A Heuristic Ceiling Point Algorithm
for General Integer Linear Programming

by
Robert M. Saltzman
and Frederick S. Hillier

TECHNICAL REPORT SOL 88-19
November 1988

Department of Operations Research
Stanford University
Stanford, CA 94305
A Heuristic Ceiling Point Algorithm
for General Integer Linear Programming

by
Robert M. Saltzman
and Frederick S. Hillier

TECHNICAL REPORT SOL 88-19
November 1988

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

Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do NOT necessarily reflect the views of the above sponsors.

Reproduction in whole or in part is permitted for any purposes of the United States Government. This document has been approved for public release and sale; its distribution is unlimited.

89 1 03 135
Abstract

A Heuristic Ceiling Point Algorithm
for General Integer Linear Programming

Robert M. Saltzman and Frederick S. Hillier
Stanford University, 1988

This report describes a heuristic algorithm for the pure, general integer linear programming problem (ILP). In attempting to quickly obtain a near-optimal solution (without concern for establishing optimality), the algorithm searches for a "feasible 1-ceiling point." 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 heuristic approach is based upon the idea that a feasible 1-ceiling point found relatively near the optimal solution for the LP-relaxation is apt to have a high (possibly even optimal) objective function value. Having applied this Heuristic Ceiling Point Algorithm to 48 test problems taken from the literature, it appears that searching for such 1-ceiling points usually does provide a very good solution with a moderate amount of computational effort.

Subject Classification for OR/MS Index: Programming – Integer Algorithms; Heuristic

Key Words: integer linear programming; general integer variables; heuristic algorithm; ceiling points; linear programming relaxation; enumeration algorithms
A Heuristic Ceiling Point Algorithm for General Integer Linear Programming

1. Introduction

This report describes a heuristic algorithm for the pure, general integer linear programming problem in \( m \) constraints and \( n \) variables \( x_j, j = 1, \ldots, n \), whose form is

\[
\text{Maximize } \, c^T x = z \\
\text{subject to } \, Ax \leq b \quad (ILP)
\]

\[ x \geq 0, \, x \text{ integer}, \]

where \( A \in \mathbb{R}^{m \times n}, \, b \in \mathbb{R}^m \) and \( c \in \mathbb{R}^n \). 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 \( FR = \{x \geq 0 | Ax \leq b\} \) 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 Taha (1975) or Garfinkel and Nemhauser (1972).

While an exact algorithm for \((ILP)\) guarantees convergence to an optimal solution, a heuristic algorithm attempts only to provide a high-quality solution. However, while the former may require a prohibitive amount of computing time to reach an optimal solution and prove its optimality, the latter is designed to speedily obtain a near-optimal solution without concern for establishing optimality. Recently, Lee and Guignard (1988) described an efficient heuristic algorithm for a special case of \((ILP)\), the multidimensional 0-1 knapsack problem. By contrast, our Heuristic Ceiling Point Algorithm is best suited for the general integer case. It attempts to quickly locate a feasible "1-ceiling point" with respect to one of the constraints binding at \( \tilde{x} \), the optimal solution for \((LP_R)\). To understand why this might be a reasonable approach, we briefly review some of the key concepts of Saltzman and Hillier (1988).
1. Introduction

An integer solution \( x \) is a 1-ceiling point with respect to the \( i^{th} \) constraint, denoted \( x = 1-CP(i) \), 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, i.e., \( x \in FR \), 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 \( j \). Saltzman and Hillier (1988) demonstrate 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 Heuristic Ceiling Point Algorithm has three main components or phases described, in turn, in Sections 2, 3 and 4. The first phase involves solving (LP_R) and extracting some information about the structure of the feasible region near \( \bar{x} \). The second phase seeks 1-CP(FR)'s by looking for 1-CP(i)'s with respect to an appropriately chosen constraint (i) and then checking for feasibility. The third phase attempts to improve upon a feasible integer solution found in the second phase by altering the value of one or two of its components to reach a higher-valued 1-CP(FR). Section 5 discusses criteria for when to terminate the algorithm, while Section 6 reports on our computational experience. Section 7 summarizes our findings, and is followed by two appendices. The first appendix gives the variable bounds and options used in the GAMS/ZOOM runs reported in Section 6, while the second lists the Fortran code implementation of the Heuristic Ceiling Point Algorithm.
2. Phase 1: Using the Linear Programming Relaxation

We first introduce some additional notation to facilitate the discussion which follows. Let \( \bar{z} \equiv c^T \bar{x} \) denote the optimal objective function value for \((LP_R)\) and \( \bar{A} \equiv \{i | a_i^T \bar{x} = b_i\} \) the set of constraints binding at \( \bar{x} \). Further, let \( FR \equiv \{x | a_i^T x \leq b_i, \forall i \in \bar{A}\} \) be the cone formed by the extreme rays of \( FR \) emanating from \( \bar{x} \). Also, the terms “search constraint” and “search constraint hyperplane” will be used interchangeably and be denoted by the same index. We assume that \( \bar{x} \) is not all-integer, for otherwise \( \bar{x} \) solves \((ILP)\).

