}|x_{j-1}\} \rfloor. \tag{8}$$

When we find $L_j|x_{j-1} > U_j|x_{j-1}$, there are no feasible completions of the partial solution $(x_1, ..., x_{j-1})$. In this case, we backtrack to the next value of x_{j-1}. The bounds (7) and (8) are vey similar to those defined in a lemma by P. Krolak which are central to his Bounded Variable Algorithm [18].

At first glance, it may appear that calculating $L_{ij}|x_{j-1}$ and $U_{ij}|x_{j-1}$ each time x_{j-1} changes is computationally prohibitive, but actually it is not. This is because $L_{ij}|x_{j-1}$ and $U_{ij}|x_{j-1}$ both change in a predictable way as x_{j-1} changes. In fact, each is a linear function with a slope that can be calculated once at the outset of the Exact Ceiling Point Algorithm, as we now demonstrate. To see the effect of changing x_{j-1}, let v_{j-1} be its current value and v'_{j-1} the value to which it is then changed, so $v'_{j-1} = v_{j-1} + \delta_{j-1}$, where $\delta_{j-1} \in \{-1, +1\}$. (Our rule is to set $\delta_{j-1} = -1$ if $c_{j-1} \geq 0$, or set $\delta_{j-1} = +1$ if $c_{j-1} < 0$, implying that we start at the more attractive end of the interval defined by the unconditional bounds.) Let $L_{ij}|x'_{j-1}$ and $U_{ij}|x'_{j-1}$ denote the lower and upper conditional bounds, respectively, on x_j when considering constraint (i) alone after fixing variables $(x_1, ..., x_{j-2}, x_{j-1})$ to $(v_1, ..., v_{j-2}, v'_{j-1})$. Similarly, let $g_{ij}|x_{j-1}$ denote the quantity g_{ij} given the partial solution $(v_1, ..., v_{j-2}, v_{j-1})$ and $g_{ij}|x'_{j-1}$ the quantity g_{ij} given the partial solution $(v_1, ..., v_{j-2}, v'_{j-1})$. Assuming that $a_{ij} > 0$, we are interested in how $U_{ij}|x_{j-1}$ differs from $U_{ij}|x'_{j-1}$:

$$\begin{aligned} U_{ij}|x'_{j-1} - U_{ij}|x_{j-1} &= [(g_{ij}|x'_{j-1} - w_{i,j+1}) - (g_{ij}|x_{j-1} - w_{i,j+1})]/a_{ij} \\ &= (g_{ij}|x'_{j-1} - g_{ij}|x_{j-1})/a_{ij} \\ &= [-a_{i,j-1}v'_{j-1} - (-a_{i,j-1}v_{j-1})]/a_{ij} \\ &= a_{i,j-1}(v_{j-1} - v'_{j-1})/a_{ij} \\ &= f_{ij}(-\delta_{j-1}) \end{aligned}$$

where $f_{ij} \equiv a_{i,j-1}/a_{ij}$. Since both f_{ij} and δ_{j-1} are independent of the value of x_{j-1}, $U_{ij}|x_{j-1}$ is a linear function of x_{j-1} with slope $\pm f_{ij}$. When $a_{ij} < 0$, a similar derivation reveals that $L_{ij}|x_{j-1}$ is also a linear function of x_{j-1} with slope $\pm f_{ij}$. Therefore, after calculating $U_{ij}|x_{j-1}$ and $L_{ij}|x_{j-1}$ for the initial value of x_{j-1} within its conditional bounds, one of the two conditional bounds always changes in an additive fashion while the other remains equal to the value of its unconditional bound:

$$L_{ij}|x'_{j-1} \equiv \begin{cases} L_{ij}|x_{j-1} - \delta_{j-1}f_{ij}, & \text{if } a_{ij} < 0; \\ l_j, & \text{if } a_{ij} \geq 0, \end{cases}$$

and

$$U_{ij}|x'_{j-1} \equiv \begin{cases} u_j, & \text{if } a_{ij} \leq 0; \\ U_{ij}|x_{j-1} - \delta_{j-1}f_{ij}, & \text{if } a_{ij} > 0. \end{cases}$$

4.2. Double Backtracking

In addition to being easy to update, the quantities $L_{ij}|x_{j-1}$ and $U_{ij}|x_{j-1}$ impart an important property to the conditional bounds $L_j|x_{j-1}$ and $U_j|x_{j-1}$. As the maximum over a set of linear functions, $L_j|x_{j-1}$ is a (piecewise linear) convex function of x_{j-1} by a well-known theorem (see [3, Theorem 4.13] for example). Similarly, $U_j|x_{j-1}$ is the minimum over a set of linear functions and therefore is a (piecewise linear) concave function of x_{j-1}. This is illustrated in Figure 4 for the case in which $a_{ij} \geq 0$, for all (i,j), so that the conditional lower bound $L_j|x_{j-1}$ is equal to the unconditional bound l_j.

As x_{j-1} ranges between its conditional bounds $L_{j-1}|x_{j-2}$ and $U_{j-1}|x_{j-2}$, the conditional bounds on x_j, $L_j|x_{j-1}$ and $U_j|x_{j-1}$, may *cross* once, twice or not at all. If we observe that $L_j|x_{j-1}$ exceeds $U_j|x_{j-1}$ for a particular value of x_{j-1}, we can certainly backtrack to

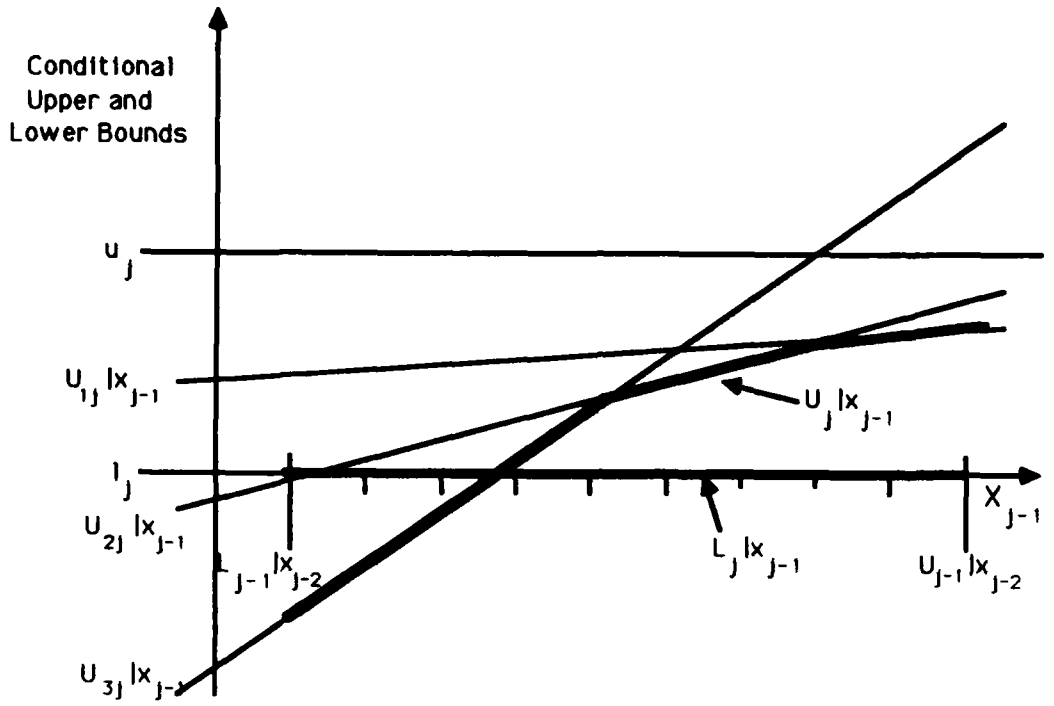

Figure 4. Upper and lower conditional bounds on x_j when all $a_{ij} \geq 0$.

x'_{j-1}, the next value of x_{j-1}, because there are no feasible completions of the partial solution $(x_1, \ldots, x_{j-1})$. But there is also a possibility that $L_j|x_{j-1}$ will exceed $U_j|x_{j-1}$ for all remaining values of x_{j-1}, due to the convexity and concavity properties of the conditional bounds. In this case, we can "double backtrack," *i.e.*, backtrack to the next value of x_{j-2}. The conditions for when it is possible to double backtrack, thereby significantly reducing the amount of enumeration, are given in the next theorem.

Theorem 2. Suppose the backtracking condition $L_j|x_{j-1} > U_j|x_{j-1}$ holds for a particular value of x_{j-1}, implying that there are no feasible completions of the partial solution $(x_1, \ldots, x_{j-1})$. A sufficient condition to double backtrack, *i.e.*, skip over all the remaining integer values of x_{j-1} within its conditional bounds, is that

$$-f_{i_L j}\delta_{j-1} \geq -f_{i_U j}\delta_{j-1}, \tag{9}$$

for some $i_L \in \arg\max_i\{L_{ij}|x_{j-1}\}$ and some $i_U \in \arg\min_i\{U_{ij}|x_{j-1}\}$.

Proof:

$$\begin{aligned} L_j|x'_{j-1} &\geq L_j|x_{j-1} - f_{i_L j}\delta_{j-1}, \text{ where } i_L \in \arg\max_i\{L_{ij}|x_{j-1}\} \\ &> U_j|x_{j-1} - f_{i_L j}\delta_{j-1}, \text{ by the backtracking condition} \\ &\geq U_j|x_{j-1} - f_{i_U j}\delta_{j-1}, \text{ where } i_U \in \arg\min_i\{U_{ij}|x_{j-1}\} \\ &\geq U_j|x'_{j-1}. \end{aligned}$$

The first and last inequalities hold by the convexity and concavity of $L_j|x'_{j-1}$ and $U_j|x'_{j-1}$, respectively, while the third inequality follows if the double backtracking condition stated by the theorem holds. Repeating the argument for subsequent values of x_{j-1}, we find that the lower conditional bound on x_j exceeds the corresponding upper conditional bound not only for x'_{j-1} but for all remaining values of x'_{j-1} within the conditional bounds of x_{j-1}. Thus, we can double backtrack. ∎

Condition (9) is stronger than necessary because there may be some other constraint (i') which is not a member of the $\arg\max_i\{L_{ij}|x_{j-1}\}$ but which is a member of the $\arg\max_i\{L_{ij}|x'_{j-1}\}$ so that $L_j|x'_{j-1} > L_j|x_{j-1} - f_{i_L j}\delta_{j-1}$. However, constraint indices i_L and i_U are readily available, having been identified in the calculation of $L_j|x_{j-1}$ and $U_j|x_{j-1}$, respectively. Note that it only makes sense to check the double backtracking condition after we have found that we can perform an ordinary backtracking step. We must check the condition of Theorem 2 because of the possibility that the conditional bounds may cross twice as x_{j-1} moves between its conditional bounds. If we double backtracked after the first observation of the lower conditional bound exceeding the upper conditional bound, we could miss some partial solutions with feasible completions. Double backtracking seems to be most helpful on problems in which the unconditional bounds are fairly wide for one or more variables, such as the fixed-charge problems discussed in Section 9.

4.3. When a New Incumbent is Found

The third feature of our Exact Algorithm which distinguishes it from ordinary enumeration is manifest when a feasible solution is completed. Because all constraints are taken into account in the calculation of the conditional variable bounds, including an

objective function constraint, any completed solution is not only feasible but also has a strictly superior objective function value to that of the current incumbent. As such, it becomes the new incumbent solution. At this stage, the intersection points between the extreme rays emanating from $\bar{x}$ and the new objective function constraint hyperplane (which define the n-simplex) move closer to $\bar{x}$. Thus, the n-simplex shrinks and a tighter set of integer component bounds defining the $SHR(S)$ may result. If so, the unconditional variable bounds developed for the current search constraint (i) will also tighten, that is, move closer together. Consequently, the minimal completion values $w_{i,j}$ defined in (3) will change, possibly leading to tighter conditional variable bounds. Thus, as we backtrack from the completed solution to new partial solutions, new conditional variable bounds are computed which reflect the improved objective value of the incumbent. The magnitude of the change in the objective function value determines the extent to which these modified conditional bounds speed up the enumeration process.

5. Overview of the Exact Ceiling Point Algorithm

Having seen how to enumerate 1-ceiling points with respect to a specific constraint, we are now in a position to see where this process fits into the overall Exact Ceiling Point Algorithm described in Figure 5. The iterative part (Step 3) of the Exact Algorithm essentially "branches" on one of the functional constraints (i) and seeks to "fathom" a subregion $SHR(i)$ of the $SHR(S)$ by enumerating implicitly or explicitly all 1-$CP(i)$'s within $SHR(i)$. Suppose that the best feasible integer solution known after having searched this region is x_i whose objective function value is $\underline{z} \equiv c^T x_i$. If the new n-simplex S' is formed by the objective function hyperplane $c^T x = \underline{z} + 1$ and the constraints binding at $\bar{x}$, then S' excludes the best known solution x_i. It is possible, then, that S' contains no integer solutions at all. A sufficient condition for this is if there is some j such that the new upper bound $\overline{S}_j$ is strictly less than the new lower bound $\underline{S}_j$. In this case, the problem has been solved. If not, we replace the previous search constraint (i) by a constraint (i'')

parallel to (i) whose right hand side is $b_{i''} \equiv b_i - \max_j |a_{ij}|$, resulting in a modified problem (ILP') which remains to be searched. The modified relaxation, (LP'_R), is then solved for its optimal solution and value $(\bar{x}', \bar{z}')$. Note that $\bar{z}' < \bar{z}$ and that $\bar{z}'$ is an upper bound for the optimal objective function value for (ILP'). Therefore, if $\bar{z}' \leq \underline{z}$, the problem is solved with x_i as an optimal solution for (ILP). If the problem is not solved, a new iteration begins by selecting another search constraint and enumerating its 1-ceiling points. Thus, the algorithm proceeds in a "search and cut" fashion until the problem is solved. (Step 2 will be described in the next section.)

Step

0. a. Solve $(LP_R) \Rightarrow (\bar{x}, \bar{z} \equiv c^T\bar{x})$.
   b. Apply Heuristic Ceiling Point Algorithm $\Rightarrow (x_H, \underline{z} \equiv c^T x_H)$.
1. Construct n-simplex S and associated search hyperrectangle $SHR(S)$.
2. a. Construct Intersection Cut (I) and search hyperrectangle $SHR(I)$.
   b. Enumerate feasible 1-$CP(I)$'s within $SHR(I)$. $\Rightarrow (x_I, \underline{z} \equiv c^T x_I)$.
   c. Reduce $\overline{z}$ to $\max_k\{c^T r^k\}$, where $r^k \equiv$ intersection of k^{th} extreme ray and (I).
   d. Stop if $\underline{z} = \overline{z}$; otherwise,
3. REPEAT
   a. Select a search constraint (i) and construct $SHR(i)$.
   b. Systematically enumerate feasible 1-CP(i)'s $\Rightarrow (x_i, \underline{z} \equiv c^T x_i)$.
   c. Replace $a_i^T x \leq b_i$ with $a_i^T x \leq b_i - \max_j |a_{ij}|$.
   d. Resolve (LP'_R) to obtain $(\bar{x}', \bar{z}' \equiv c^T\bar{x}')$.

   UNTIL OPTIMAL ($FR = \emptyset$ or $\underline{z} \geq \bar{z}'$).

Figure 5. Outline of the Exact Ceiling Point Algorithm.

Theorem 3. Suppose the set of feasible solutions for (ILP) is non-empty and bounded (Assumption 1), $\bar{x}$ is unique (Assumption 3) and $\underline{z} > -\infty$. Then the iterative procedure (Step 3) of the Exact Ceiling Point Algorithm given in Figure 5 is guaranteed to find an optimal solution for (ILP) in a finite number of steps.

Proof: By Theorem 2 of [23] and Theorem 1 above, the total region T which needs to be examined in order to solve the problem is the portion of the n-simplex S which contains all of its 1-$CP(FR)$'s:

$$T \equiv \bigcup_i \{x \geq 0 |\ b_i - \max_j |a_{ij}| < a_i^T x \leq b_i,\ \ c^T x \geq \underline{z}\}.$$

Since T consists of subsets of S, T itself is bounded. There are only a finite number of constraints, each of which is searched at most once for its 1-ceiling points. What remains to be shown is that the scheme of Section 4 for enumerating 1-$CP(i)$'s with respect to a specific constraint (i) is finite.