Even though it is possible to construct an \((ILP)\) whose optimal solution \( x^* \) is arbitrarily far from \( \bar{x} \), it still seems to be a good idea in practice to search for \( x^* \) in the neighborhood of \( \bar{x} \). Several others have taken this approach, including Glover (1973) and Hillier (1969a) in the pure, general integer case, and Ibaraki, Ohashi and Mine (1974) and Faaland and Hillier (1979) in the mixed integer case. An outline of the Heuristic Ceiling Point Algorithm’s search for a good feasible integer solution is as follows. Start at \( \bar{x} \) and move away from \( \bar{x} \) on a constraint hyperplane (the current “search constraint hyperplane”) binding at \( \bar{x} \). While moving along the surface of the feasible region, periodically round a continuous solution (somehow) to a nearby integer solution. How this is accomplished is described in the next section. Of course, in \( \mathbb{R}^2 \) the constraint hyperplanes binding at \( \bar{x} \) coincide with the extreme rays of \( FR \). In higher dimensions, a search direction along a binding constraint hyperplane can be formed from among the several extreme rays defining the feasible portion of this constraint hyperplane. The main purpose of the first phase then is to provide the heuristic algorithm with \( \bar{x} \), the set of constraints binding at \( \bar{x} \), and the set \( \{d^1, d^2, \ldots\} \) of normalized extreme directions defining the cone \( FR \). This can be accomplished, for example, by applying the simplex method to \((LP_R)\).

3. Phase 2: Locating 1-Ceiling Points

Given the structure of the feasible region near \( \bar{x} \), as defined by the set of constraint
3. Phase 2: Locating 1-Ceiling Points

hyperplanes binding at $\bar{x}$ and the extreme directions emanating from $\bar{x}$, the second phase looks for 1-ceiling points with respect to one particular binding constraint. This section will describe (1) how a specific search constraint hyperplane $(h)$ is chosen, (2) how a search direction $d$ lying on $(h)$ is found, (3) how to move along the search constraint hyperplane $(h)$ in the direction $d$ and round to integer solutions, and finally, (4) whether or not the rounding procedure is guaranteed to find a 1-CP$(h)$.

3.1. Choosing a Search Constraint Hyperplane $(h)$

Depending on $c$, 1-ceiling points with respect to one constraint might tend to be higher-valued than those with respect to another constraint. In a maximization problem, the objective function decreases as we move away from $\bar{x}$ along every extreme direction $d^k$. The rate of change of the objective function per unit step taken away from $\bar{x}$ along $d^k$ is given by $\rho^k \equiv c^T d^k$ (since $\|d^k\| = 1$). We want to identify a constraint hyperplane $(h)$ along which the objective changes as little as possible, for 1-ceiling points with respect to this constraint are apt to have relatively high objective function values. Since the feasible portion of each binding constraint hyperplane in $\mathbb{R}^n$ is generated by nonnegative combinations of the (linearly independent) extreme directions emanating from $\bar{x}$, a reasonable choice for a search constraint hyperplane is that which has the minimum sum of rates $\rho^k$ over all extreme directions defining the hyperplane. Letting $E_i$ be the set of $(n - 1)$ extreme directions emanating from $\bar{x}$ which lie on the $i^{th}$ constraint hyperplane, our choice of search constraint hyperplane $(h)$ is such that

$$h \in \text{arg min}_{i \in E_i} \sum_{k \in E_i} \rho^k.$$ 

3.2. Specifying a Search Direction $d$

Having selected a search constraint hyperplane $(h)$ on or just below which we hope to locate 1-CP$(h)$'s, we need to specify a direction of movement along $(h)$ away from
Figure 1. Search direction $d = d^1 + d^2$.

$x$, denoted as the search direction $d$. There are a number of ways to construct such a search direction, but one which corresponds to the manner in which the search constraint hyperplane $(h)$ is chosen is simply to give equal weight to all of the extreme directions which generate the feasible portion of $(h)$, i.e., take

$$d = \sum_{k \in E_h} d^k.$$ 

In $\mathbb{R}^3$, for example, the search direction $d$ in Figure 1 runs midway between the two extreme directions $d^1$ and $d^2$ which delineate the feasible part of constraint hyperplane (2). An alternative method of selecting both a search constraint hyperplane and search direction would be to associate a nonnegative weight $\omega^k$ with the $k^{th}$ extreme direction and then use $\omega^k d^k$ instead of $d^k$ in the calculations of $\rho^k$, $h$ and $d$. Such weights might reflect a balance between feasibility and objective function considerations for each extreme direction.
### 3.3. Rounding From a Non-integer to an Integer Solution

Movement away from \( \bar{x} \) occurs parametrically on constraint hyperplane \((h)\) along the ray \( \bar{x} + \theta d \) by determining positive values of \( \theta \), say \( \theta^1, \theta^2, \theta^3, \ldots \), such that the point \( x^t \equiv \bar{x} + \theta^t d \) contains at least one integer component. A stopping point \( x^t \) occurs when the ray \( \bar{x} + \theta d \) meets a "coordinate hyperplane" of the form \( x_j = \text{integer} \). The heuristic algorithm stops at each \( x^t \) corresponding to a \( \theta^t \) and rounds the remaining non-integer components of \( x^t \) in a manner yielding an all-integer solution \( y \) which is at least feasible with respect to the search constraint hyperplane \((h)\). There are two key questions concerning one of these integer solutions \( y \). First, is \( y \) a 1-CP\((h)\)? Second, does \( y \) satisfy all of the other constraints? If the answer to both questions is yes then we have located a 1-CP\((FR)\), which is the goal of our heuristic approach. Before further examining these questions, the rounding procedure will be described.