For a given constraint (i), a search hyperrectangle $SHR(i)$ is defined and then examined for 1-$CP(i)$'s. Being a subset of T, this search hyperrectangle $SHR(i)$ is bounded, implying that all of the corresponding unconditional variable bounds $\{[l_j, u_j], j = 1, \ldots, n\}$ are bounded. The *conditional* variable bounds $\{[L_j|x_{j-1}, U_j|x_{j-1}], j = 1, \ldots, n\}$ are also bounded since, by definition, $L_j|x_{j-1} \geq l_j$ and $U_j|x_{j-1} \leq u_j$ for all j. Thus, while the total number of integer solutions contained within $SHR(i)$ may be large, it is finite. Finally, the scheme which enumerates integer solutions within $SHR(i)$ examines at most once each complete solution or partial solution leading to one or more complete solutions. Partial solutions are fathomed if they are shown to be unable to lead to a feasible completion that is better than the incumbent. Any completed solution is fathomed by virtue of becoming the new incumbent. Thus, the scheme of Section 4 is finite. ∎

The choice of a search constraint (i) is the subject of Section 7. However, before searching any of the constraints binding at $\bar{x}$ for their 1-ceiling points, a preliminary search is made for the 1-ceiling points with respect to a specially-constructed constraint called an intersection cut, which is described in the next section.

6. Searching the Intersection Cut

The intersection cuts employed in the Exact Ceiling Point Algorithm were developed by Balas [4] as a class of cutting planes for solving integer programming problems. They possess the usual feature of chopping off the optimal (LP_R) solution without cutting off any feasible integer solutions for (ILP). Furthermore, they make use of the same structural information from the (LP_R) as do the Ceiling Point Algorithms described here. Our interest in these cuts stems not from their ability to solve (ILP)'s as part of an iterative cutting plane procedure, but from their ability to define a region near $\bar{x}$ which is likely to contain a near-optimal or even optimal feasible integer solution. Thus, an intersection cut is introduced into (ILP) after the Heuristic Ceiling Point Algorithm has been executed. A suitable region associated with this intersection cut is searched for its 1-ceiling points as a heuristic step that uses the framework of the exact enumeration scheme described in Section 4.

The spherical intersection cut introduced in [4] is derived from a convex set denoted $HS(UHC[\bar{x}])$, the hypersphere circumscribing $UHC[\bar{x}]$, and is illustrated in Figure 6. A companion article [5] describes another intersection cut derived from a larger region called the dual to the unit hypercube, denoted $DUHC[\bar{x}]$. One cut or the other is employed in the Exact Ceiling Point Algorithm, depending upon the size of the problem. The intersection cut, denoted by (I), plays a role similar to that of the functional search constraints described previously. We first define a search hyperrectangle for the intersection cut, denoted $SHR(I)$, and proceed to enumerate its feasible 1-ceiling points. The only difference from the case of ordinary search constraints is that our $SHR(I)$ is not so large as to contain all of the 1-ceiling points with respect to (I), but only the ones most likely to be feasible.

The search region $SHR(I)$ depends upon a set of points $\{r^k,\ k = 1,2,\ldots\}$ located along the extreme rays of the feasible region. Each r^k represents the intersection of the k^{th} extreme ray and the $HS(UHC[\bar{x}])$. The intersection cut is then constructed so as to pass through this set of points. These points also provide an upper bound $\bar{z}$ on the optimal objective function value for (ILP): $\bar{z} = \max_k\{c^T r^k\}$. The point r^k has the form

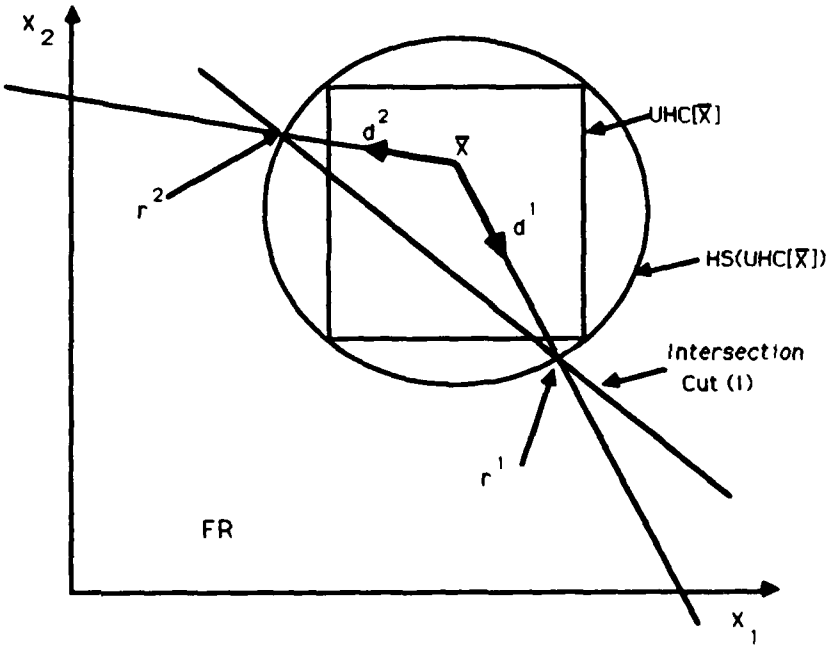

Figure 6. An Intersection Cut derived from $HS(UHC[\bar{x}])$.

$\bar{x} + \bar{\lambda}^k d^k$, where $\bar{\lambda}^k$ is defined such that $\bar{x} + \bar{\lambda}^k d^k$ lies on the surface of $HS(UHC[\bar{x}])$. Precise expressions for the $\bar{\lambda}^k$ are given in [4] and [5]. Each of these non-integer points r^k is rounded to an integer solution $\bar{r}^k$ (a vertex of $UHC[r^k]$) which is feasible with respect to the intersection cut.

The search hyperrectangle $SHR(I)$ is then determined as a set of lower and upper bounds $\{[\underline{I}_j, \overline{I}_j], j = 1, \ldots, n\}$ by finding the minimum and maximum, respectively, over the j^{th} component of all the $\bar{r}^k$'s. As with the bounds defining $SHR(i)$, the bounds defining $SHR(I)$ should be no wider than those defining $SHR(S)$ since searching beyond the boundary of $SHR(S)$ cannot improve upon the incumbent. More precisely,

$$\underline{I}_j \equiv \min\{\min_k\{\bar{r}_j^k\}, \underline{S}_j\}, \text{ for } j = 1, \ldots, n,$$

and

$$\overline{I}_j \equiv \max\{\max_k\{\bar{r}_j^k\}, \overline{S}_j\}, \text{ for } j = 1, \ldots, n.$$

If, for any component j, we find $\overline{I}_j < \underline{I}_j$, then $SHR(I) = \emptyset$, i.e., there are no integer solutions at all in this search hyperrectangle. In this case, the exact algorithm proceeds to search for 1-ceiling points with respect to an original functional constraint. Assuming this is not the case, we enumerate the integer solutions contained within the search hyperrectangle $SHR(I)$ in the same manner as described in Section 4 for an ordinary search constraint (i).

A nice feature about the intersection cut is that the volume of $SHR(I)$ reflects the shape of the feasible region near $\bar{x}$. When the cone $\overline{FR}$ with vertex $\bar{x}$ is narrow, the search hyperrectangle $SHR(I)$ is apt to contain a relatively small number of integer solutions (feasible or not) and so not much time will be wasted searching an unpromising region. On the other hand, when the cone $\overline{FR}$ is fairly wide, $SHR(I)$ is apt to contain a relatively large number of integer solutions and the subsequent search will be thorough in a part of the feasible region with a good chance of containing an optimal solution.

7. Choice of Search Constraint

We now describe our rule for how a search constraint is selected in the Exact Algorithm. As background, note that once constraint (i) has been chosen and the "band" or volume of the feasible region between (i) and (i') has been exhaustively searched for 1-$CP(i)$'s, constraint (i) may be replaced by

$$a_i^T x \leq b_i - \max_j |a_{ij}| \qquad (i'')$$

for purposes of continuing the search elsewhere in the feasible region. Therefore, when this stage in the algorithm is reached, (LP_R) should be resolved with b_i replaced by $b_{i''} = b_i - \max_j |a_{ij}|$, leading to a reduction in the optimal objective function value of (LP_R). Let $(LP'_R)_i$ denote this new but related (LP_R) whose optimal solution value is $\bar{z}'_i$. Recall that $\bar{z}$ is the optimal objective function value of (LP_R) and serves as an upper bound on the optimal objective function value of (ILP). To estimate the extent to which $\bar{z}$ and $\bar{z}'_i$ differ without actually solving $(LP'_R)_i$, let π_i be the shadow price of binding constraint

(i) for (LP_R), where this shadow price measures the rate at which the objective function $\bar{z}$ changes as b_i is changed by a small amount. If $\Delta b_i \equiv b_i - b_{i''} = \max_j |a_{ij}|$, then $\bar{z} - \bar{z}'_i \geq \pi_i \Delta b_i$. The inequality is necessary because constraint (i'') may not be binding at the optimal solution to $(LP'_R)_i$. However, we approximate $\bar{z}'_i$ by $\bar{z}''_i \equiv \bar{z} - \pi_i \Delta b_i$.

Now let $\bar{A} \equiv \{i |\ a_i^T \bar{x} = b_i\}$ be the set of constraints binding at $\bar{x}$ and let $\bar{A}' \equiv \{i \in \bar{A} |\ \bar{z}'_i \leq \underline{z}\}$, where $\underline{z}$ is the objective function value of the incumbent. Thus, $\bar{A}'$ includes any given binding constraint only if fathoming that constraint would indicate that the problem has been solved because $\underline{z}$ is at least as large as the objective function value of any solution in the remaining (ILP). Our rule for choosing a search constraint is to select the binding constraint (h) such that

$$h \in \arg\min_{i \in \bar{A}} \{\bar{z}''_i\}.$$

The motivation for this rule is that fathoming constraint (h), *i.e.*, exhaustively searching $SHR(h)$ for 1-$CP(h)$'s, gives the largest guaranteed reduction in $\bar{z}$. If, in addition, $h \in \bar{A}'$, then fathoming constraint (h) may entail more searching than is absolutely necessary to solve the problem as a result of the ceiling point constraint (h'') associated with (h) cutting too deeply into the feasible region. To prevent any unnecessary searching, the right hand side $b_{h''}$ for ceiling point constraint (h'') is modified (assuming that all c_j are integer) so that $\bar{z}''_h = \underline{z} + 1$:

$$b_{h''} = b_h - (\bar{z} - (\underline{z} + 1))/\pi_h.$$

In this case, we will be seeking d-$CP(h)$'s, where $d \geq 1$. On any subsequent iterations, constraint (h'') would not be considered for selection as the search constraint.

Although similar, this rule for choosing a search constraint and that proposed for the Heuristic Ceiling Point Algorithm in [24, Section 3.1] may select different search constraints. The rule presented earlier places more emphasis on how the objective function value changes over just the feasible portion of the constraint. Here, the emphasis is on the effect of fathoming a particular constraint and its associated search hyperrectangle $SHR(i)$; the selected constraint is that which guarantees the largest decrease in $\bar{z}$, and hence, the largest decrease in the value of the upper bound on the optimal objective function value of the remaining (ILP). This emphasis increases the likelihood that only a

small number of constraints must be searched for their 1-ceiling points before the problem is solved. It is also clear that the better the quality of the solution that the Exact Ceiling Point Algorithm starts with, the fewer iterations it is likely to require to solve the problem.

8. Other Computational Issues

During the computational testing of the Exact Ceiling Point Algorithm it was often noticed that the same number of integer solutions were contained within the search hyperrectangles $SHR(i)$ and $SHR(S)$, *i.e.*, $\prod_j(1 + u_j - l_j) = \prod_j(1 + \overline{S}_j - \underline{S}_j)$, probably indicating that the two hyperrectangles coincided. This seemed to imply that either the current solution was very close to being optimal, leading to a relatively small $SHR(S)$, or that the distance between the constraint hyperplanes (i) and (i') was fairly wide, thereby generating a relatively large $SHR(i)$, or a combination of both. Under these circumstances it seemed wise to simply search the hyperrectangle associated with the n-simplex, $SHR(S)$, for any feasible integer solution and not search only for feasible 1-$CP(i)$'s. Any such solution found at this stage would mostly likely be a feasible 1-ceiling point with respect to either (i) or some other constraint binding at $\bar{x}$. Furthermore, the algorithm can be terminated once all solutions within the n-simplex have been enumerated.

9. Computational Experience

This section presents our computational experience with the Exact Ceiling Point Algorithm, using the relevant parts of [8] as a guide to reporting our results. Having already described our algorithmic approach, we begin in the next subsection by describing its computer-based implementation. Subsection 9.2 discusses the experimental design, including the objective of the experiment, the origin of all of the test problems used and the choice of performance indicators. Computational results with the Exact Ceiling Point

Algorithm are reported in subsection 9.3.

9.1. Computer Implementation

Computational testing of the algorithms was performed on a Digital Equipment Corporation VaxStation II with ten megabytes of main memory, under the MicroVMS operating system, version 4.5. All of the code was written in Fortran and compiled with the VAX Fortran Compiler, version 4.5, using the default settings that include an optimizer. Real variables were declared as double precision variables. Groups of test problems were submitted as a batch job in order to maintain consistent timing results.

A clock-reading routine due to [20] returning CPU time in centiseconds was employed to establish execution times of various parts of the code. Thus, execution times reported for the Ceiling Point Algorithms are accurate to at most 0.01 CPU seconds. However, it is felt that this relatively small uncertainty in the timing can be safely ignored in the following analysis. All execution times are given in CPU seconds. Those execution times that apply specifically to the Exact Ceiling Point Algorithm include the time required to read in the data but not to write out any information; those reported for other algorithms may or may not include input/output time.

9.2. Experimental Design

The main objective of our computational testing was to assess whether or not the methods for enumerating 1-ceiling points described in this report constitute a practical approach for exactly solving general integer linear programming problems. To assess the effectiveness of the Exact Ceiling Point Algorithm, we shall compare its performance to those of other algorithms on a common set of test problems. It should be emphasized that these computational results provide only a general indication of an algorithm's performance rather than conclusive evidence because not only are we examining performance based on a

relatively limited amount of computational experience, but also the algorithms have been coded by different authors, run on computers of different generations and sizes, and so forth.

The 48 test problems taken from the literature have been grouped into two categories: "realistic" (because these problems were drawn from real applications) and "randomly generated" (because the parameters of these problems were randomly generated). Characteristics of the sets of realistic and random test problems are shown in Tables I(a) and I(b), respectively. The first two columns of each table give the size of the constraint matrix (rows by columns) and the name by which we shall refer to each problem. Density is simply the percentage of coefficients of the constraint matrix which are nonzero. A negative entry in the column of optimal objective function values indicates that the problem originally was in the form of a minimization rather than a maximization. The last two columns provide two measures of the distance between the optimal objective function values for (ILP) and (LP_R). The first measure is the *normalized duality gap*,

$$D(\bar{x}, x^*) \equiv (c^T\bar{x} - c^T x^*)/\|c\|_2,$$

where $\|c\|_2$ is the Euclidean norm of c. This quantity measures the Euclidean distance between the optimal objective function hyperplanes for (ILP) and (LP_R), *i.e.*, between $c^T x = \bar{z}$ and $c^T x = z^*$. It is a reasonably good guide for indicating the difficulty of the problem: the larger the normalized duality gap, generally the more difficult it is to find an optimal integer solution and prove its optimality. The second measure is the duality gap in percentage terms, $100 \times (c^T\bar{x} - c^T x^*)/c^T\bar{x}$, which provides some perspective on the importance of actually finding an optimal integer solution for a particular problem once a good feasible solution has been discovered. An integer solution found to be within some small percentage of the optimal LP-relaxation objective function value may be "close enough" for all practical purposes.

All 24 of the realistic problems appeared in the study by Trauth and Woolsey [27]. These consist of ten fixed-charge problems, {FC-1, FC-2,..., FC-10}, five of the IBM test problems, {IBM-1, IBM-2,..., IBM-5}, and nine allocation problems, {AL-55, AL-60,..., AL-100}. The set of allocation problems are all the same 0-1 knapsack problem except

Table I(a). Realistic Test Problem Characteristics.