The process begins by increasing \( \theta \) until reaching a point \( x^1 \equiv \bar{x} + \theta^1 d \) which possesses at least one integer component, say component \( l \). Essentially, \( l \) is the component of \( \bar{x} + \theta d \) which reaches an integer value first because either the fractional part of \( \bar{x}_l \) is close to 0 or 1, or the magnitude of \( d_l \) is large relative to the other components, or a combination of both. More precisely, \( l \) is determined as

\[
l \in \arg \min_{j \mid d_j \neq 0} \{ f_j/d_j \},
\]

where

\[
f_j \equiv \begin{cases} 
1 - (\bar{x}_j - |\bar{x}_j|), & \text{if } d_j < 0; \\
(|\bar{x}_j| - \bar{x}_j) - 1, & \text{if } d_j > 0.
\end{cases}
\]

Here, \( f_j \in (0, 1] \) and will equal 1 when \( \bar{x}_j \) is integer so that \( f_j \) measures the distance to the next integer coordinate hyperplane for \( x_j \) when moving in direction \( d \). Thus, \( \theta^1 \equiv f_l/d_l \) and \( x^1 \equiv \bar{x} + \theta^1 d \). On the next iteration, \( \theta^2 \) is found by replacing \( \bar{x} \) with \( x^1 \) in the definition of \( f_j \). In general, iteration \( t \) determines \( \theta^t \) and \( x^t \equiv x^{t-1} + \theta^t d \). If the \( \arg \min_j \{ f_j/d_j \} \) contains \( q > 1 \) elements, the next stopping point will contain \( q \) integral components instead of just one. In Figure 2, the search constraint hyperplane is \((1)\) and the search direction \( d \) is the extreme direction emanating from \( \bar{x} \) that coincides with \((1)\). The first stopping
3. Phase 2: Locating 1-Ceiling Points

point $x^1$ is the point on (1) having $x_1 = 2$, the second stopping point $x^2$ is the point on (1) having $x_1 = 1$, and the third stopping point $x^3$ is the point on (1) having $x_2 = 3$.

For ease of specifying a rule on how to round $x^t$ to an integer solution $y^t$, all constraints are first converted into $\leq$ form. This is done just after the $(LP_R)$ has been solved by multiplying through any $\geq$ constraints by $-1$. Recall from Section 1 that no equality constraints are assumed to be part of the formulation of $(ILP)$. The following rule for rounding the other components $j \not\in \arg\min_j f_j/d_j$ yields an integer solution $y^t$ that satisfies constraint $(h)$ because $a^T_h y^t \leq a^T_h x^t = b_h$:

$$y^t_j = \begin{cases} 
\lfloor x^t_j \rfloor, & \text{if } a^t_{hj} > 0; \\
\lfloor x^t_j + \frac{1}{2} \rfloor, & \text{if } a^t_{hj} = 0; \\
\lceil x^t_j \rceil, & \text{if } a^t_{hj} < 0. 
\end{cases} \quad (1)$$

Thus, when $a^t_{hj}$ is positive, rounding to the feasible side of constraint $(h)$ requires rounding the $j^{th}$ component of $x^t$ down, whereas when $a^t_{hj}$ is negative, rounding to the feasible side
of \((h)\) requires rounding the \(j^{th}\) component of \(x^{t}\) up. Of course, for any direction \(j\) in which \(a_{hj} = 0\), the \(j^{th}\) component of the continuous solution \(x^{t}\) can be safely rounded either up or down since the feasibility of \(y^{t}\) with respect to \((h)\) is unaffected by the value of its \(j^{th}\) component. Our rule in this case is to round \(x^{t}_{j}\) to the nearest integer. In the example of Figure 2, both components of \(x^{t}\) are rounded down for \(t = 1, 2, 3\) since both \(a_{11}\) and \(a_{12}\) are positive. It is worth emphasizing that other rounding schemes may also produce a solution which is feasible with respect to \((h)\); however, they are likely to require some comparison of the relative magnitudes of the search constraint’s coefficients. Whether the extra computational effort is worthwhile is an area that could be investigated in the future.

### 3.4. Results of the Rounding Procedure

Does the above rounding rule yield a solution which is a \(1-CP(h)\)? In \(\mathbb{R}^{2}\), the rounded solution \(y^{t}\) is guaranteed to be a \(1-CP(h)\). This is because one component of \(y^{t}\) is fixed at an integer while the other component is found by rounding the corresponding component of \(x^{t}\) up or down, so respectively decreasing or increasing this latter component of \(y^{t}\) by one yields a solution which violates \((h)\). In higher dimensions, however, the rounding rule does not guarantee that \(y^{t}\) is a \(1-CP(h)\) even though \(y^{t}\) is clearly nonnegative (since \(x^{t}\) is) and all-integer (by definition).

**Lemma 1.** For \(n \geq 3\), rounding the continuous solution \(x^{t} \in \mathbb{R}^{n}\) by the rule specified in (1) does not necessarily yield a solution \(y^{t}\) which is a \(1-CP(h)\).

**Proof:** From Definition 3.3 of Saltzman and Hillier (1988), an integer solution \(y^{t}\) is a 1-ceiling point with respect to a constraint \((h)\) if (1) \(a_{h}^{T}y^{t} \leq b_{h}\), and (2) \(a_{h}^{T}y^{t} + |a_{hj}| > b_{h}\) for at least one \(j\). Letting \(s_{h}(y^{t}) \equiv b_{h} - a_{h}^{T}y^{t}\) be the slack of \(y^{t}\) with respect to \((h)\), a necessary and sufficient condition for \(y^{t}\) to be a \(1-CP(h)\) is: \(0 \leq s_{h}(y^{t}) < \max_{j}|a_{hj}|\). To check whether or not this condition holds, let us first define \(\delta_{j} \equiv y^{t} - x^{t}\), i.e.,

\[
\delta_{j} \equiv \begin{cases} 
[x^{t}_{j}] - x^{t}_{j}, & \text{if } a_{hj} > 0; \\
[x^{t}_{j} + \frac{1}{2}] - x^{t}_{j}, & \text{if } a_{hj} = 0; \\
[x^{t}_{j}] - x^{t}_{j}, & \text{if } a_{hj} < 0.
\end{cases}
\]
Because $x^t$ satisfies constraint (h) exactly, we have $s_h(y^t) = b_h - a_h^T(x^t + \delta) = -a_h^T\delta = -\sum_j a_{hj}\delta_j \geq 0$. The last inequality follows since each $a_{hj}\delta_j \leq 0$. ↑ Thus, by its construction, $y^t$ always satisfies constraint (h), as asserted previously. Note that $|\delta_j| \in [0,1)$ for all $j$, so that on average, $|\delta_j| \approx \frac{1}{2}$. Then we might expect $s_h(y^t) \approx \frac{1}{2} \sum_j |a_{hj}|$, or $s_h(y^t) \approx \frac{1}{2}\|a_h\|_1$, half the $L_1$-norm of $a_h$. On the other hand, $\max_j |a_{hj}| \equiv \|a_h\|_{\infty}$, the $L_{\infty}$-norm of $a_h$. However, since $\|a_h\|_1 \geq \|a_h\|_2 \geq \ldots \geq \|a_h\|_{\infty}$, it is certainly possible that $\frac{1}{2}\|a_h\|_1 \geq \|a_h\|_{\infty}$. In this case, $s_h(y^t)$ may be larger than $\max_j |a_{hj}|$ and then the integer solution $y^t$ is not a $1$-$\text{CP}(h)$. With this procedure, the chances of rounding to a $1$-$\text{CP}(h)$ improve as the magnitude of the largest coefficient in constraint (h) increases relative to that of the average coefficient in constraint (h). ↑

Although we cannot guarantee that $y^t$ is a $1$-$\text{CP}(h)$ at any particular iteration $k$, it is likely that over the course of several iterations at least one of the $y^t$'s generated will be a $1$-$\text{CP}(h)$. The more important question is whether or not the integer solution $y^t$ satisfies all the other functional constraints. When $y^t$ is feasible but is not a $1$-$\text{CP}(h)$, it is frequently a 1-ceiling point with respect to some other constraint, in which case $y^t$ is a $1$-$\text{CP}(FR)$. If $y^t$ is not a 1-ceiling point with respect to some other constraint, the Phase 3 procedures described in the next section will locate another feasible solution related to $y^t$ which is a $1$-$\text{CP}(FR)$. In most of the test problems run, the first feasible $y^t$ turned out to be either a $1$-$\text{CP}(h)$ or a $1$-$\text{CP}(i)$ with respect to some other constraint binding at $\bar{x}$.

As alluded to above, it may be worthwhile computationally to do a more thorough search for $1$-$\text{CP}(h)$'s as we move along constraint hyperplane (h) because one such ceiling point is never "too far away". To be more precise, suppose that only the $j$th component of a stopping point $x^t$ is integer, i.e., $x_j^t = K$; then, on the intersection of the feasible region with the coordinate hyperplane $x_j = K$, as many as half of the vertices of the $(n-1)$-dimensional unit hypercube about $x^t$ with all-integer vertices are $1$-$\text{CP}(h)$'s. This was seen in $\mathbb{R}^2$ at the beginning of this subsection and will be shown for $n \geq 3$ in the next theorem.

↑ For any $j$ such that $a_{hj} = 0$, $y_j$ does not affect the feasibility of $y$ with respect to (h), so $\delta_j$ may be set to 0. Also, note that $\delta_j = 0$ for all $j \in \arg\min_j \{f_j/d_j\}$.
Theorem 2. For $n \geq 3$, let $x \in \mathbb{R}^n$ be any point containing one integer component and satisfying constraint $(h)$ with equality. Then, ignoring the dimension corresponding to the integer-valued component of $x$, the number of $1$-$CP(h)$'s contained in the unique $(n - 1)$-dimensional $UHC[x]$ is a strictly positive integer not exceeding $2^{n-2}$.

Proof: The proof will be by induction on $n$, the dimension of $x$, beginning with $n = 3$. Being interested in $1$-ceiling points, we may confine the discussion to the all-integer vertices of $UHC[x]$. For $n = 3$, there are four cases to consider corresponding to the number of feasible vertices of a 2-dimensional $UHC[x]$, as shown in Figure 3. The arrows in the figure point to the feasible side of the constraint.

Case I. Exactly 1 of the four vertices of $UHC[x]$ is feasible. Since a unit step from the feasible vertex along an edge of $UHC[x]$ leads to one of the other three infeasible vertices, the feasible vertex is a $1$-$CP(h)$.
3. Phase 2: Locating 1-Ceiling Points

Case II. Exactly 2 of the four vertices of $UHC[z]$ are feasible. By the linearity of constraint $(h)$, each of the two feasible vertices is adjacent to one of the other two infeasible vertices. Hence, each feasible vertex is a 1-CP($h$).

Case III. Exactly 3 of the four vertices of $UHC[z]$ are feasible. Only two of the three feasible vertices are adjacent to the one infeasible vertex. Hence, only two of the feasible vertices of $UHC[z]$ are 1-CP($h$)'s.

Case IV. All four vertices of $UHC[z]$ are feasible. There are 2 subcases:

A. Constraint $(h)$ passes through a single vertex $v$ of $UHC[z]$. If $|a_{h1}| = |a_{h2}|$, then only $v$ is a 1-CP($h$) since both $(v - e_1) + e_2$ and $(v - e_2) + e_1$ are feasible with respect to $(h)$. If $|a_{h1}| \neq |a_{h2}|$, both $v$ and one of the two vertices adjacent to $v$ are 1-CP($h$)'s, but the other two vertices are not. For example, if $a_{h2} > a_{h1} > 0$, then the vertex $v - e_1$ is a 1-CP($h$) since $(v - e_1) + e_2$ violates $(h)$; however, vertex $v - e_2$ is not a 1-CP($h$) since $(v - e_2) + e_1$ satisfies $(h)$ while vertex $v - e_1 - e_2$ is clearly not a 1-CP($h$).

B. Constraint $(h)$ coincides with an edge of $UHC[z]$. The two vertices on this edge satisfy constraint $(h)$ with equality, so they both are 1-CP($h$)'s. The other two feasible vertices, however, are adjacent only to other feasible vertices and therefore are not 1-CP($h$)'s.