			Optimal Value for		Duality Gap	
$m \times n$	Problem	Density	$LP_R : \bar{z}$	$ILP : z^*$	Norm.[a]	Pct. [b]
4 × 5	FC-1	70	8.79	7	1.032	20.5
4 × 5	FC-2	70	9.61	8	0.931	16.7
4 × 5	FC-3	70	11.81	10	1.046	15.3
4 × 5	FC-4	70	9.22	8	0.704	13.0
6 × 5	FC-5	53	88.61	76	7.279	14.2
6 × 5	FC-6	53	118.13	106	7.003	10.2
4 × 5	FC-7	70	88.61	76	7.279	14.2
4 × 5	FC-8	70	118.13	106	7.003	10.2
6 × 6	FC-9	50	12.00	9	1.732	25.0
10 × 12	FC-10	50	18.71	17	0.698	9.1
7 × 7	IBM-1	57	-7.50	-8	0.189	6.7
7 × 7	IBM-2	57	-5.75	-7	0.472	20.7
3 × 4	IBM-3	100	-179.78	-187	0.271	4.0
15 × 15	IBM-4	53	-9.25	-10	0.194	7.5
15 × 15	IBM-5	53	-12.88	-15	0.549	16.3
11 × 10	AL-55	18	50.30	50	0.008	0.6
11 × 10	AL-60	18	54.50	52	0.063	4.6
11 × 10	AL-65	18	58.67	57	0.042	2.9
11 × 10	AL-70	18	62.83	62	0.021	1.3
11 × 10	AL-75	18	67.00	67	0.000	0.0
11 × 10	AL-80	18	70.60	68	0.065	3.7
11 × 10	AL-85	18	74.20	70	0.105	5.7
11 × 10	AL-90	18	77.80	75	0.070	3.6
11 × 10	AL-100	18	85.00	85	0.000	0.0

[a] Gives the normalized duality gap: $D(\bar{x}, x^*) \equiv (c^T\bar{x} - c^Tx^*)/||c||_2$.
[b] Gives the duality gap in % terms: $100 \times (c^T\bar{x} - c^Tx^*)/c^T\bar{x}$.

Table I(b). Randomly Generated Test Problem Characteristics.

			Optimal Value for		Duality Gap	
$m \times n$	Problem	Density	$LP_R: \bar{z}$	$ILP: z^*$	Norm.[a]	Pct. [b]
15 × 15	I-1	100	2956.1	2893	0.384	2.1
15 × 15	I-2	100	2650.8	2570	0.573	3.0
15 × 15	I-5	100	6356.0	6171	1.117	2.9
15 × 15	I-6	100	2289.1	2234	0.333	2.4
15 × 15	II-1	100	1896.3	1875	0.091	1.1
15 × 15	II-2	100	1758.8	1725	0.178	1.9
15 × 15	II-3	100	2029.9	1983	0.189	2.3
15 × 15	II-4	100	2478.0	2429	0.220	2.0
15 × 15	II-5	100	1574.8	1558	0.079	1.1
15 × 15	II-6	100	1575.5	1556	0.083	1.2
15 × 15	II-7	100	2088.0	2056	0.147	1.5
15 × 15	II-8	100	1592.8	1548	0.199	2.8
15 × 15	II-9	100	1756.8	1743	0.063	0.8
15 × 15	II-10	100	1764.7	1734	0.137	1.8
30 × 15	II-11	100	1522.5	1491	0.129	2.1
30 × 15	II-12	100	1449.9	1424	0.138	1.8
15 × 30	II-13	100	1811.6	1785	0.092	1.5
15 × 30	II-14	100	2337.4	2309	0.089	1.2
6 × 21	II-M	100	643.0	594	0.181	7.6
15 × 15	III-2	53	110.7	99	0.055	10.6
15 × 15	III-3	48	144.5	130	0.061	10.0
15 × 15	III-4	50	124.3	92	0.157	27.6
15 × 15	III-5	45	119.5	97	0.097	18.8
15 × 15	III-8	49	123.3	113	0.054	8.4

[a] Gives the normalized duality gap: $D(\bar{x}, x^*) \equiv (\bar{z} - z^*)/||c||_2$.

[b] Gives the duality gap in % terms: $100 \times (\bar{z} - z^*)/\bar{z}$.

that the right hand side increases from 55 to 100. It should be noted that the LP-relaxations associated with two of the allocation problems, AL-75 and AL-100, possess an all-integer optimal solution. Thus, AL-75 and AL-100 are solved immediately by the Heuristic Ceiling Point Algorithm but are included in this study in order to compare our results more completely with those reported elsewhere. The fixed-charge and IBM problems were first presented in [13] and, though small, are "hard" to solve in the sense that the optimal solutions for (ILP) and (LP_R) are relatively far apart, as indicated by large values of the normalized duality gap. Characteristic of the fixed-charge problems is that simple rounding of $\bar{x}$ almost never yields a feasible integer solution.

With one exception, all 24 of the problems with randomly generated coefficients have been taken from Hillier's study [16] and are fully specified in [17]. These problems are labeled as {I-1, I-2, I-5, I-6}, {II-1, II-2,..., II-14} and {III-2,..., III-5, III-8}. Their integer coefficients were generated from a uniform distribution over the intervals shown in the Table II. The one additional problem (labeled "II-M") is similar to a Type II problem except that the b_i's are smaller. Originally proposed as a 0-1 problem in [21], II-M was solved as a general integer problem in [19], as it is here.

Table II. Coefficient Ranges for Randomly Generated Test Problems.

	Problem Type		
	I	II	III
c_j	[-20, 79]	[0, 99]	[0, 99]
a_{ij}	[-40, 59]	[0, 99]	[0, 1]
b_i	[500,999]	[1000,1999]	1
x_j	general	general	binary

With large values of the right hand sides (b_i's) and with constraint matrices which are essentially 100 percent dense, the Type I and Type II problems are not easy to solve. The Type I problems are especially tough because approximately 40 percent of their constraint coefficients are negative, while the other 60 percent are positive. The Type III problems, on the other hand, are not particularly challenging. As shown in Table I(b), the normalized

duality gap for a typical Type I problem is roughly two to three times as large as that for an average Type II problem which, in turn, is about twice that for a Type III problem.

For algorithms which solve the (LP_R) associated with (ILP), an alternative to simply reporting CPU time is to examine the ratio of total CPU time to CPU time required to solve the LP-relaxation. This ratio gives an idea of how much work is required by the entire algorithm in relation to a relatively efficient and dependable algorithm (the simplex method) used in the first phase of the algorithm to solve (LP_R). It also provides a crude basis of comparison for LP-based algorithms which perhaps have been coded in different languages and/or tested on different types of computers. With this measure, various algorithms' execution times are normalized by the amount of time to solve (LP_R). It must be emphasized, however, that the LP solvers embedded within the respective integer programming algorithms may have been designed and implemented quite differently, causing such comparisons to be rather rough.

In evaluating exact algorithms for (ILP), the most important performance indicator would seem to be the total CPU time since all algorithms being compared presumably find an optimal solution. It may also be meaningful to express the total CPU time as a fraction of the time required to solve the associated (LP_R). In contrast to some other areas of mathematical programming, such as nonlinear programming, the numerical accuracy of an optimal solution found by the Exact Ceiling Point Algorithm is not really a subject of much concern since it works only with integer solutions.

9.3. Results with the Exact Ceiling Point Algorithm

Tables 6-III(a) and (b) indicate the relative performance of the various phases of the Exact Ceiling Point Algorithm on the realistic and randomly generated test problems, respectively. The columns labeled "LP_R," "Heur.", "I-Cut," and "Exact" give the fraction of the total CPU time spent in the various components of the algorithm: solving the linear programming relaxation associated with (ILP), running Phases 2 and 3 of the Heuristic Ceiling Point Algorithm, searching the region defined by the Intersection Cut (see Section

Table 6-III(a). Execution Times of the Exact Ceiling Point Algorithm on Realistic Problems.

	% of Total CPU time				Total	
Problem	LP_R	Heur.	I-Cut	Exact	CPU time	Ratio[a]
FC-1	50.0	25.0	10.7	14.3	0.28	2.0
FC-2	69.6	26.1	4.3	0.0	0.23	1.4
FC-3	56.5	13.0	13.0	17.4	0.23	1.8
FC-4	76.5	17.6	5.9	0.0	0.17	1.3
FC-5	22.1	23.5	2.9	51.5	0.68	4.5
FC-6	25.4	23.7	3.4	47.5	0.59	3.9
FC-7	21.7	25.0	3.3	50.0	0.60	4.6
FC-8	27.3	25.4	1.8	45.5	0.55	3.7
FC-9	51.5	15.1	6.1	27.3	0.33	1.9
FC-10	25.0	51.7	14.7	8.6	1.16	4.0
IBM-1	53.8	28.2	18.0	0.0	0.39	1.9
IBM-2	55.0	40.0	5.0	0.0	0.40	1.8
IBM-3	60.0	28.0	8.0	4.0	0.25	1.7
IBM-4	24.2	75.8	0.0	0.0	2.69	4.1
IBM-5	1.3	4.1	6.4	88.2	47.10	77.2
AL-55	27.5	72.5	0.0	0.0	1.09	3.6
AL-60	51.8	28.6	8.9	10.7	0.56	1.9
AL-65	41.1	46.6	5.5	6.8	0.73	2.4
AL-70	43.3	56.7	0.0	0.0	0.67	2.3
AL-75	100.0	0.0	0.0	0.0	0.28	1.0
AL-80	39.7	45.2	8.2	6.9	0.73	2.5
AL-85	39.5	44.7	7.9	7.9	0.76	2.5
AL-90	38.9	47.2	6.9	6.9	0.72	2.6
AL-100	100.0	0.0	0.0	0.0	0.29	1.0

[a] Ratio = (Total CPU time)/(CPU time solving LP_R)

Table 6-III(b). Execution Times of the Exact Ceiling Point Algorithm on Randomly Generated Problems.

	% of Total CPU time				Total	
Problem	LP_R	Heur.	I-Cut	Exact	CPU time	Ratio
I-1	0.1	0.3	0.2	99.4	348.49	645.4
I-2	1.1	7.2	1.6	90.1	40.37	85.9
I-5	<0.1	<0.4	<0.1	>99.2	>900	>2000
I-6	<0.1	<0.2	<0.1	>99.6	>900	>1837
II-1	20.7	46.5	12.7	20.2	2.13	4.8
II-2	21.6	27.9	10.6	39.9	2.08	4.6
II-3	7.4	10.6	3.1	78.9	6.79	13.6
II-4	10.3	17.5	5.0	67.2	3.99	9.7
II-5	28.5	53.5	11.1	6.9	1.44	3.5
II-6	1.5	6.7	0.6	91.2	28.49	66.3
II-7	1.8	2.6	1.2	94.4	23.35	54.3
II-8	0.4	0.5	2.2	96.9	116.26	252.7
II-9	22.5	49.3	9.6	18.7	2.09	4.5
II-10	9.0	30.5	6.4	54.0	4.98	11.1
II-11	11.4	42.3	8.8	37.6	6.17	8.8
II-12	0.3	0.6	0.3	98.8	231.50	308.7
II-13	<0.1	<0.3	<0.2	>99.4	>900	>1154
II-14	2.7	23.4	4.4	21.2	30.48	37.2
II-M	2.2	5.0	8.0	84.8	13.13	45.3
III-2	35.8	37.2	18.3	8.8	1.37	2.8
III-3	29.6	49.3	13.2	7.9	1.52	3.4
III-4	20.5	60.5	12.4	6.5	1.85	4.9
III-5	40.0	31.0	15.0	14.0	1.00	2.5
III-8	38.3	34.0	18.1	9.6	0.94	2.6

Ratio = (Total CPU time)/(CPU time solving LP_R)

6), and executing the iterative portion of the Exact Ceiling Point Algorithm (see Section 5). The next to last column gives the total CPU time in seconds, while the last column gives the ratio of the total execution time to the amount of time solving (LP_R).

Based on both total CPU time and the ratio of total CPU time to time spent solving (LP_R), the Exact Ceiling Point Algorithm appears to be an efficient method for solving all of the realistic test problems except IBM-5. Aside from this one problem, the only problems where more than half the total CPU time was spent outside of the Heuristic Algorithm were {FC-5,..., FC-8}. These four problems are poorly scaled versions of {FC-1,..., FC-4} in the sense that the right hand sides of some of their constraints have been multiplied by a factor of ten, greatly enlarging the feasible region. Furthermore, the normalized duality gap for each of these problems is particularly large. In solving each of these four problems, all four functional constraints binding at $\bar{x}$ had to be searched for 1-ceiling points. Thus, more time was spent in the iterative portion of the exact algorithm on these problems than in the other problems. As can be seen in Table 6-IV(a), the Intersection Cut search proved ineffective on all problems except FC-10, another indication of the difficulty of the fixed-charge test problems.

Table 6-III(b) indicates that the performance of the Exact Ceiling Point Algorithm on the randomly generated problems varied noticeably with the type of problem. This is partly due to the fact that the same is true of the Heuristic Ceiling Point Algorithm, which provides the Exact Algorithm with an initial feasible integer solution. The better the objective function value of this initial solution, the smaller the size of the region examined by the Exact Ceiling Point Algorithm, and hence, the quicker it solves the problem.

On all of the Type III problems, the Heuristic Algorithm identified an optimal solution and the Exact Algorithm was able to prove optimality rather quickly. Of the fifteen Type II problems, the Heuristic Algorithm located an optimal solution in six cases. On the remaining nine Type II problems, the Intersection Cut search was effective in locating a better solution four times, one of which was optimal. While only moderately successful, the Intersection Cut search is cheap enough computationally to make it worthwhile. The iterative portion of the Exact Ceiling Point Algorithm was fairly successful in solving the majority of Type II problems reasonably quickly. However, it was quite slow in finishing

Table 6-IV(a). Raw Data for the Exact Ceiling Point Algorithm's Performance on Realistic Problems.

	LP_R+Heur.		I-Cut		Exact		Total
Problem	CPU	$\underline{z}$	CPU	$\underline{z}$	CPU	z^*	CPU
FC-1	0.21	7	0.03	7	0.04	7	0.28
FC-2	0.22	8	0.01	8	0.00	8	0.23
FC-3	0.16	10	0.03	10	0.04	10	0.23
FC-4	0.16	8	0.01	8	0.00	8	0.17
FC-5	0.31	75	0.02	75	0.35	76	0.68
FC-6	0.29	105	0.02	105	0.28	106	0.59
FC-7	0.28	75	0.02	75	0.30	76	0.60
FC-8	0.29	105	0.01	105	0.25	106	0.55
FC-9	0.22	9	0.02	9	0.09	9	0.33
FC-10	0.89	15	0.17	17	0.10	17	1.16
IBM-1	0.32	-9	0.07	-8	0.00	-8	0.39
IBM-2	0.38	-7	0.02	-7	0.00	-7	0.40
IBM-3	0.22	-187	0.02	-187	0.01	-187	0.25
IBM-4	2.69	-10	0.00	-10	0.00	-10	2.69
IBM-5	2.56	-15	3.00	-15	41.54	-15	47.10
AL-55	1.09	50	0.00	50	0.00	50	1.09
AL-60	0.45	52	0.05	52	0.06	52	0.56
AL-65	0.64	57	0.04	57	0.05	57	0.73
AL-70	0.67	62	0.00	62	0.00	62	0.67
AL-75	0.28	67	0.00	67	0.00	67	0.28
AL-80	0.62	68	0.06	68	0.05	68	0.73
AL-85	0.64	70	0.06	70	0.06	70	0.76
AL-90	0.62	75	0.05	75	0.05	75	0.72
AL-100	0.29	85	0.00	85	0.00	85	0.29

Table 6-IV(b). Raw Data for the Exact Ceiling Point Algorithm's Performance on Randomly Generated Problems.