Thus, when $n = 3$, either one or two vertices of the 2-dimensional $UHC[z]$ are 1-CP($h$)'s. To proceed by induction, now assume the theorem holds when the number of dimensions is $3, 4, ..., n$, and consider whether it holds when the number of dimensions is $n + 1$. Consider the unique $((n + 1) - 1)$-dimensional $UHC[z]$. Since we are only interested in the vertices of this $UHC[z]$, this $n$-dimensional $UHC[z]$ may be examined as a pair of $(n - 1)$-dimensional $UHC[z]$'s. By the induction hypothesis, each of these $(n - 1)$-dimensional $UHC[z]$'s contains no more than $2^{n-2}$ vertices which are 1-CP($h$)'s. Therefore, the number of 1-CP($h$)'s contained in the unique $((n + 1) - 1)$-dimensional $UHC[z]$ is a strictly positive integer not exceeding $2^{n-2} + 2^{n-2} = 2^{n-1}$, which completes the proof by induction.

Since the stopping point $x^t$ contains $q \geq 1$ integer components, we restate the result of Theorem 2 in a slightly more general way. The proof is identical after ignoring these $q$
4. Phase 3: Improving Upon a Feasible Integer Solution

When the Phase 2 procedure described in the preceding section successfully locates a feasible integer solution, Phase 3 procedures are invoked to improve upon this solution, if possible, by altering either one or two of its components. Starting with a feasible solution, the procedures described in the next two subsections always identify a 1-CP(FR). Some other Phase 3 ideas and possible improvements are considered subsequently.

4.1. One-variable Changes to a Feasible Solution

Given a feasible integer solution \( y \), an attempt is first made to improve upon the objective value of \( y \) by altering just one of its components in a routine we shall refer to as “STAYFEAS.” In essence, STAYFEAS investigates all integer solutions of the form \( y + K e_j \), for all \( j = 1, \ldots, n \) and all integer values of \( K \). Let the nonnegative quantity \( s_i(y) \equiv b_i - a_i^T y \) be the slack on the \( i^{th} \) constraint when evaluated at \( y \). Also, let \( \delta_{ij} \equiv s_i(y)/a_{ij} \) be the maximum change that can be made to the \( j^{th} \) component of \( y \) without violating
4. Phase 3: Improving Upon a Feasible Integer Solution

the \(i^{th}\) constraint, i.e., \(y + \delta_{ij} e_j\) exactly satisfies (i). Sufficiently large movement away from an integer solution \(y\) in the plus \(j\) direction will eventually violate constraint (i) if \(a_{ij} > 0\), whereas sufficiently large movement away from \(y\) in the minus \(j\) direction will eventually violate (i) if \(a_{ij} < 0\). Thus, for each component of \(y\), there exists a range of allowable changes that may be made to this component without violating a given constraint. Specifically, \(y + \Delta e_j\) remains feasible with respect to constraint (i) for all \(\Delta \in [l_{ij}, r_{ij}]\), where this interval of acceptable changes is defined to be

\[
[l_{ij}, r_{ij}] = \begin{cases} 
(-\infty, \delta_{ij}], & \text{if } a_{ij} > 0; \\
(-\infty, +\infty), & \text{if } a_{ij} = 0; \\
[\delta_{ij}, +\infty), & \text{if } a_{ij} < 0.
\end{cases}
\]

Whether it is beneficial to increase or decrease \(y_j\) depends upon the sign of \(c_j\). If \(c_j\) is positive, increasing \(y_j\) will improve the objective, whereas if \(c_j\) is negative, decreasing \(y_j\) will improve the objective. When \(c_j = 0\), no attempt is made to alter the value of \(y_j\) since doing so will not improve the objective function. Suppose that \(c_j > 0\). The largest permissible change in \(y_j\) is the tightest (smallest) right endpoint \(r_{ij}\) among all those found for the respective constraints. Being interested in feasible integer changes only, this quantity is rounded down and is denoted as \(\Delta_j\). When \(c_j < 0\), the largest permissible change in absolute value to \(y_j\) is based on the largest left endpoint \(l_{ij}\) among all those found for the respective constraints. More precisely, we define the largest change that may be made to \(y_j\) and still yield an integer solution which is feasible with respect to all constraints as

\[
\Delta_j = \begin{cases} 
\min_l r_{ij}, & \text{if } c_j > 0; \\
\max\{\max_l l_{ij}, -y_j\}, & \text{if } c_j < 0.
\end{cases}
\]

In the latter case, \(\Delta_j\) is negative and so is not allowed to exceed \(y_j\) in absolute value in order to keep \(y_j + \Delta_j \geq 0\).

Once \(\Delta_j\) has been computed for each \(j\), the last step is to choose the best component to change based on the amount of improvement it makes in the objective function (if any). We select component \(l\) to change by the amount \(\Delta_l\), where \(l \in \arg\max_j \{c_j \Delta_j\}\), provided that \(c_l \Delta_l\) is positive. Otherwise, no alteration of any single component of \(y\) leads to a feasible integer solution with an objective value superior to that of \(y\).
4. Phase 3: Improving Upon a Feasible Integer Solution

Table I(a). Example of the One-variable Change Routine STAYFEAS.

<table>
<thead>
<tr>
<th>(i)</th>
<th>$a_{i,1}$</th>
<th>$a_{i,2}$</th>
<th>$a_{i,3}$</th>
<th>$a_{i,4}$</th>
<th>$b_i$</th>
<th>$s_i(y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>-1</td>
<td>1</td>
<td>-2</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>(2)</td>
<td>2</td>
<td>-2</td>
<td>7</td>
<td>-1</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>(3)</td>
<td>1</td>
<td>2</td>
<td>-6</td>
<td>0</td>
<td>23</td>
<td>5</td>
</tr>
<tr>
<td>$c_j$</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table I(b). Intervals of Acceptable Changes.