	LP_R+Heur.		I-Cut		Exact		Total
Prob	CPU	$\underline{z}$	CPU	$\underline{z}$	CPU	z^*	CPU
I-1	1.64	2807	0.55	2893	346.30	2893	348.49
I-2	3.37	2507	0.64	2565	36.36	2570	40.37
I-5	4.04	6007	0.39	6007	>896	≥6007	>900
I-6	2.22	2231	0.43	2231	>898	≥2231	>900
II-1	1.43	1835	0.27	1864	0.43	1875	2.13
II-2	1.03	1725	0.22	1725	0.83	1725	2.08
II-3	1.22	1983	0.21	1983	5.36	1983	6.79
II-4	1.11	2426	0.20	2426	2.68	2429	3.99
II-5	1.18	1558	0.16	1558	0.10	1558	1.44
II-6	2.33	1556	0.17	1556	25.99	1556	28.49
II-7	1.03	2052	0.28	2052	22.04	2056	23.35
II-8	1.03	1533	2.51	1548	112.72	1548	116.26
II-9	1.50	1735	0.20	1735	0.39	1743	2.09
II-10	1.97	1732	0.32	1732	2.69	1734	4.98
II-11	3.31	1488	0.54	1488	2.32	1491	6.17
II-12	2.18	1406	0.58	1421	228.74	1424	231.50
II-13	3.40	1785	1.49	1785	>896	≥1785	>900
II-14	7.94	2309	1.33	2309	21.21	2309	30.48
II-M	0.95	564	1.05	568	11.13	594	13.13
III-2	1.00	99	0.25	99	0.12	99	1.37
III-3	1.20	130	0.20	130	0.12	130	1.52
III-4	1.50	92	0.23	92	0.12	92	1.85
III-5	0.71	97	0.15	97	0.14	97	1.00
III-8	0.68	113	0.17	113	0.09	113	0.94

off both II-8 and II-12 and failed to prove optimality on the 30-variable problem II-13 within a 900 second execution time limit.

It should be noted that for each of the randomly generated problems, a run-time option was activated which reordered the variables based on the relative magnitudes of their associated cost components. Specifically, the variable with the largest c_j was relabeled as x_1 (the first variable to be fixed), the variable with the second largest cost coefficient relabeled as x_2 (the second variable to be fixed), and so forth. For some of the randomly generated problems, this simple scheme was very effective in reducing the execution time of the enumeration scheme. The execution time of II-12, for example, was reduced to one third of that required when the variables were not reordered.

The last set of tables give the performance of the Exact Ceiling Point Algorithm (XCPA) on the test problems along with the performances of a few other exact algorithms, one of which is contained within a widely available package called the Generalized Algebraic Modeling System (GAMS, Version 2.04) developed by Brooke, Kendrick and Meeraus [6]. When faced with a mixed integer linear programming problem, GAMS calls upon the Zero/One Optimization Methods (ZOOM/XMP, Version 2.0) developed by Roy Marsten. In brief, ZOOM converts every (bounded) general integer variable into a sum of binary variables and applies the Pivot & Complement heuristic device of Balas and Martin [7] to find an initial solution. It then proceeds with an LP-based branch-and-bound scheme. Fairly tight upper bounds on the variables were specified in order to keep the number of binary variables relatively small. These are given in Appendix A, along with values of the GAMS/ZOOM run-time options that we specified.

A blank entry in a table indicates that no time was reported for that algorithm on that problem. Only the Exact Ceiling Point Algorithm and GAMS/ZOOM were executed on the same computer (a VaxStation II), so it is difficult to directly compare all of the stated execution times. However, in one study [9] of various computers' performances in solving a dense system of linear equations using a standard double precision Fortran package (LINPACK), the IBM-370/168 appeared to be at least five times as fast as the IBM-370/158, and at least seven times as fast as the VAX 11/780. Though the IBM-360/67 and the VaxStation II are not included in this study, a knowledgeable computer

Table 6-V(a). Comparison of Performances by Exact Algorithms on Realistic Problems.

	XCPA		GAMS/ZOOM		[10]		[2]	[14]
	VaxStation II		VaxStation II		IBM 360/67		370/168	7090
Problem	Time	Ratio	Time	Ratio	Time	Ratio	Time	Time
FC-1	0.28	2.0	4.30	14.8	0.32	10.7	0.19	1.83
FC-2	0.23	1.4	3.13	11.6	0.27	9.0	0.19	1.35
FC-3	0.23	1.8	3.41	11.4	0.33	11.0	0.13	1.88
FC-4	0.17	1.3	2.24	7.7	0.28	9.3	0.13	1.48
FC-5	0.68	4.5	8.62	27.8	41.77	464.4	0.18	9.01
FC-6	0.59	3.9	12.68	39.6	22.07	220.7	0.18	7.57
FC-7	0.60	4.6	8.22	24.9	41.77	1392.3	0.09	7.83
FC-8	0.55	3.7	12.05	38.9	21.81	727.0	0.09	6.42
FC-9	0.33	1.9	1.92	5.6	0.43	7.2	0.40	3.23
FC-10	1.16	4.0	13.67	14.9			1.61	9.15
IBM-1	0.39	1.9	3.02	6.9	0.53	6.6		1.87
IBM-2	0.40	1.8	6.46	15.4	0.55	6.9		3.02
IBM-3	0.25	1.7	3.19	11.8	0.21	10.5		2.87
IBM-4	2.69	4.1	33.97	22.8				11.67
IBM-5	47.10	77.2	>900	>671	9.28	15.5		66.48
AL-55	1.09	3.6	1.04	2.7			0.18	2.42
AL-60	0.56	1.9	1.04	2.7			0.19	5.08
AL-65	0.73	2.4	1.26	3.0			0.14	3.90
AL-70	0.67	2.3	1.27	3.2			0.18	2.60
AL-75	0.28	1.0	0.34	1.0			0.15	1.87
AL-80	0.73	2.5	1.02	2.8			0.15	3.87
AL-85	0.76	2.5	1.33	3.2			0.17	7.88
AL-90	0.72	2.6	1.22	3.9			0.16	4.52
AL-100	0.29	1.0	0.31	1.0			0.17	1.87

Ratio = (Total CPU time)/(CPU time solving LP_R)

Table 6-V(b). Comparison of Exact Algorithms on Randomly Generated Problems.

	XCPA		GAMS/ZOOM		[16]		[2]
	VaxStation II		VaxStation II		IBM-360/67		370/168
Problem	Time	Ratio	Time	Ratio	Time	Ratio	Time
I-1	348.49	645.4	429.96	462.3	17.32	19.0	3.18
I-2	40.37	85.9	407.84	261.4	2.94	3.2	1.10
I-5	>900	>2000	>900	>612	19.49	20.5	
I-6	>900	>1837	>900	>967	9.34	7.5	
II-1	2.13	4.8	83.48	132.5	2.68	3.9	0.92
II-2	2.08	4.6	64.92	95.5	2.33	2.9	0.98
II-3	6.79	13.6	79.34	90.2	2.21	3.8	1.20
II-4	3.99	9.7	83.32	119.0	2.28	3.2	1.02
II-5	1.44	3.5	59.30	76.0	1.48	2.4	0.81
II-6	28.49	66.3	78.39	137.5	3.30	5.4	1.16
II-7	23.35	54.3	74.97	88.2	3.22	5.4	1.90
II-8	116.26	252.7	93.45	118.3	4.22	6.2	1.93
II-9	2.09	4.5	47.83	83.9	1.85	2.9	0.68
II-10	4.98	11.1	97.58	112.2	2.53	3.1	0.85
II-11	6.17	8.8			6.48	2.0	
II-12	231.50	308.7			7.17	2.1	
II-13	>900	>1154			162.25	144.9	
II-14	30.48	37.2			7.38	6.7	
II-M	13.13	45.3					
III-2	1.37	2.8	4.40	6.7	1.32	2.4	0.40
III-3	1.52	3.4	2.65	4.1	1.61	2.5	0.55
III-4	1.85	4.9	3.89	7.8	1.60	2.7	0.52
III-5	1.00	2.5	6.50	11.4	1.58	2.8	0.39
III-8	0.94	2.6	5.92	9.6	1.41	2.3	0.45

Ratio = (Total CPU time)/(CPU time solving LP_R)

scientist informed us that the IBM-370/158 is at least as fast as the IBM-360/67, and that the VAX 11/780 is directly comparable to the VaxStation II [25].

For the realistic test problems, Table 6-V(a) compares the Exact Ceiling Point Algorithm, the GAMS/ZOOM package, Faaland and Hillier's Accelerated Bound-and-Scan Algorithm [10] run on an IBM-360/67, Austin and Ghandforoush's Surrogate Cutting Plane Algorithm [2] (an extension of their Reduced Advanced Start Algorithm [1]) run on an IBM-370/168 and Haldi and Isaacson's cutting plane code LIP-1 [14] run on an IBM-7090. (Reasonably good times on a subset of these problems are also reported by [12], but since they do not report anything for the more difficult problems, their limited results were not included here.) Based on the execution times and relative speeds of the computers, the Exact Ceiling Point Algorithm appears to be highly competitive with each of the other algorithms on all of the realistic problems except IBM-5. Run on the fastest computer of any shown here, the Surrogate Cutting Plane Algorithm's execution times are the smallest and most stable. It is interesting that the Surrogate Cutting Plane Algorithm did not find the poorly-scaled {FC-5,..., FC-8} problems any more difficult to solve than {FC-1,..., FC-4}. All of the other algorithms found the set of problems {FC-5,..., FC-8, FC-10, IBM-4, IBM-5} relatively difficult to solve.

For the randomly generated test problems, Table 6-V(b) compares the Exact Ceiling Point Algorithm, GAMS/ZOOM, Hillier's Bound-and-Scan Algorithm [16] run on an IBM-360/67 and the Surrogate Cutting Plane Algorithm. The total time figures for the Bound-and-Scan Algorithm were compiled by adding the times for the heuristic algorithm described in [15] to those times given in [16] since the best solution yielded by the former was used as an initial solution for the latter algorithm [16, p. 668]. The execution times reported for the algorithms of [16] and [2] are very stable. However, since the IBM-360/67 is probably several times slower than the IBM-370/168, Hillier's algorithm may be faster than the algorithm of Austin and Ghandforoush. Overall, both of these algorithms are more consistent than the Exact Ceiling Point Algorithm (and GAMS/ZOOM) on problems of Type I and II. While the Exact Ceiling Point Algorithm is competitive with these two algorithms on about half of the Type II problems, GAMS/ZOOM is not really competitive at all, despite the fact that it was only required to find a solution within 2 percent of op-

timality for Type II problems. On Type III problems, the Exact Ceiling Point Algorithm, the Bound-and-Scan Algorithm and the Surrogate Cutting Plane Algorithm all appear to be of roughly comparable efficiency. Even on the 0-1 problems (classes AL and Type III) for which it was designed, GAMS/ZOOM did not perform as well as the Exact Ceiling Point Algorithm, which was not designed for 0-1 problems.

Tables 6-V(a) and (b) are summarized by problem class in Table 6-V(c).

Table 6-V(c). Summary of Performances by Exact Algorithms.

	XCPA		GAMS/ZOOM		[10]		[2]	[14]
	VaxStation II		VaxStation II		IBM 360/67		370/168	7090
Class	Time	Ratio	Time	Ratio	Time	Ratio	Time	Time
FC	0.48	2.9	7.02	9.7	12.91	285.2	0.32	4.98
IBM	10.17	17.3	>189	>145	2.64	9.9		17.18
AL	0.65	2.2	1.17	3.1			0.17	4.29

	XCPA		GAMS/ZOOM		[16]		[2]
	VaxStation II		VaxStation II		IBM 360/67		370/168
Class	Time	Ratio	Time	Ratio	Time	Ratio	Time
I	>546	>1142	>659	>575	12.27	12.6	2.14
II*	19.16	42.5	76.26	105.3	2.61	3.9	1.15
III	1.34	3.2	4.67	7.9	1.50	2.5	0.46

* Averages over {II-1,...,II-10}.

10. Summary

In this report, we have presented an exact algorithm for solving (ILP): given enough computer time, it could in theory solve to optimality any (ILP) for which $\bar{x}$ exists and is unique. The Exact Ceiling Point Algorithm is based upon theorems implying that to solve (ILP) it is sufficient to enumerate only the feasible 1-ceiling points lying within the n-simplex S. This is done by systematically searching for feasible 1-ceiling points with

respect to one constraint at a time. While enumerating such solutions, three features described in Section 4 help to accelerate the fathoming process, including the construction of conditional variable bounds and an ability to double backtrack. Because the iterative part of the Exact Ceiling Point Algorithm benefits greatly by having as good an initial solution as possible, a heuristic device is employed prior to the execution of the iterative procedure. This device is to search for 1-ceiling points with respect to an intersection cut located close to $\bar{x}$, thereby examining the most promising part of the feasible region first.

Our computational experience leads us to believe that the Exact Ceiling Point Algorithm is a good approach for tackling some problems, but not the best approach for tackling others. Specifically, the Exact Ceiling Point Algorithm performed very well on the set of small realistic problems, but did not perform as well as several other algorithms on some of the more difficult randomly generated problems. On the set of realistic problems, particularly the fixed-charge problems, the double backtracking feature (see Section 4.2) of the exact enumeration scheme seems to be very helpful in quickly fathoming large numbers of solutions. The variables also appear to be arranged in an order that is advantageous to this scheme. However, on the randomly generated problems, the ability to double backtrack appears to be of little help because there is no pattern of ratios $a_{i,j-1}/a_{ij}$ from one column to the next. Also, when the Heuristic Ceiling Point Algorithm did not find a very good solution, as was the case for most of the Type I problems, it often led to an inefficient performance by the Exact Ceiling Point Algorithm. Further research is needed to better determine what types of problems are best and least suited to our ceiling point approach.

References

[1] Austin, L., and Ghandforoush, P., "An Advanced Dual Algorithm with Constraint Relaxation for All-Integer Programming," *Naval Research Logistics Quarterly*, 30, 133-143 (1983).

[2] Austin, L., and Ghandforoush, P., "A Surrogate Cutting Plane Algorithm for All-Integer Programming," *Computers & Operations Research*, **12**, 241-250 (1985).

[3] Avriel, M., ***Nonlinear Programming: Analysis and Methods***, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[4] Balas, E., "Intersection Cuts – A New Type of Cutting Planes for Integer Programming," *Operations Research*, **19**, 19-39 (1971).

[5] Balas, E., Bowman, V., Glover F., and Sommer, D., "An Intersection Cut From the Dual of the Unit Hypercube," *Operations Research*, **19**, 40-44 (1971).

[6] Brooke, A., D. Kendrick and A. Meeraus, ***GAMS: A User's Guide***, The Scientific Press, Redwood City, Calif., 1988.

[7] Balas, E. and C. Martin, "Pivot and Complement: A Heuristic for 0-1 Programming," *Management Science*, **26**, 86-96 (1980).

[8] Crowder, H., R. Dembo and J. Mulvey, "Reporting Computational Experiments in Mathematical Programming," *Mathematical Programming*, **15**, 316-329 (1978).

[9] Dongarra, J., "Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment," *ACM SIGNUM Newsletter*, **19**, 23-26 (1984).

[10] Faaland, B., and Hillier, F., "The Accelerated Bound-and-Scan Algorithm for Integer Programming," *Operations Research*, **23**, 406-425 (1975).

[11] Garfinkel, R. and G. Nemhauser, ***Integer Programming***, John Wiley, New York, 1972.

[12] Gorry, G., and Shapiro, J., "An Adaptive Group Theoretic Algorithm for Integer Programming Problems," *Management Science*, **17** , 285-306 (1971).

[13] Haldi, J., "25 Integer Programming Test Problems," Working Paper No. 43, Graduate School of Business, Stanford Univesity, Stanford, Calif., December 1964.

[14] Haldi, J., and Issacson, L., "A Computer Code for Integer Solutions to Linear Programs," *Operations Research*, **13**, 946-959 (1965).

[15] Hillier, F., "Efficient Heuristic Procedures for Integer Linear Programming with an Interior," *Operations Research*, **17**, 600-637 (1969).

[16] Hillier, F., "A Bound-and-Scan Algorithm for Pure Integer Linear Programming with General Variables," *Operations Research*, **17**, 638-679 (1969).

[17] Hillier, F., "A Bound-and-Scan Algorithm for Pure Integer Linear Programming with General Variables," Technical Report No. 11, Dept. of Operations Research, Stanford University, Stanford, Calif., May 1969.

[18] Krolak, P., "Computational Results of an Integer Programming Algorithm," *Operations Research*, **17**, 743-749 (1969).

[19] Land, A. and A. Doig, "An Automatic Method of Solving Discrete Programming Problems," *Econometrica*, **28**, 497-520 (1960).

[20] Lustig, I., "Comparisons of Composite Simplex Algorithms," Technical Report SOL 87-8, Dept. of Operations Research, Stanford University, Stanford, Calif., June 1987.

[21] Markowitz, H., and A. Manne, "On the Solution of Discrete Programming Problems," *Econometrica*, **25**, 84-110 (1957).

[22] Saltzman, R., "Ceiling Point Algorithms for General Integer Linear Programming," unpublished Ph.D. dissertation, Dept. of Operations Research, Stanford University, Stanford, Calif., December 1988.

[23] Saltzman, R., and F. Hillier, "The Role of Ceiling Points in General Integer Linear Programming," Technical Report SOL 88-11, Dept. of Operations Research, Stanford University, Stanford, Calif., August 1988.

[24] Saltzman, R., and F. Hillier, "A Heuristic Ceiling Point Algorithm for General Integer Linear Programming," Technical Report SOL 88-19, Dept. of Operations Research, Stanford University, Stanford, Calif., November 1988.

[25] Saunders, M., private communication, November 3, 1988.

[26] Taha, H., *Integer Programming: Theory, Applications and Computations*, Academic Press, New York, 1975.

[27] Trauth, C., and R. Woolsey, "Integer Linear Programming: A Study in Computational Efficiency," *Management Science*, **15**, 481-493 (1969).

In order for GAMS/ZOOM to convert each general integer variable into a sum of binary variables, a reasonably tight upper bound was specified for each general integer variable, as shown in Table VI. The number n' of binary variables in the transformed problem is given in the last column.

Table VI. Upper Bounds Specified in the GAMS/ZOOM Runs.

Problem/Class	Upper Bounds	n'
FC-1,...,FC-4	$x_j \le 1,\ j = 1,2;\quad x_j \le 10,\ j = 3,...,5$	14
FC-5,...,FC-8	$x_j \le 1,\ j = 1,2;\quad x_j \le 100,\ j = 3,...,5$	23
FC-9	$x_j \le 1,\ j = 1,...,3;\quad x_j \le 10,\ j = 4,...,6$	15
FC-10	$x_j \le 1,\ j = 1,...,6;\quad x_j \le 15,\ j = 7,...,12$	30
IBM-1, IBM-2	$x_j \le 7,\quad j = 1,...,7$	21
IBM-3	$x_j \le 31,\quad j = 1,...,4$	20
IBM-4, IBM-5	$x_j \le 3,\ j = 1,...,15$	30
AL	$x_j \le 1,\ j = 1,...,10$	10
Types I & II*	$x_j \le 31,\ j = 1,...,15$	75
I-5	$x_j \le 50,\ j = 1,...,15$	90
Type III	$x_j \le 1,\ j = 1,...,15$	15

* except I-5

GAMS/ZOOM Options specified in every Program file "PROBLEM.GMS"

OPTCA = 0.0
OPTCR = 0.001 (0.020 for all Type II problems)

GAMS/ZOOM Options listed in Specs file "GAMSZOOM.SPC"

BRANCH = YES
DIVE = YES
EXPAND = 3
HEURISTIC = YES
INCUMBENT = -1000 (+1000 for IBM problems)
MAX SAVE = 5
PRINT CONTINUOUS = 0
PRINT HEURISTIC = 0
PRINT BRANCH = 0
PRINT TOUR = 0
QUIT = NO
SELECT = 2

All other options assumed their default values. A preliminary run was made for each test problem with PRINT HEURISTIC = 1 in order to find $z_H \equiv c^T x_H$.