<table>
<thead>
<tr>
<th>(i)</th>
<th>$[l_{i,1}, r_{i,1}]$</th>
<th>$[l_{i,2}, r_{i,2}]$</th>
<th>$[l_{i,3}, r_{i,3}]$</th>
<th>$[l_{i,4}, r_{i,4}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>$(-1, \infty)$</td>
<td>$(-\infty, 1)$</td>
<td>$(-2, \infty)$</td>
<td>$(-\infty, 1)$</td>
</tr>
<tr>
<td>(2)</td>
<td>$(-\infty, 0]$</td>
<td>$[0, \infty)$</td>
<td>$(-\infty, 0)$</td>
<td>$[0, \infty)$</td>
</tr>
<tr>
<td>(3)</td>
<td>$(-\infty, 5]$</td>
<td>$(-\infty, \frac{5}{2})$</td>
<td>$[-\frac{5}{2}, \infty)$</td>
<td>$(-\infty, \infty)$</td>
</tr>
</tbody>
</table>

| $\Delta_j$ | 0 | 1 | 0 | 1 |
| $c_j\Delta_j$ | 0 | 8 | 0 | 3 |

Table I illustrates this procedure for a problem in which $y = (8, 35, 10, 0)$ is the given feasible integer solution. The data of the problem (taken from Garfinkel and Nemhauser 1972, p. 328) are given in Table I(a), along with the slack $s_i(y)$ in each of the three constraints when evaluated at the near-optimal point $y$. Table I(b) shows the intervals of acceptable change with respect to the individual constraints, as defined by (2). Since all objective function coefficients $c_j$ are positive, the largest acceptable change $\Delta_j$ for each component $j$ is found by taking the minimum of the corresponding right endpoints $r_{ij}$. The second to last row of Table I(b) shows that only the second and fourth components may change and yield a higher-valued feasible integer solution than $y$. Finally, since $c_2\Delta_2 > c_4\Delta_4$, component 2 is selected to change by $\Delta_2 = 1$. The resultant solution $y' \equiv y + \Delta_2e_2 = (8, 36, 10, 0)$ is a 1-ceiling point with respect to constraint (1) since constraint (1) provided the smallest upper bound $r_{i,j}$ for component 2. In fact, whenever this procedure is executed, it identifies a feasible 1-CP(i), as demonstrated in the following lemma.
4. Phase 3: Improving Upon a Feasible Integer Solution

Lemma 3. If the procedure STAYFEAS is successful in locating a feasible integer solution better than $y$, it locates one which is a $1-CP(FR)$. If unsuccessful, then $y$ itself is a $CP(FR)$.

Proof: First, the resultant point $y' \equiv y + \Delta_i e_i$ is feasible, by construction of $\Delta_i$. Second, $y'$ is "as close as possible" to the argmin or argmax constraint $(i^*)$ found in the computation of $\Delta_i$ in the sense that either $y' + e_i$ violates constraint $(i^*)$ (if $\Delta_i > 0$) or $y' - e_i$ violates $(i^*)$ (if $\Delta_i < 0$). Therefore, whenever $\Delta_i \neq 0$, the procedure is successful and the resultant solution $y'$ is a feasible $1-CP(i^*)$, i.e., $y' = 1-CP(FR)$. If the procedure is unsuccessful, i.e., $\Delta_i = 0$, then $\Delta_j = 0, \forall j$, implying that for each $j$ there is some constraint $(i)$ which is violated by either $y + e_j$ (if $a_{ij} > 0$) or by $y - e_j$ (if $a_{ij} < 0$). Thus, $y = CP(FR)$ by Definition 1 of Saltzman and Hillier (1988).

The one-variable change routine STAYFEAS is used as a subroutine of the two-variable change routine discussed next.

4.2. Two-variable Changes to a Feasible Solution

Given a feasible integer solution $y$, a slightly more involved effort is also made in Phase 3 to improve upon $y$ by simultaneously altering two of its components. From among several possible strategies to alter two variables, the one that we employ in the Heuristic Ceiling Point Algorithm is as follows. The first component to change, say $y_j$, is modified by either $+1$ or $-1$. If the resultant point $y' \equiv y_j^+$ or $y' \equiv y_j^-$ is feasible, the STAYFEAS procedure described in the preceding section is called upon to attempt to change another component $k \neq j$ of the feasible solution $y'$ to reach a $1-CP(FR)$ whose value exceeds that of $y$. If $y'$ is not feasible, a second procedure referred to as "GAINFEAS" (described subsequently) is called upon to attempt to change another component $k \neq j$ of the infeasible point $y'$ to reach a feasible solution whose value exceeds that of $y$. After all such two-variable changes have been tried, select that combination which leads to the greatest improvement over the objective function value of $y$, if any.