Listing of Fortran Code Implementation of
the Exact Ceiling Point Algorithm
for General Integer Linear Programming

```
C PLIST.FOR:  mm(nn) = maximum value of M(N)  Put into all routines
      IMPLICIT REAL*8 (A-H,O-Z)
      parameter (mm=37, nn=31, tol=2.10734D-08, bigm=1.D5,
     -  maxit=75, one=1.D0, zero=0.D0)

C COMLP1.FOR:  Global vars. created/used in LP routines
      dimension A(mm,nn),B(mm),C(nn),INDCT(mm),
     -    ABAR(mm,nn+mm+mm+1),ABAL(nn,nn),
     -    ibasis(mm),nonbas(nn+mm),indbas(nn+mm+mm),initbv(mm),
     -    NPIV(2),XBAR(nn+mm+mm),X0(nn),signac(mm,nn),
     -    dirs(nn,nn),CTXPT(nn,mm),BFCX0(mm),PRICES(mm)
      common/clp1/A,B,C,INDCT,INDOBJ,M,N,
     -    ABAR,ABAL,ibasis,nonbas,indbas,NPIV,XBAR,X0,Z0,IR,IS,
     -    signac,nx,dirs,ctxpt,BFCX0,PRICES
      logical*2 indbas,signac,ctxpt,BFCX0

C COMHRUN.FOR:  Global vars. used in HRUN routines
      dimension FIS3(mm,1+nn),RATES(nn),SDIR(nn),CTVAL(mm)
      common/chrun/ffeas,FIS3,pct01,RATES,SDIR,ncp,CTVAL
      logical*2 ffeas,ncp

C COMRUN1.FOR:  Global vars. primarily used in RUN
      real*8  RTIMES(-1:30),RPCTS(-1:30),RVALS(-1:30),
     -        AIMIN(mm),AIMAX(mm)
      integer IXSTAR(nn),CORDER(nn),VARORD(nn)
      common/crun1/IXSTAR,ZSTAR,AIMIN,AIMAX,ZUP,ALL,
     -        RTIMES,RPCTS,RVALS,CORDER,VARORD

C COMXRUN2.FOR:  Global vars. used in XRUN and RUNCUT
      Dimension LAMDA(nn), ALPHA(nn,nn), P(nn,nn), SUB(nn)
      Dimension LOBD(nn),UPBD(nn),LONS(nn),UPNS(nn),CUT(nn+1)
      common/cxrun2/LAMDA,ALPHA,P,SUB,LOBD,UPBD,LONS,UPNS,CUT
      integer LOBD,UPBD,LONS,UPNS,SUB
      real*8  LAMDA

C COMXRUN3.FOR:  Global vars. used in XRUN
      integer irange(nn),icase
      real*8  CMPMIN(mm,nn+1),RA(mm,nn),AXINC(mm,nn),
     -    LL(mm+1,nn),UU(mm+1,nn),sizlim,callim,xcalls(nn)
      common/cxrun3/irange,icase,LL,UU,CMPMIN,RA,AXINC,
     -    sizlim,callim,xcalls

C COMXRUN4.FOR: More XRUN global vars., especially XCP-related routines
      integer LJ,first(nn),final(nn),inc(nn),newz(nn),
     -        loc(nn),upc(nn),ISN(nn)
      real*8  gap(mm,nn+1)
      common/cxrun4/gap,LJ,first,final,inc,newz,loc,upc,
     -        ISN,minarg,maxarg

C COMPRT.FOR:  Print Switches
      common/cprint/hprint,xprint
      integer*2 hprint(25),xprint(25)

C COMIO.FOR:  I/O files
      common/cinout/infile,iorun,iohrun,iocut,ioxrun,iolp
      character*64  infile,iorun,iohrun,iocut,ioxrun,iolp
```

```
c-----------------------------------------------------------------
c By Robert M. Saltzman
c
c Applies the Exact Ceiling Point Algorithm to each problem
c listed in the file ILPDATA.DAT.
c
c Parenthetical comments with Section numbers refer to parts of:
c Saltzman, R., "Ceiling Point Algorithms for General Integer
c Linear Programming," unpublished Ph.D. dissertation, Dept. of
c Operations Research, Stanford University, Stanford, Calif.,
c December 1988.
c-----------------------------------------------------------------
c
      include '$DISK2:[SALTZ.ILP1]plist.for'
      include '$DISK2:[SALTZ.ILP1]comio.for'
      include '$DISK2:[SALTZ.ILP1]comprt.for'
      include '$DISK2:[SALTZ.ILP1]comxrun3.for'
      open( 2,file='$DISK2:[SALTZ.ILP1]ilpdata.dat', status='old')
      open( 5,file='$DISK2:[SALTZ.ILP1]outhrun.dat', status='unknown')
      open( 6,file='$DISK2:[SALTZ.ILP1]outrun.dat',  status='unknown')
      open(22,file='$DISK2:[SALTZ.ILP1]switches.dat',status='old')
c
c     Read in print switches and run-time options
      read(22,*) (hprint(j),j=1,25)
      read(22,*) (xprint(j),j=1,25)
      read(22,*)  sizlim,callim
      write(*,*) 'sizlim,callim ',sizlim,callim
      write(6,*)'Exact Ceiling Point Algorithm Summary'
      write(6,*)' '
      write(6,*)'Problem      LP     Z0 Set+Heur   Z  ',
     - ' IntCut   Z    Exact   Z    Total   Ratio'
      write(6,*)'-------      --     -- --------   -  ',
     - ' ------   -    -----   -    -----   -----'
c
c     Headlines for Heuristic Summary report, if desired
      if (hprint(19) .eq. 1) then
         write(5,*)'Heuristic Ceiling Point Algorithm Summary'
         write(5,*)' '
         write(5,*)'Problem      LP     Z0 Set+Ph.1   Z  ',
     -     ' Phase2   Z    Phase3  Z    Total   Ratio'
         write(5,*)'-------      --     -- --------   -  ',
     -     ' ------   -    -----   -    -----   -----'
      endif
c
c     Loop through all problems specified in ILPDATA.DAT
      do 8100 ip = 1, 40
c        Get the name of the input data file, e.g., HALDI5.DAT
         read(2,8105) infile
         if (infile(1:3) .eq. 'end') goto 8150
         if (infile(1:3) .eq. 'END') goto 8150
         write (*,*) '*****  Starting problem ***** ',infile
c
         call RUN
c
         write (*,*) '*****  Finished problem ***** ',infile
8100  continue
8105  format(A11)
8150  continue
      stop
      end
```

```
c-----------------------------------------------------------------
c
      subroutine RUN
c
c Runs Heuristic and Exact Ceiling Point Algorithms for 1 problem.
c (Overview of entire algorithm given in Section 5.5.)
c Note: Code for Heuristic Ceiling Point Algorithm subroutines
c {LPSOLVE, SETUP, HRUN} is listed in Saltzman and Hillier [24].
c-----------------------------------------------------------------
c
      include '$DISK2:[SALTZ.ILP1]plist.for'
      include '$DISK2:[SALTZ.ILP1]comio.for'
      include '$DISK2:[SALTZ.ILP1]comlp1.for'
      include '$DISK2:[SALTZ.ILP1]comprt.for'
      include '$DISK2:[SALTZ.ILP1]comrun1.for'
      include '$DISK2:[SALTZ.ILP1]comxrun4.for'
      integer izz(5)
c
      open(4,file='$DISK2:[SALTZ.ILP1]outlp.dat',     status='unknown')
      open(8,file='$DISK2:[SALTZ.ILP1]outruncut.dat',status='unknown')
      open(9,file='$DISK2:[SALTZ.ILP1]outxrun.dat',   status='unknown')
c
c     Initialize clock reading routines and summary values
      call XTIMER(0,0,0)
      do 8001 i = -1, 30
         rtimes(i) = zero
         rpcts(i)  = zero
         rvals(i)  = zero
8001  continue
c
c     Solve LP-relaxation of (ILP)
      call XTIMER(1,0,zero)
      call LPSOLVE
      call XTIMER(-1,0,RTIMES(-1))
      if (xprint(8) .eq.0)
     -    write(*,*)'***> LPSOLV:',RTIMES(-1),' Z0=',Z0
c
c     Check for all-integer LP solution
      do 8004 j = 1, n
         if (DABS(X0(j)-IRNDWN(X0(j))) .gt. tol) goto 8008
8004  continue
      write(6,*)'---> LP solution is all-integer (X*=X0)'
      do 8006 j = 1, n
         ixstar(j) = IRNDWN(X0(j))
8006  continue
      goto 8090
c
c     Initialize {ALL, X*, ZUP}
8008  call XTIMER(1,0,zero)
      ALL = one
c     ALL = 1 => only search for solns. strictly better than incumbent
      ZUP = DBLE(IRNDWN(Z0))
      do 8010 j = 1, n
         IXSTAR(j) = -1
8010  continue
c
```

```
c       Calculate global vars. (SIGNAC, ABAL, DIRS, CTXPT, BFCX0)
        call SETUP
        call XTIMER(-1,0,rtimes(0))
        if (xprint(8 ).eq.0) write(*,*) '***> SETUP: ',rtimes(0)
c
c       Set default values for Z* in case HRUN fails or is bypassed
        if (indobj .eq.  1) ZSTAR = zero
        if (indobj .eq. -1) ZSTAR = -2.*DABS(Z0)
        if (xprint(21) .eq. 1) goto 8025
        if (xprint(22) .eq. 1) goto 8035
c
c.....................................................................
c       Run Heuristic Ceiling Point Algorithm
        call XTIMER(1,0,zero)
c
        call HRUN
c
        if (ixstar(1) .lt. 0)
     -  write(*,*)'***> HRUN failed.  Initial Z* =',ZSTAR
        call XTIMER(-1,0,rtimes(1))
c
        izz(1) = IRNDWN(ZSTAR)
        if (xprint(8).eq.0)
     -      write(*,*)'***> HRUN:  ',rtimes(1),' Z*=',izz(1)
c
c       Write out summary information for the Heuristic Algorithm
        if (hprint(19) .eq. 1) then
          rtimes(1) = rtimes(11)+rtimes(12)+rtimes(13)-rtimes(0)
          write(5,8015) infile,rtimes(-1),Z0,rtimes(11),rtimes(12),
     -    rvals(12),rtimes(13),rvals(13),rtimes(14),rpcts(15)
8015      format(1X,A11,F5.2,F7.1,F7.2,6X,F8.2,F6.0,F7.2,F6.0,2F7.2)
          write(5,8020) infile,rpcts(10),rpcts(11),rpcts(12),rpcts(13)
8020      format(1X,A11,F5.2,6X,F8.2,6X,F8.2,5X,F8.2)
        endif
c
        if (DBLE(izz(1)) .ge. ZUP) then
           write(6,*)'---> Heuristic alg. found optimal solution'
           izz(2) = izz(1)
           izz(3) = izz(1)
           goto 8090
        endif
c
c.....................................................................
c       Search the Intersection Cut prior to full enumeration
8025    IF (XPRINT(18) .EQ. 1) GOTO 8090
        call XTIMER(1,0,zero)
        call RUNCUT
        call XTIMER(-1,0,rtimes(2))
C
        izz(2) = IRNDWN(ZSTAR)
        if (xprint(8).eq.0)
     -      write(*,*)'***> RUNCUT:',rtimes(2),' Z*=',izz(2)
        if (DBLE(izz(2)) .ge. ZUP) then
           write(6,*) '---> I-Cut search found optimal solution'
           izz(3) = izz(2)
           goto 8090
        endif
```

```
c
        IF (XPRINT(19) .EQ. 1) GOTO 8090
c
c.....................................................................
c       Eliminate redundant constraints
        call XTIMER(1,0,zero)
        call REDCTS
c
c       Run Exact Ceiling Point Algorithm
8035    call XRUN
        call XTIMER(-1,0,rtimes(3))
        izz(3) = IRNDWN(ZSTAR)
        if (xprint(8).eq.0)
     -      write(*,*)'***> XRUN:  ',rtimes(3),' Z*=',izz(3)
c
c       Write out an optimal solution & execution times
8090    call REPRT1(izz)
c
        return
        end
c---------------------------------------------------------------------
c
        subroutine REPRT1(iz)
c
c Called by RUN to report a few important pieces of information
c---------------------------------------------------------------------
c
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comio.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
c
        integer ixx(nn),ixxx(nn),iz(5)
c       ixx & ixxx are reordered versions of IXSTAR
c       iz  is a vector of objective function values
c
c       Put X* in original variable order before writing out
        do 8091 j = 1, n
           ixx(varord(j)) = IXSTAR(j)
8091    continue
        do 8092 j = 1, n
           ixxx(corder(j)) = ixx(j)
8092    continue
c
        write (9,*) ' '
        write (9,*) infile
        write (9,*) '   Z* =',iz(3)
        do 8093 j = 1, n
           write(9,*) '   X*(',j,') =',ixxx(j)
8093    continue
c
```

```
c.....................................................................
c       Write out summary information for the Exact Algorithm
        do 8095 i = -1, 3
           rtimes(4) = rtimes(4) + rtimes(i)
8095    continue
        if (xprint(8).eq.0)
     -      write(*,*)'***> TOTAL TIME: ',rtimes(4)
c
c       Combine times of SETUP & HRUN
        rtimes(1) = rtimes(0) + rtimes(1)
c
c       Get each part's percentage of TOTAL time & TOTAL/LP ratio
        do 8096 i = -1, 3
           rpcts(i) = 100.D0*rtimes(i)/rtimes(4)
8096    continue
        tot2lp = rtimes(4)/rtimes(-1)
c
        write(6,8097) infile,rtimes(-1),Z0,rtimes(1),
     -  iz(1),rtimes(2),iz(2),rtimes(3),iz(3),rtimes(4),tot2lp
8097    format(1X,A11,F5.2,F7.1,F8.2,I5,F8.2,I5,F8.2,I5,2F8.2)
C
        write(6,8098) infile,rpcts(-1),rpcts(1),rpcts(2),rpcts(3)
8098    format(1X,A11,F5.2,7X,F8.2,5X,F8.2,5X,F8.2)
c
        return
        end
```

```
c-------------------------------------------------------------
c
        subroutine RUNCUT
c
c Called by RUN to search intersection cut ("I-Cut")
c-------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
c
c       Perform preparatory routines
        call PRECUT
c
        if (zstar .ge. zup) goto 4020
        ihp = m-1
        if (xprint(1).eq.1) write(8,*) 'I-CUT',(A(ihp,j),j=1,n)
c
c       Find RA once (since there's no variable reordering here)
        call ARATIO
c
c       Enumerate feasible 1-CP's with respect to I-Cut
        call XCP(ihp)
c
c       AIMAX(i)= largest coefficient in absolute value of
c       constraint (i) over the support of C.
        do 4015 i = 1, m
           rmax = -bigm
           do 4010 j = 1, n
              if (C(j) .eq. zero) goto 4010
              if (DABS(a(i,j)) .gt. rmax) rmax = DABS(A(i,j))
4010       continue
           AIMAX(i) = rmax - one
4015    continue
c
c       Adjust AIMAX for I-cut since its coefficients are non-integer
        AIMAX(m-1) = AIMAX(m-1) + one
c
4020    continue
        return
        end
c-------------------------------------------------------------
c
        subroutine PRECUT
c
c Called by RUNCUT to prepare for intersection cut search
c-------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
c
c       all = 1 => search only for strictly better solutions
        all = one
        zlevel = zstar+all
        if (xprint(2).eq.1)write(8,*)
     -    'PRECUT: zlevel=',zlevel,' zup=',zup
c
```

```
c       Find {P(k)'s} = Set of intersection points
        call OBJCUT(zlevel)
c
        if (n .ge. 15) goto 4025
        if (xprint(20) .eq. 1) goto 4025
c
c       Either get spherical intersection cut points {alpha}
        call CUTPT
        goto 4030
c
c       or, get dual-UHC intersection cut points {alpha*}
4025    call CUTPT2
c
4030    continue
c
c       Create SUB = Simple Upper Bounds
        call BOUNDS
        if (ISRSIM(m) .eq. 1) goto 4035
        if (xprint(2).eq.1) write(8,*) 'Shr(ns)=0 => Z* = ',zstar
        zup = -bigm
        goto 4090
c
4035    continue
c
c       Find ALPHA(k) with largest objective function value
c       It becomes a new (smaller) upper bound on Z*
        vmax = -bigm
        do 4040 k = 1, n
           value = VDOT(n,C,ALPHA(1,k))
           if (value .gt. vmax) vmax = value
4040    continue
        ZUP = DBLE(IRNDWN(vmax))
        if (zstar .lt. vmax) goto 4045
        if (xprint(2).eq.1) write (8,*) 'Z* >= zup => Z*=',zstar
c       goto 4090
c
4045    continue
c       Determine the I-cut coefficients
        call CUTHP
c
c       Append I-Cut & Obj.fn. to bottom of A matrix and B vectors
        do 4050 j = 1, n
           A(m+1,j) = CUT(j)
           A(m+2,j) =  -C(j)
4050    continue
        B(m+1) = CUT(n+1)
        B(m+2) = zero
c.....................
        M = M + 2
c.....................
4090    continue
        return
        end
```

```
c-------------------------------------------------------------
c
        subroutine OBJCUT(objval)
c
c Called by PRECUT & INCMOD to find the set of P(k)'s, where
c P(k) = point where the kth extreme ray intersects objective
c         function hyperplane: cx = objval.  (See Section 5.2)
c-------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
c
        real*8 objval,theta
        if (xprint(3).eq.1)
     -  write (8,*) 'OBJCUT:objval = ',objval,' Z0 =',Z0
        zgap = Z0 - objval
c
c       For each extreme direction k, find intersection point P(k)
        do 4120 k = 1, nx
           theta = bigm
           dot   = -VDOT(n,C,DIRS(1,k))
c          dot   = 0 => ext. dir. is // objective function hyperplane
           if (DABS(dot) .lt. tol) goto 4100
           theta = zgap/dot
c
4100       do 4110 j = 1, n
              P(j,k) =  X0(j) + theta*DIRS(j,k)
4110       continue
           if (xprint(3).eq.1)
     -     write (8,*) 'OBJCUT: Pk=',(P(j,k),j=1,n)
4120    continue
        return
        end
c-------------------------------------------------------------
c
        subroutine CUTPT
c
c Called once by PRECUT to find the set of ALPHA(k)'s, where
c ALPHA(k) = point where kth extreme ray intersects I-cut hyperplane
c See paper by E. Balas [Ba71] on Spherical Intersection Cuts.
c-------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
c
        dimension f(nn),fe2(nn),emf(nn)
        do 4200 j = 1, n
           f(j)   = X0(j) - DINT(X0(J))
           fe2(j) = f(j)  - 0.5D0
           emf(j) = one   - f(j)
4200    continue
        fef = VDOT(n,f,emf)
c
```

```
      do 4220 k = 1, nx
         ak2 = VDOT(n,ABAL(1,k),ABAL(1,k))
         hk  = VDOT(n,fe2,ABAL(1,k))
         lamda(k) = DSQRT((hk*hk)+(fef*ak2))
         lamda(k) = (hk + lamda(k))/ak2
         do 4210  j = 1, n
            alpha(j,k) = X0(j) - lamda(k)*abal(j,k)
4210     continue
         if (xprint(4).eq.1)write(8,*)'Alpha',(alpha(j,k),j=1,n)
4220  continue
      if (xprint(4).eq.1) write(8,*) 'Lamda',(lamda(j),j=1,n)
      return
      end
c-------------------------------------------------------------
c
      subroutine CUTHP
c
c Called once by PRECUT to compute I-cut coefficients ("CUT")
c-------------------------------------------------------------
      include '$DISK2:[SALTZ.ILP1]plist.for'
      include '$DISK2:[SALTZ.ILP1]comprt.for'
      include '$DISK2:[SALTZ.ILP1]comlp1.for'
      include '$DISK2:[SALTZ.ILP1]comxrun2.for'
c
      do 4300 k = 1, n
         cut(k) = zero
         if (.not. indbas(k)) cut(k) = -one/lamda(k)
4300  continue
      cut(n+1) = -one
c
      do 4320 j = 1, n
         nb = nonbas(j)
         if (nb .le. n) goto 4320
         i  = nb - n
         if (i  .gt. m) goto 4320
         do 4310 k = 1, n
            cut(k) = cut(k) + (A(i,k)/lamda(j))
4310     continue
         cut(n+1) = cut(n+1) + (B(i)/lamda(j))
4320  continue
c
c     Make sure that CUT chops off X0 (if not, mult. by -1)
      vlhs = VDOT(n,CUT,X0)
      if (vlhs .gt. CUT(n+1)) goto 4340
      do 4330 j = 1, n+1
         cut(j) = -cut(j)
4330  continue
4340  if (xprint(5).eq.1) write (8,*) 'ICUT',(CUT(j),j=1,n+1)
      return
      end
```

```
c-------------------------------------------------------------
c
        subroutine CUTSHR
c
c Called by ISHR to calculate SHR(I-Cut), the set of unconditional
c bounds for all variables.  They are based on the alpha(k,j)'s,
c rounded according to the sign of CUT(j).
c-------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
c
c       Work through each component j
        do 4430 j = 1, n
           lobd(j) = IRNDWN(bigm)
           upbd(j) = 0
c
c          Consider all the ALPHA(k)'s
           do 4425 k = 1, n
c
              if (CUT(j)) 4405,4410,4415
c             if cut(k) < 0, round alpha(j,k) up
4405          ibd = IRNDUP(alpha(j,k))
              go to 4420
c             if cut(k) is 0, round alpha(j,k) to nearest int
4410          ibd = IDNINT(alpha(j,k))
              go to 4420
c             if cut(k) > 0, round alpha(j,k) down
4415          ibd = IRNDWN(alpha(j,k))
c
4420          if (ibd .lt. lobd(j)) lobd(j) = ibd
              if (ibd .gt. upbd(j)) upbd(j) = ibd
4425       continue
c
c          Take intersection of n-simplex and uncdl. bds.
           if (lons(j) .gt. lobd(j)) lobd(j) = lons(j)
           if (upns(j) .lt. upbd(j)) lobd(j) = upns(j)
4430    continue
c
4450    continue
        return
        end
c-------------------------------------------------------------
c
        subroutine ARATIO
c
c Called by PRECUT and REORDR to calculate the ratio of adjacent
c elements of A(i,), row by row (= f(i,j) in Section 5.4).  Also
c sets up global variables INC and AXINC.
c-------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
        include '$DISK2:[SALTZ.ILP1]comxrun3.for'
        include '$DISK2:[SALTZ.ILP1]comxrun4.for'
```

```
c
c       Determine INC(j)= direction in which to enumerate X(j) values
        do 4510 j = 1, n
c          Assume c(j) >= 0 => X(j) starts at upper bound (INC(j) = -1)
           inc(j) = -1
           if (c(j) .lt. zero) inc(j) = 1
           do 4505 i = 1, m
              AXINC(i,j) = A(i,j)*DBLE(inc(j))
4505       continue
4510    continue
c
c       Now calculate RA, the matrix of ratios
        do 4520 i = 1, m
           do 4515 j = 2, n
              RA(i,j) = zero
              if (A(i,j) .ne. zero) RA(i,j) = A(i,j-1)/A(i,j)
4515       continue
           if (xprint(7) .eq. 1) then
              write (8,*) 'ARATIO: RA=',(RA(i,j),j=1,n)
              write (8,*) 'ARATIO: AXINC=',(AXINC(i,j),j=1,n)
           endif
4520    continue
        if (xprint(7) .eq. 1)
     -  write (8,*) 'ARATIO: Inc=',(inc(j),j=1,n)
        return
        end
c-------------------------------------------------------------
c
        subroutine CUTPT2
c
c Called by PRECUT; calc. {alpha*} = int. pts. of rays & I-cut
c See paper by Balas, et al. [BBGS71] on  using DUAL-UHC[X0].
c-------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
c
        Integer indr(nn)
        Real*8  f(nn),fe2(nn),sigma(nn),L(nn),LIr0,vec(nn),theta(nn)
        Logical nplus(nn)
        dn2 = DBLE(n)/2.D0
        do 4600 j = 1, n
           f(j)   = X0(j) - DINT(X0(j))
           fe2(j) = f(j)  - 0.5D0
           indr(j)= 0
4600    continue
c
c       Loop through all of the extreme directions
        do 4690 k = 1, nx
           NL = 0
           do 4610 j = 1, n
              L(j) = bigm
              nplus(j) = .false.
```

```
            if (zero .lt. (fe2(j)*ABAL(j,k))) then
               nplus(j) = .true.
               NL = NL + 1
               L(j) = fe2(j)/ABAL(j,k)
            endif
4610        continue
c
            LIr0 = zero
            if (NL .eq. 0) goto 4640
            call RSORT1(n,L,indr)
c
            do 4630 i = 1, NL
               do 4625 j = 1, n
                  vec(j) = fe2(j) - ABAL(j,k)*L(indr(i))
4625           continue
               vecnrm = V1NORM(n,vec)
               if (vecnrm .gt. dn2) goto 4640
               LIr0 = L(indr(i))
4630        continue
c
4640        do 4650 j = 1, n
               theta(j) = f(j) - ABAL(j,k)*LIr0
4650        continue
c
            nn1 = 0
            do 4660 j = 1, n
               sigma(j) = -1.D0
               if ((theta(j).gt.0.5D0).or.
     -             ((theta(j).eq.0.5D0).and.(ABAL(j,k).lt.zero)))then
                  sigma(j) = one
                  nn1 = nn1 + 1
               endif
4660        continue
c
            temp1 = VDOT(n,f,sigma)
            temp2 = VDOT(n,ABAL(1,k),sigma)
            LAMDA(k)= (temp1-dble(nn1))/temp2
            do 4670 j = 1, n
               ALPHA(j,k) = X0(j) - LAMDA(k)*ABAL(j,k)
4670        continue
            if (xprint(4) .eq. 1) then
               write(4,*) '------------> Ray k = ',k
               write(4,*) 'ABAL  ',(ABAL(j,k),j=1,n)
               write(4,*) 'nplus ',(nplus(j),j=1,n)
               write(4,*) 'L     ',(L(j),j=1,n)
               write(4,*) 'indr  ',(indr(j),j=1,n)
               write(4,*) 'LIr0  ',LIR0
               write(4,*) 'sigma ',(sigma(j),j=1,n)
               write(4,*) 'theta ',(theta(j),j=1,n)
               write(4,*) 'nn1   ',nn1
               write(4,*) 'LAMDA ',LAMDA(k)
               write(4,*) 'ALPHA ',(ALPHA(j,k),j=1,n)
            endif
4690     continue
         return
         end
```

```
c-------------------------------------------------------------
c
        subroutine REDCTS
c
c  Called by RUN prior to XRUN.
c  Eliminates redundant cts. not intersecting n-simplex S.
c  THIS ROUTINE MODIFIES PROBLEM DATA.
c-------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]PLIST.FOR'
        include '$DISK2:[SALTZ.ILP1]COMPRT.FOR'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
        include '$DISK2:[SALTZ.ILP1]comxrun3.for'
        include '$DISK2:[SALTZ.ILP1]comxrun4.for'
        logical*2 active(mm)

        if (hprint(17).eq.1) write (8,*) 'REDCTS: eliminating cts'
c
c       nactiv = counts the number of active (non-redundant) cts.
        nactiv = 0
c
c       Check each constraint for redundancy
        do 6970 i = 1, m-2
           active(i) = .false.
           if (BFCX0(i)) then
c             Constraint binding at X0: always active
              active(i) = .true.
              nactiv    = nactiv + 1
              if (hprint(17).eq.1) write (8,*) ' ct.',i,' active'
              goto 6970
           endif
c
c          Nonbinding ct: Does it cause a P(k) to be infeasible?
           do 6960 k = 1, nx
              aipk = zero
              do 6957 j = 1, n
                 aipk = aipk + A(i,j)*P(j,k)
6957          continue
c
c             Test feasibility of P(k) wrt this constraint
              if (aipk .gt. B(i)) then
c                P(k) violates (i), so (i) is active
                 active(i) = .true.
                 nactiv    = nactiv + 1
                 if (hprint(17).eq.1) write (8,*) ' ct.',i,' active'
                 goto 6970
              endif
c
6960       continue
c
6970    continue
c
c       Make sure last 2 contraints (I-Cut & OBJ) are active
        active(m-1) = .true.
        active(m)   = .true.
        nactiv      = nactiv + 2
c       If all original cts. binding at X0, then none redundant
        if (nactiv-2  .eq. m) goto 6990
```

```
c
c       Delete/overwrite redundant cts. from XRUN global vars.
        ia = 0
        do 6980 i = 1, m
           if (.not. active(i)) goto 6980
           ia = ia + 1
           B(ia)     = B(i)
           INDCT(ia) = INDCT(i)
           BFCX0(ia) = BFCX0(i)
           AIMIN(ia) = AIMIN(i)
           AIMAX(ia) = AIMAX(i)
           PRICES(ia)= PRICES(i)
           do 6975 j = 1, n
              A(ia,j)      = A(i,j)
c             AXINC(ia,j)  = AXINC(i,j)
c             RA(ia,j)     = RA(i,j)
c             CMPMIN(ia,j) = CMPMIN(i,j)
6975       continue
           do 6977 k = 1, nx
              CTXPT(k,ia) = CTXPT(k,i)
6977       continue
c          CMPMIN(ia,N+1) = CMPMIN(i,N+1)
6980    continue
c.....................
6990    M = nactiv
c.....................
        return
        end
c-----------------------------------------------------------------
c
        subroutine REORDR
c
c  Called once per search hp by XCP to reorder the variables.
c  Variables fixed at a value (by SHR(hp) bounds) go first.
c  THIS ROUTINE MODIFIES PROBLEM DATA.  (See Section 6.5)
c-----------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
        include '$DISK2:[SALTZ.ILP1]comxrun3.for'
        include '$DISK2:[SALTZ.ILP1]comxrun4.for'
C
        logical*2 fxdvar(nn)
        real*8  tC(nn),tX0(nn),tP(nn,nn),tDIRS(nn,nn)
        real*8  tABAL(nn,nn),tALPHA(nn,nn)
        real*8  tA(mm,nn),tLL(mm+1,nn),tUU(mm+1,nn),tCTXPT(nn,mm)
        integer tSUB(nn),tLONS(nn),tUPNS(nn),tLOBD(nn),tUPBD(nn)
        integer tXSTAR(nn),tINC(nn)
c       tVAR  = temporary copy of variable VAR
        if (hprint(18).eq.1) write (8,*) 'REORDR: reordering vars.'
c
c       varord: all fixed vars. first, followed by remaining vars.
        nfixed = 0
c
```

```
c       Check all the variables for being fixed
        do 6910 j = 1, n
           fxdvar(j) = .false.
           if (lons(j) .eq. upns(j)) then
              fxdvar(j) = .true.
              nfixed    = nfixed + 1
              varord(nfixed) = j
              if (hprint(18).eq.1)write(8,*)' Var',j,'fixed=',lons(j)
           endif
6910    continue
c
c       If nfixed = 0, then variables need not be reordered
        if (nfixed  .eq. 0) goto 6945
c
c       Otherwise, put free variables after fixed variables
        jfree = 0
        do 6915 j = 1, n
           if (.not. fxdvar(j)) then
              jfree = jfree + 1
              varord(nfixed+jfree) = j
           endif
6915    continue
        if (hprint(18).eq.1) write(8,*)' varord ',(varord(L),L=1,n)
c
c       Reorder global variables into temporary global vars.
        do 6930 j = 1, n

           jv = varord(j)
           tC(j)     = C(jv)
           tX0(j)    = X0(jv)
           tSUB(j)   = SUB(jv)
           tLONS(j)  = LONS(jv)
           tLOBD(j)  = LOBD(jv)
           tUPNS(j)  = UPNS(jv)
           tUPBD(j)  = UPBD(jv)
           tXSTAR(j)= IXSTAR(jv)
           tINC(j)   = INC(jv)
c
           do 6920 k = 1, nx
              tP(j,k)      = P(jv,k)
              tALPHA(j,k) = ALPHA(jv,k)
              tDIRS(j,k)   = DIRS(jv,k)
              tABAL(j,k)   = ABAL(jv,k)
6920       continue
c
           do 6925 i = 1, m
              tA(i,j) =  A(i,jv)
              tLL(i,j)= LL(i,jv)
              tUU(i,j)= UU(i,jv)
              tCTXPT(j,i)=CTXPT(jv,i)
6925       continue
           tLL(m+1,j)=LL(m+1,jv)
           tUU(m+1,j)=UU(m+1,jv)
6930    continue
```

```
c
c       Reorder global variables from temporary global vars.
        do 6940 j = 1, n

           C(j)      = tC(j)
           X0(j)     = tX0(j)
           SUB(j)    = tSUB(j)
           LONS(j)   = tLONS(j)
           LOBD(j)   = tLOBD(j)
           UPNS(j)   = tUPNS(j)
           UPBD(j)   = tUPBD(j)
           IXSTAR(j)= tXSTAR(j)
           INC(j)    = tINC(j)
c
           do 6932 k = 1, nx
              P(j,k)      = tP(j,k)
              ALPHA(j,k) = tALPHA(j,k)
              DIRS(j,k)  = tDIRS(j,k)
              ABAL(j,k)  = tABAL(j,k)
6932       continue
c
           do 6936 i = 1, m
              A(i,j) =  tA(i,j)
              LL(i,j)= tLL(i,j)
              UU(i,j)= tUU(i,j)
              CTXPT(j,i)=tCTXPT(j,i)
6936       continue
           LL(m+1,j) = tLL(m+1,j)
           UU(m+1,j) = tUU(m+1,j)
c
6940    continue
c
c       Recalculate other global variables (CMPMIN, INC, RA, AXINC)
        call MINCMP
        call ARATIO
c
6945    continue
        return
        end
```

```
c-----------------------------------------------------------------------
c
        subroutine XRUN
c
c Called by RUN to organize Exact Ceiling Point Algorithm (XCPA),
c given initial solution & value (X*,Z*).  (See Section 5.5, Step 3)
c
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun3.for'
        logical*2 hplist(mm)
        if (xprint(12).eq.1) write(9,*) 'XRUN: Z* = ',ZSTAR
c
        icase = 1
c       Initialize hplist = list of constraint hp's already searched
        do 6001 i = 1, m
           hplist(i) = .false.
6001    continue
c
c       Loop thru constraints, enumerating 1-Ceiling Points
        do 6080 i = 1, m
c
c          Select a search hyperplane (ihp)
           ihp = IXPICK(hplist)
           if (xprint(12).eq.1) write(9,*) 'XRUN:search ct =',ihp
           hplist(ihp) = .true.
           if (ihp .eq. 0)  goto 6090
c
c          Enumerate feasible 1-CP(ihp)'s better than incumbent
           call XCP(ihp)
c
           if (xprint(12).eq.1)write(9,*)'xcalls',(xcalls(L),L=1,n)
           if (zstar .ge. zup) goto 6090
           if (icase .eq. 2) goto 6090
c
c          Reoptimize (LPr)' = tightened LP-relaxation of ILP
           call XREOPT(ihp)
c
6080    continue
c
6090    continue
        return
        end
c-----------------------------------------------------------------------
c
        subroutine XCP(hp)
c
c Called by RUN & RUNCUT to find ceiling points wrt constraint (hp).
c (See Section 5.4)
c-----------------------------------------------------------------------

        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
```

```
      include '$DISK2:[SALTZ.ILP1]comxrun2.for'
      include '$DISK2:[SALTZ.ILP1]comxrun3.for'
      include '$DISK2:[SALTZ.ILP1]comxrun4.for'
      integer hp
      if (xprint(13).eq.1)
     -  write(9,*) 'XCP: Z* =',ZSTAR,' hp =',hp
c
      B(m) = -(zstar + all)
      do 6100 i = 1, m
         GAP(i,1) = B(i)
6100  continue
      if (xprint(13).eq.1)write (9,*) 'gap(,1)=',(gap(i,1),i=1,m)
c
      iempty = ISHR(hp)
      if (1 .eq. iempty) goto 6105
      if (xprint(8) .eq.0)write (*,*) 'XCP: NO INT. SOLNS. in SHR'
      if (xprint(13).eq.1)write (9,*) 'XCP: NO INT. SOLNS. in SHR'
      goto 6190
c
c     Put the fixed vars. first, if not searching I-Cut
6105  if (hp .ne. m-1) call REORDR
c
c     Initialize key arrays
      do 6110 j = 1, n
         upc(j)   = upbd(j)
         loc(j)   = lobd(j)
c        upc(j),loc(j) = upper & lower conditional bounds on X(j)
         first(j) = 0
         final(j) = 0
c        first(j),final(j) = interval over which X(j) will range
         ISN(j)   = 0
c        ISN      = current integer solution being spelled out
         newz(j)  = 0
c        newz(j)  = 1 => New Z* found with previous partial solution
         xcalls(j)= zero
c        xcalls(j)= counter for number of calls to XCB(j)
6110  continue
c
      lj = 1
      if (ICHKBD(lj) .eq. 0) goto 6190
      call LOOPBD(lj)
c
c.....  Main Enumeration Loop ......... (See Section 5.3) .........
c
6120  ISN(lj) = ISN(lj) + INC(lj)
c
c     Take a forward step & calc. conditional variable bounds
      lj = lj + 1
      call XCB(lj)
      if (ICHKBD(lj) .eq. 0) goto 6140
c
      if (lj .lt. n) then
c        Set the loop bounds and take a forward step
         call LOOPBD(lj)
         goto 6120
      endif
```

```
c
c       Feasible Completion
6130    continue
        iempty = ILASTV(hp,lj)
        if (iempty .eq. 0) goto 6190
c
c       Backtrack to lower level
6140    lj = lj - 1
        if (IBOTLP(lj) .eq. 1) goto 6120
        if (lj .lt. 2) goto 6190
        goto 6140
c
c..... End of Main Loop ..........
6190    continue
c
        return
        end
c-----------------------------------------------------------------------
c
        subroutine XCB(j)
c
c Called by XCP to compute bounds for X(j) conditioned on the values
c of {X(1), X(2),..., X(j-1)}.  Also called by IBOTLP after a new
c incumbent has been found.  X(j) corresponds to ISN(j)
c (See Section 5.4.1)
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun3.for'
        include '$DISK2:[SALTZ.ILP1]comxrun4.for'
        integer j
c
        if (xprint(14) .eq. 1) write (9,*) 'XCB: j =',j
        xcalls(j) = xcalls(j) + one
c       if (j .eq. 1) goto 6220
        if (ISN(j-1) .eq. first(j-1)) goto 6205
        if (newz(j) .eq. 1) goto 6205
c
c       Use additive form after first value of ISN(j-1)
        do 6203 i = 1, m
           gap(i,j) = gap(i,j) - axinc(i,j-1)
           if (a(i,j)) 6201, 6203, 6202
6201       LL(i,j) = LL(i,j) + RA(i,j)
           goto 6203
6202       UU(i,j) = UU(i,j) + RA(i,j)
6203    continue
        goto 6230
c
c       On first value of ISN(j-1), use multiplicative form
6205    do 6210 i = 1, m
           gap(i,j) = gap(i,j-1) - A(i,j-1)*ISN(j-1)
6210    continue
c
```

```
c       Compute conditional bounds wrt each constraint i
6220    do 6225 i = 1, m
           if (A(i,j)) 6221, 6225, 6222
6221       LL(i,j) = (gap(i,j) - CMPMIN(i,j+1))/A(i,j)
           goto 6225
6222       UU(i,j) = (gap(i,j) - CMPMIN(i,j+1))/A(i,j)
6225    continue
        if (xprint(14) .eq. 1) then
           write(9,*)'xcb:LLj',(LL(i,j),i=1,m+1)
           write(9,*)'xcb:UUj',(UU(i,j),i=1,m+1)
        endif
c
c       Pick the tightest condl. bound (row M+1 <--> uncdl. bds)
6230    maxarg = m+1
        minarg = m+1
        do 6250 i = 1, m
           if (A(i,j)) 6235, 6250, 6245
6235       if (LL(i,j) .gt. LL(maxarg,j)) maxarg = i
           goto 6250
6245       if (UU(i,j) .lt. UU(minarg,j)) minarg = i
6250    continue
        upc(j) = IRNDWN(UU(minarg,j))
        loc(j) = IRNDUP(LL(maxarg,j))
        return
        end
c-----------------------------------------------------------------------
c
         Integer Function ICHKBD(j)
c
c Called by XCP to check if ISN(j) is at its final bound.
c Returns 1 if loc(j) <= upc(j)  => take forward step
c         0 "    "    >   "      => take backward step
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comxrun3.for'
        include '$DISK2:[SALTZ.ILP1]comxrun4.for'
        integer j
c
c       if (xprint(14) .eq. 1)
c     - write (9,*) 'ICHKBD j=',j,' loc',loc(j),' upc',upc(j)
c
        ICHKBD = 1
        if (loc(j) .gt. upc(j)) then
c
c          Can backtrack to next value of ISN(j-1)
           ICHKBD = 0
           if (RA(minarg,j) .gt. RA(maxarg,j)) goto 6390
c
c          Can double backtrack to next value of ISN(j-2)
           lj = j - 1
        endif
c
6390    continue
        return
        end
```

```
c-----------------------------------------------------------------------
c
         subroutine LOOPBD(j)
c
c Called by XCP to set first and final values for level j.
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comxrun4.for'
        integer j
c
c       If INC(j) = -1, start enumerating from upper bound
c       INC(j) is set in ARATIO
        if (INC(j) .eq. -1) then
           first(j) = upc(j)
           final(j) = loc(j)
c
c       Otherwise, start enumerating from lower bound
        else
           first(j) = loc(j)
           final(j) = upc(j)
        endif
        ISN(j) = first(j) - INC(j)
        return
        end
c-----------------------------------------------------------------------
c
         Integer Function ILASTV(hp,j)
c
c Called by XCP when a feasible completion has been reached.
c Returns 0 if SHR(ns) or SHR(hp) is empty.
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun4.for'
        integer hp,j
        if (xprint(16).eq.1) write(9,*) 'LASTV:hp=',hp,' j=',j
c
c       Assign last component based on sign of C(n)
        ISN(n) = upc(n)
        if (C(n) .lt. zero) ISN(n) = loc(n)
        do 6500 i = 1,m
           gap(i,n+1) = gap(i,n) - A(i,n)*DBLE(ISN(n))
6500    continue
c
c       Calculate objective function value of new incumbent
        z = zero
        do 6505 ij = 1, n
           z = z + C(ij)*DBLE(ISN(ij))
6505    continue
        if (xprint(16) .eq. 1) then
           write(9,*) 'ILASTV: z=',z,' IS=',(ISN(ij),ij=1,n)
           write(9,*) 'ILASTV: gap=',(gap(i,n+1),i=1,m)
        endif
c
c       Call new incumbent routines
        ILASTV = INCMOD(hp,z)
        return
        end
```

```
c-----------------------------------------------------------------------
c
         Integer Function IBOTLP(j)
c
c Called by XCP. Return 0 if ISN(j) has reached its loop limit
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comxrun4.for'
        integer j
c
        IBOTLP = 1
        if (ISN(j) .eq. FINAL(j)) then
           IBOTLP = 0
           goto 6605
        else
           if (newz(j) .eq. 0) goto 6605
           if (j .eq. 1) goto 6605
c          O/w, New Z* just found and j .ne. 1, so get new conditional
c          bounds (See Section 5.4.3).
           call XCB(j)
           newz(j) = 0
        endif
c
6605    continue
        return
        end
c-----------------------------------------------------------------------
c
         Integer Function INCMOD(hp,val)
c
c Called by ILASTV whenever a new incumbent is found (fairly rare).
c Returns 0 if new (possibly smaller) SHR is empty; 1, otherwise.
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun4.for'
        integer hp,iret4
        real*8  val
        if (xprint(16).eq.1)write(9,*) 'INCMOD:hp',hp,' val ',val
c
        iret4 = 1
        delz = val - ZSTAR
        if (delz .lt. tol) goto 6790
c
c       We have an improved feasible solution
        ZSTAR = val
        do 6700 j = 1, n
           ixstar(j)= ISN(j)
           newz(j)  = 1
           gap(m,j) = gap(m,j) - delz
6700    continue
        gap(m,n+1) = gap(m,n+1) - delz
        if (xprint(8) .eq.0) write(*,*) ' New Z* =',ZSTAR
        if (xprint(16).eq.1) then
           write(9,*) ' New Z* = ',zstar
           write(9,*) ' New X* = ',(ixstar(j),j=1,n)
        endif
```

```
c
        B(m) = -(zstar + all)
        call OBJCUT(zstar + all)
        iret4 = ISHR(hp)
C
6790    INCMOD = iret4
        return
        end
c-------------------------------------------------------------------------
c
        integer function IXPICK(list)
c
c Called by XRUN to select search constraint hyperplane.
c Do not cut any deeper than CX = Z* hyperplane. (See Section 5.7)
c-------------------------------------------------------------------------
c$include:plist.for
c$include:comprt.for
c$include:comlp1.for
c$include:comrun1.for
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
        logical*2 list(mm)
c
c       list(i) = .true. => hp (i) has already been a search ct.
        if (xprint(17) .eq.1) write (9,*) 'IXPICK: zup= ',zup
c
        IXPICK = 0
        zbarmn = bigm
c
c       Find ct. (IXPICK) yielding largest decrease in zbar'
        do 6810 i = 1, m
           if (.not. BFCX0(i)) goto 6810
           if (list(i)) goto 6810
C          Opt. price on ct. i comes from final tableau.
C          (Works fine if ALL cts. are .GE. or ALL are .LE.)
           delb  = one + AIMAX(i)
           delz0 = DABS(PRICES(i))*delb
           zbar  = z0 - delz0
c
           if (zbar .lt. zbarmn) then
c          Have found a new lowest upper bound zbar'
              IXPICK = i
              zbarmn = zbar
           endif
6810    continue
c
        if (xprint(17) .eq. 1)
     -  write (9,*) 'IXPICK: prices=',(prices(i),i=1,m)
        if (IXPICK .eq. 0) GOTO 6840
C
c       Assume old case: decrease zbar as much as possible
        zup = DINT(zbarmn)
c       Check for new case: only decrease upper bound to Z*
        if (zup .gt. zstar) goto 6820
        zup = zstar
        AIMAX(IXPICK) = (Z0 - zstar)/DABS(prices(IXPICK))
```

```
c
c       Replace I-cut with ceiling point constraint <--> IXPICK
6820    B(m-1) = B(IXPICK) - AIMAX(IXPICK)
        do 6830 j = 1, n
           A(m-1,j) = -A(IXPICK,j)
6830    continue
        AIMAX(m-1) = AIMAX(IXPICK)
c
6840    if (xprint(17) .eq. 1)
     -  write (9,*) 'IXPICK ct: ',IXPICK,' => zup=',zup,' Z*=',zstar
        return
        end
c-----------------------------------------------------------------------
c
        subroutine XREOPT(hp)
c
c Called by XRUN to reoptimize (LPr)' after searching SHR(hp).
c (See Section 5.7)
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
        integer hp
        if (xprint(8).eq.0) write(*,*)'  XREOPT: hp=',hp
c
c       Translate constraint hyperplane (hp) by t
        M = M - 2
        t = AIMAX(hp) + one
        B(hp) = B(hp) - t
c
c       Find solution to (LPr)' as defined in Section 5.7
c       XBAR' = XBAR - t*BINVERSE(,hp)
c       ZBAR' = ZBAR - t*PI(hp)
c
        do 6850 i = 1, m
           XBAR(ibasis(i)) = XBAR(ibasis(i))-t*ABAR(i,INITBV(hp))
6850    continue
        do 6855 j = 1, n
           X0(j)  = XBAR(j)
6855    continue
        Z0 = Z0 - t*PRICES(hp)
c
c       Do some PRECUT-type operations on the data
        call OBJCUT(ZSTAR+one)
C       Reappend ceiling point condition & objective fn. constraints
        do 6860 j = 1, n
           A(m+1, j) = zero
           A(m+2, j) = -C(j)
6860    continue
        B(m+1) = zero
        B(m+2) = zero
        M = M + 2
        ZUP = DINT(Z0)
        if (xprint(8).eq.0) write(*,*)'  XREOPT: Z0=',Z0
C
        return
        end
```

```
c-----------------------------------------------------------------------
c
        function ISHR(hp)
c
c Called by XCP to calc. unconditional bounds = Search HyperRectangle
c Returns 0 if SHR is clearly empty, 1 if SHR is not necc. empty.
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
        integer hp,iret1
        if (xprint(9) .eq. 1) write(8,*) 'ISHR: hp =',hp
c
        if (hp .ne. m-1) goto 5005
c
c       CUTSHR:special SHR routine for Search I-Cut
        call CUTSHR
c
        iret1 = ISREND(hp)
        goto 5010
c
c       Search ct. is a functional ct. or objective fn. hp
5005    iret1 = ISHRSC(hp)
5010    ISHR = iret1
        return
        end
c-----------------------------------------------------------------------
c
        integer function ISHRSC(hp)
c
c Called by ISHR to calc. uncondl.bds. for functional search ct.(hp).
c Returns 0 if SHR is clearly empty, 1 if SHR is not necc. empty.
c (See Section 5.3)
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comrun1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
        include '$DISK2:[SALTZ.ILP1]comxrun3.for'
        include '$DISK2:[SALTZ.ILP1]comxrun4.for'
        integer hp,iret2
        real*8 pts(nn,3*nn),arow(nn),rmin,rmax
        if (xprint(9) .eq. 1)
     -      write (8,*) 'ISHRSC: hp,icase =',hp,icase
c
c       First, get the bounds for SHR(n-simplex)
        iret2 = ISRSIM(m)
        ti = aimax(hp)
c
c       Store coefficients of the search ct. hyperplane
        do 5050 j = 1, n
           arow(j) = A(hp,j)
5050    continue
c
```

```
      nrow = 0
c     Get points where kth extreme dir. meets ceiling point ct.(i')
      do 5060 k = 1, nx
         if (CTXPT(k,hp)) goto 5060
         nrow = nrow + 1
         t = bigm
         tdot = VDOT(n,arow,dirs(1,k))
         if (tdot .gt. .001) t = ti/tdot
         do 5055 j = 1, n
            pts(j,nrow) = X0(j) - t*dirs(j,k)
5055     continue
5060  continue
c
c     Add the alpha(k)'s and P(k)'s to the list of points
      do 5063 k = 1, n
         if (.not. CTXPT(k,hp)) goto 5063
         do 5062 j = 1, n
            pts(j,nrow+1) = alpha(j,k)
            pts(j,nrow+2) = P(j,k)
5062     continue
         nrow = nrow + 2
5063  continue
c
c     For each component j, find the min & max over all pts(,k)
      do 5080 j = 1, n
         rmin = pts(j,1)
         rmax = pts(j,1)
         do 5070 k = 2, nrow
            if (pts(j,k) .lt. rmin) rmin = pts(j,k)
            if (pts(j,k) .gt. rmax) rmax = pts(j,k)
5070     continue
c
c        Round these reals to integer bounds
         lobd(j) = IRNDUP(rmin)
         upbd(j) = IRNDWN(rmax)
c
c        Take intersection of n-simplex and uncdl. bds.
         lobd(j) = MAX0(lobd(j), lons(j))
         upbd(j) = MIN0(upbd(j), upns(j))
5080  continue
c
c     Check for icase = 2 => {shr(ns)} = {shr(hp)}
      if (icase .eq. 2) goto 5088
      do 5082 j = 1, n
         if ((upns(j).eq.upbd(j)).and.(lons(j).eq.lobd(j)) goto 5082
         icase = 3
         if (xprint(8) .eq. 0) then
            write(8,*) ' *** Case C: |shr(hp)| < |shr(ns)| ***'
            write(*,*) ' *** Case C: |shr(hp)| < |shr(ns)| ***'
         endif
         goto 5088
5082  continue
c
```

```
        icase = 2
c       In case 2, remove ceiling point constraint (i')
        B(m-1)    = zero
        do 5084 j = 1, n
           A(m-1,j) = zero
5084    continue
        AIMAX(m-1) = zero
c
5088    iret2 = ISREND(hp)
        do 5090 j = 1, n
           irange(j) = irange(j)*iret2
5090    continue
c
        ISHRSC = iret2
        return
        end
c-----------------------------------------------------------------------
c
        integer function ISREND(hp)
c
c Called by ISHR to complete calc. uncondl.bds.
c Returns 1 => SHR non-empty; = 0 => SHR empty
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
        include '$DISK2:[SALTZ.ILP1]comxrun3.for'
        integer hp
c
c       Find range(j) := [U(j)-L(j)]+1; size := product of Ranges
        size = one
        do 5100 j = 1, n
c          Prevent uncondl. bds. from exceeding [0, SUB]
           lobd(j) = MAX0(lobd(j), 0)
           upbd(j) = MIN0(upbd(j),SUB(j))
           irange(j) = 1+(upbd(j)-lobd(j))
           size = size*DBLE(irange(j))
5100    continue
        if (xprint(8) .eq. 0)
     -  write(*,*) '    ISREND: size =',size,' sizlim =',sizlim
c
        if (xprint(10) .eq. 1) then
           write (8,*) '  ----------------------'
           write (8,*) '  Search hp ',hp
           write (8,*) '  Pts in hp ',size
           do 5103 j = 1, n
              write (8,*) j,lobd(j),upbd(j)
5103       continue
           write (8,*) '  ----------------------'
        endif
c
c       Exit if the search hyperrectangle is empty
        ISREND = 0
        if (size .le. zero) goto 5140
c
```

```
c       If it is non-empty, (re)initialize key variables
        ISREND = 1
        if (hp .lt. m) then
c          Compute CMPMIN(i,j)
           call MINCMP
c
c          Initialize Conditional variable bounds wrt each ct.
           do 5120 i = 1, m+1
              do 5115 j = 1, n
                 LL(i,j) = DBLE(LOBD(j))
                 UU(i,j) = DBLE(UPBD(j))
5115          continue
5120       continue
           if (size .lt. sizlim) goto 5140
           STOP '  **** Search Rectangle size exceeds sizlim ****'
        endif
5140    continue
        return
        end
c-----------------------------------------------------------------------
c
        integer function ISRSIM(hp)
c
c Called by ISHRSC and CUTSHR to calculate unconditional bounds.
c Returns 1 => SHR non-empty; 0 => SHR empty.
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
        include '$DISK2:[SALTZ.ILP1]comxrun3.for'
        integer hp,iret3
        real*8  shr(2*nn,nn),shrmin,shrmax
        if (xprint(10) .eq. 1) write (8,*) 'ISRSIM: hp=',hp
c
c       Append P(j,k) to shr, the matrix of points used to form SHR
        nrow = 0
        do 5160 k = 1, nx
c          if hp = m, all adj. ext. pts. are used to form SHR(ns)
           if (hp .eq. m) goto 5148
           if (.not. ctxpt(k,hp)) goto 5160
5148       nrow = nrow + 1
           do 5150 j = 1, n
              shr(nrow,j) = P(j,k)
5150       continue
c
c          Append ALPHA(k) to shr if it's closer to X0 than P(k)
           if (DABS(X0(1) - ALPHA(1,k)) .lt.
     -         DABS(X0(1) - P(1,k))) then
              nrow = nrow + 1
              do 5155 j = 1, n
                 shr(nrow,j) = ALPHA(j,k)
5155          continue
           endif
5160    continue
c
```

```
c       Now get bounds from SHR
        do 5170 j = 1, n
           shrmin = shr(1,j)
           shrmax = shr(1,j)
           do 5165 i = 2, nrow
              if (shr(i,j) .lt. shrmin) shrmin = shr(i,j)
              if (shr(i,j) .gt. shrmax) shrmax = shr(i,j)
5165       continue
c
           lobd(j) = IRNDUP(shrmin)
           upbd(j) = IRNDWN(shrmax)
5170    continue
c
        iret3 = ISREND(hp)
        do 5175 j = 1, n
           irange(j) = irange(j)*iret3
5175    continue
c
        ISRSIM = iret3
        if (hp .eq. m) then
           do 5180 j = 1, n
              lons(j) = lobd(j)
              upns(j) = upbd(j)
5180       continue
        endif
c
        return
        end
c-----------------------------------------------------------------------
c
        subroutine MINCMP
c
c Called by ISREND & REORDR to calc. CMPMIN = Matrix of min. completions
c CMPMIN(i,j)= SUM from k=j+1 to n of Min{A(i,k)*UPBD(k),A(i,k)*LOBD(k)}
c            (= "w(i,j)" in Section 5.4)
c-----------------------------------------------------------------------
        include '$DISK2:[SALTZ.ILP1]plist.for'
        include '$DISK2:[SALTZ.ILP1]comprt.for'
        include '$DISK2:[SALTZ.ILP1]comlp1.for'
        include '$DISK2:[SALTZ.ILP1]comxrun2.for'
        include '$DISK2:[SALTZ.ILP1]comxrun3.for'
c
        do 5220 i = 1, m
c
c          (Initialize CMPMIN for the multirun case)
           CMPMIN(i,n+1) = zero
c
c          More efficient to start from last column & work toward first
           do 5215 j = n, 1, -1
              if (A(i,j)) 5205, 5205, 5210
c             if  A(i,j) < or = 0:
5205          CMPMIN(i,j) = CMPMIN(i,j+1) + A(i,j)*DBLE(upbd(j))
              goto 5215
c             if  A(i,j) > 0:
5210          CMPMIN(i,j) = CMPMIN(i,j+1) + A(i,j)*DBLE(lobd(j))
5215       continue
        if (xprint(10).eq.1)write(8,*)' CMPMIN',(cmpmin(i,j),j=1,n+1)
5220    continue
        return
        end
```