The guiding principle behind our 2-variable change is that altering the first component,
4. Phase 3: Improving Upon a Feasible Integer Solution

$y_j$, should generally increase feasibility, while altering the second component, $y_k$, should improve the objective function. Whether $y_j$ is to be increased or decreased by one depends upon the signs of the coefficients in the $j^{th}$ column of $A$. For instance, if the entries in the $j^{th}$ column of $A$ are all nonnegative, then $y_j^-$ has greater feasibility than $y$ in the sense that $s_i(y_j^-) \geq s_i(y), \forall i$. The rule used is: decrease $y_j$ by one if the sum of the coefficients in the $j^{th}$ column of $A$ is positive and $y_j > 0$, since $\sum_i s_i(y_j^-) > \sum_i s_i(y)$; otherwise, increase $y_j$ by one. (It may be worthwhile to separately try both increasing and decreasing the first variable to change, if possible. \dagger)

The routine GAINFEAS is similar in spirit to STAYFEAS in that it attempts to change just one component of the current solution in order to reach a feasible 1-$CP(i)$; the main difference is that GAINFEAS starts from a solution $y'$ which is not feasible. Let $V \equiv \{i|s_i(y') < 0\}$ be the set of constraints violated by $y'$ and $S \equiv \{i|s_i(y') \geq 0\}$ be the set of constraints satisfied by $y'$. For $i \in V$, interpret $\delta_{ik} \equiv s_i(y')/a_{ik}$ as the minimum change that must be made to the $k^{th}$ component of $y'$ in order to satisfy the $i^{th}$ constraint. For $i \in S$, the quantity $\delta_{ik}$ has the same meaning as it did previously in STAYFEAS, namely, the maximum change that can be made to the $k^{th}$ component of $y'$ without violating the $i^{th}$ constraint. In either case, $y' + \delta_{ik} e_k$ exactly satisfies (i). As before, there exists a range of changes to $y_k'$ which lead to a solution that is feasible with respect to the $i^{th}$ constraint. More precisely, $y' + \Delta e_k$ satisfies (i) for all $\Delta \in [l_{ik}, r_{ik}]$ where

$$[l_{ik}, r_{ik}] \equiv \begin{cases} \langle -\infty, \delta_{ik} \rangle, & \text{if } a_{ik} > 0; \\ \langle -\infty, +\infty \rangle, & \text{if } a_{ik} = 0; \\ \delta_{ik}, +\infty \rangle & \text{if } a_{ik} < 0. \end{cases}$$

However, note that $\delta_{ik}$ is negative when $a_{ik} > 0$ and $i \in V$, and $\delta_{ik}$ is positive when $a_{ik} < 0$ and $i \in V$ because $s_i(y') < 0, \forall i \in V$.

Now, for each component $k \neq j$, we find the allowable range of values that may be made to component $k$ so that all constraints currently violated by $y'$ become satisfied and all constraints currently satisfied remain so after movement away from $y'$. This is

\dagger Designing a specific heuristic algorithm involves making many choices among possible alternatives. The heuristic rules proposed here attempt to balance the tradeoff between computation time and objective function improvement, and have performed reasonably well on our test problems. For an excellent discussion on the design of heuristic algorithms, see Muller-Merbach (1981).
4. Phase 3: Improving Upon a Feasible Integer Solution

accomplished by taking the intersection over $i$ of all the ranges defined above. Let the resultant interval be denoted $[L_k, R_k]$, where $L_k \equiv [\max_i l_{ik}]$ and $R_k \equiv [\min_i r_{ik}]$. If $L_k > R_k$, then this interval is empty and it is not possible to reach a feasible solution by altering just one component of $y'$. Otherwise, the interval $[L_k, R_k]$ is nonempty and the sign of $c_k$ determines which endpoint of the interval to use to modify $y'_k$. When $c_k > 0$, the integral change to $y'_k$ yielding the greatest benefit to the objective function is $\Delta_k \equiv R_k$, whereas if $c_k < 0$, we take $\Delta_k \equiv L_k$. When $c_k = 0$, no attempt is made to alter the value of $y_k$ since doing so will not improve the objective function. In any case, if $\Delta_k$ is negative, it is not allowed to exceed $y_k$ in absolute value, in order to keep $y_L + \Delta_k \geq 0$.

Once $\Delta_k$ has been computed for each $k$, the component of $y'$ which is the best to change is selected based on the amount of improvement it makes in the objective function (if any). Component $l$ is chosen to change by the amount $\Delta_l$, where $l \in \arg \max_k c_k \Delta_k$. For each first component $j$ to change by $\pm 1$, a best second component $k \neq j$ is found by either the STAYFEAS or GAINFEAS procedure. Repeating this process for all first components $j = 1, \ldots, n$, the two components ultimately modified are those whose combined changes lead to the feasible solution with the greatest objective value, provided this value exceeds that of the original feasible integer solution $y$.

Lemma 4. If the procedure GAINFEAS is successful in locating a feasible integer solution better than $y$, it locates one which is a $1$-$CP(FR)$.

Proof: This follows because the resultant point $y'' \equiv y' + \Delta_l e_l$ is either found by STAYFEAS, which yields a $1$-$CP(FR)$ when it is successful (by Lemma 3) or by GAINFEAS. When this latter procedure is successful, it finds a point $y''$ which is "narrowly feasible" with respect to the constraint ($i^*$) having the largest left endpoint $l_{ij}$ (if $c_k < 0$) or smallest right endpoint $r_{ij}$ (if $c_k > 0$) in its range of acceptable changes for the second component. By narrowly feasible we mean that $s_{i^*}(y'') \geq 0$ but either $s_{i^*}(y'' + e_l) < 0$, if $a_{i^*l} > 0$, or $s_{i^*}(y'' - e_l) < 0$, if $a_{i^*l} < 0$. This results from $\Delta_l$ being the rounded version of either $l_{i^*l}$ or $r_{i^*l}$. Hence, $y''$ is a $1$-$CP(FR)$. 

Regardless of the success or failure of GAINFEAS, we know that the original point $y$ is a $1$-$CP(FR)$ if GAINFEAS is invoked at all because changing one of $y$'s components by
4. Phase 3: Improving Upon a Feasible Integer Solution

+1 or -1 led to the infeasible solution $y'$. 