SWITCHES.DAT

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
10.D16  1.D10
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
```

1st row is hprint:	2nd row is xprint:
1. getabc (data echo)	1. Runcut
2. z2phase	2. Precut
3. zsolve	3. Objcut, Redcts
4. zsetrs, zpivot4	4. Cutpt,Cutpt2
5. setup	5. Cuthp
6. balas	6. Cutshr
7. hrun	7. Aratio
8. hsdir	8. 1=>MINIMAL SCREEN PRINTING
9. findfs	9. Ishr
10. fis2hp	10. Ishrsc,Isrend,Mincmp
11. iffeas	11. Bounds
12. phase3	12. Xrun
13. stayfs	13. Xcp
14. twovar, getfes	14. Xcb
15. hround, hrndpt	15. Ichkbd
16. VarOrder: 1)lo-hi; 2)hi-lo	16. Ilastv, Incmod
17. REDCTS: list active cts.	17. Ixpick
18. REORDR: fixed vars. first	18. 1 => STOP AFTER HRUN
19. HRUN: time/print Phases	19. 1 => STOP AFTER ICUT
20. HRUN: 1 => nhps = No.BFCX0	20. 1 => only use cutpt2
21. HRUN: 1 => avoid NCP check	21. Skip HRUN, start RUNCUT w/ Z*=0
22.	22.
23.	23.
24.	24.
25.	25.

3rd row: sizlim (for SHR size), callim (for XCALLS)--NOT USED

Note: for RANDOMLY GENERATED PROBLEMS: set hprint(16) = 2
for REALISTIC " : = 0

ILPDATA.DAT (sample file)

```
FC10.DAT
END
```

Input Format

```
m     n
a11   . . .  a1n    b1     {1,-1}   (1 => ≤ constraint; -1 => ≥)
 .    .         .     .       .
 .      .       .     .       .
 .         .    .     .       .
am1   . . .  amn    bm     {1,-1}
c1    . . .  cn   {1,-1}             (1 => maximize; -1 => min.)
```

Example: ("FC-10" from Trauth and Woolsey, 1969)

FC10.DAT

```
10   12
 9    7   16    8   24    5    3    7    8    4    6    5  110    1
12    6    6    2   20    8    4    6    3    1    5    8   95    1
15    5   12    4    4    5    5    5    6    2    1    5   80    1
18    4    4   18   28    1    6    4    2    9    7    1  100    1
-12   0    0    0    0    0    1    0    0    0    0    0    0    1
 0  -15    0    0    0    0    0    1    0    0    0    0    0    1
 0    0  -12    0    0    0    0    0    1    0    0    0    0    1
 0    0    0  -10    0    0    0    0    0    1    0    0    0    1
 0    0    0    0  -11    0    0    0    0    0    1    0    0    1
 0    0    0    0    0  -11    0    0    0    0    0    1    0    1
 0    0    0    0    0    0    1    1    1    1    1    1    1
```

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report SOL 88-20	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) An Exact Ceiling Point Algorithm for General Integer Linear Programming		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert M. Saltzman and Frederick S. Hillier		8. CONTRACT OR GRANT NUMBER(s) N00014-85-K-0343
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research - SOL Stanford University Stanford, CA 94305-4022		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 1111MA
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research - Dept. of the Navy 800 N. Quincy Street Arlington, VA 22217		12. REPORT DATE November 1988 13. NUMBER OF PAGES 76 Pages
		14. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

integer linear programming; general integer variables; exact algorithm; ceiling points; implicit enumeration; linear programming relaxation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(Please see other side)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Abstract

An Exact Ceiling Point Algorithm for General Integer Linear Programming

Robert M. Saltzman and Frederick S. Hillier

Stanford University, 1988

This report describes an exact algorithm for the pure, general integer linear programming problem (ILP). Common applications of this model occur in capital budgeting (project selection), resource allocation and fixed-charge (plant location) problems. The central theme of our algorithm is to enumerate a subset of all solutions called "feasible 1-ceiling points." A feasible 1-ceiling point may be thought of as an integer solution lying on or near the boundary of the feasible region for the LP-relaxation associated with (ILP). Precise definitions of 1-ceiling points and the role they play in an integer linear program are presented in a recent report by the authors. One key theorem therein demonstrates that all optimal solutions for an (ILP) whose feasible region is non-empty and bounded are feasible 1-ceiling points. Consequently, such a problem may be solved by enumerating just its feasible 1-ceiling points. Our approach is to implicitly enumerate 1-ceiling points with respect to one constraint at a time while simultaneously considering feasibility. Computational results from applying this incumbent-improving Exact Ceiling Point Algorithm to 48 test problems taken from the literature indicate that this enumeration scheme may hold potential as a practical approach for solving problems with certain types of structure.