4.3. Other Variations

Certainly other “Phase 3” procedures are possible. We now examine related strategies employed by some other researchers once they have located a feasible integer solution $y$ for a general integer linear programming problem. In a spirit similar to our one-variable change routine STAYFEAS, Echols and Cooper (1968) first make an effort to change one component of $y$ at a time to locate a solution which both improves the objective function and is close to a constraint. However, rather than picking the single best component of $y$ to change, they perform as many one-variable changes as possible, starting with component 1 and ending with component $n$. In their attempt to modify two components simultaneously, they initially alter just the first component $y_j$ by some fixed integer amount $Q$ in a direction that increases the objective function. If the resultant solution $y'$ is feasible, the change is made and the two-variable process repeats from $y'$ (without having altered the second component) starting with the next larger index $(j+1)$. On the other hand, if the resultant solution $y'$ is infeasible, the $k^{th}$ component $(k \neq j)$ of $y'$ is changed by the smallest integer amount necessary to satisfy the constraint most violated by $y'$. If the resultant solution $y''$ is feasible, the change is made and the two-variable process repeats from $y''$ starting with the next larger index $(j+1)$. Otherwise, the $k+1^{st}$ component of $y'$ is modified until all second components have been tried. If this step fails for all $k \neq j$, the two-variable routine is restarted altering the first component by $|Q| - 1$, and so forth. In a more time-consuming scheme, Echols and Cooper also attempt to modify three variables simultaneously. They reapply their two-variable change routine to all possible changes in components $j$ and $k$, seeking to locate a suitable third component $l \neq j, k$.

In his search for a feasible integer solution better than $y$, Hillier (1969a) alternates between a one-variable change routine and a two-variable change routine. In the first routine, he also computes a quantity like our $\delta_{ij}$ which measures the maximum change that can be made to the $j^{th}$ component of $y$ without violating the $i^{th}$ constraint while
improving upon the objective function. However, unlike the Heuristic Ceiling Point Algorithm, only unit changes are actually made to a particular variable. This logic is designed to keep the resultant solution away from the boundary of the feasible region, providing room for movement on subsequent iterations. Then, in attempting to modify two components simultaneously, Hillier considers pairs of variables such that the objective function coefficient $c_j$ corresponding to the first variable $y_j$ is larger in absolute value than that corresponding to the second variable $y_k$. He alters the first component $y_j$ by one unit (first in one direction and then in the other) and seeks a second component whose modification still yields a net improvement in the objective function. If an improved solution is found during this two-variable routine, the procedure returns to the one-variable change routine, repeating the process until no improved solution is found.

5. Termination Criteria

Assuming that the optimal solution for the linear programming relaxation of (ILP) is not all-integer, Phases 2 and 3 are executed. Being heuristic in nature, these procedures need some way of deciding when to stop iterating, particularly Phase 2, which is somewhat open-ended. This section first presents the criteria used for terminating the Phase 2 procedure and then the Phase 3 procedure.

Let $K_1, ..., K_6$ be constants which are each a function of the size of the problem. (Alternatively, they might instead be run-time parameters specified by the user.) The Phase 2 procedure moves along a particular search constraint hyperplane $(h)$ and uses a rounding procedure to locate a solution $y^t$ satisfying constraint $(h)$. If $y^t$ is feasible, then we have four possible reasons to stop iterating along this search constraint:

F1. $y^t = CP(FR)$, a ceiling point with respect to the feasible region (see Saltzman and Hillier 1988, Definition 1). In this case, we have located relatively near $\bar{x}$ an element of a class of points which contains an optimal solution (Saltzman and Hillier 1988, Theorem 1). Since it has been observed by Hillier (1969b, p. 640) that even for fairly large problems
5. Termination Criteria

there are relatively few solutions which are very close to being optimal, we stop iterating in Phase 2 when \( y' \) passes the following sufficient condition for being a \( CP(FR) \): for some \( i \), \( s_i(y') \equiv b_i - a_i'y' < \min_j |a_{ij}| \). This condition is sufficient since, for all \( j \), either \( y' + e_j \) or \( y' - e_j \) violates (i).

\[ F_2. \quad c^Ty' < c^Ty^* \text{ for some prior iteration } s \in \{1, \ldots, t-1\} \]  
In this case, it is likely (though not necessary) that integer solutions \( y^{t+1}, y^{t+2}, \ldots \) found along this search direction on subsequent iterations will have an objective function value no greater than that of one of the previous iterations. A new search constraint hyperplane is tried next.

\[ F_3. \quad c^Ty' = \bar{\bar{z}} \]  
While unlikely, it is possible for the feasible solution \( y' \) to be identified as optimal by virtue of having an objective value as large as \( \bar{\bar{z}} \), the upper bound for the value of the optimal solution of \( (ILP) \). (If not all \( c_j \) are integer, use \( \bar{z} \) instead of \( \bar{\bar{z}} \).)

\[ F_4. \]  
With \( y' \), we have identified what is believed to be an adequate number, say \( K_1 \), of feasible solutions from which to launch Phase 3. Phase 2 is terminated when \( y' \) is infeasible, another reason to stop iterating is:

\[ V_1. \]  
The sum of infeasibilities \( \sum_{i \in V} |s_i(y')| \) has not decreased for say, \( K_2 \), consecutive iterations. It appears that this search constraint hyperplane is not a fruitful one on which to continue searching for \( 1-CP(FR) \)'s. A new search constraint hyperplane is tried next.

In addition, we also terminate our search along this search constraint hyperplane and move to a new search constraint hyperplane if the number of iterations exceeds some limit \( K_3 \). Finally, after \( K_4 \) constraint hyperplanes have been searched in Phase 2, we proceed to Phase 3. Other untried possibilities are (1) to move along the same search constraint hyperplane but in a different search direction, and (2) to allow components of the search direction \( d \) to change on different iterations. This second possibility might be beneficial when moving along a search constraint hyperplane which suddenly becomes infeasible due to the presence of another constraint not binding at \( \bar{z} \).

Phase 3 is applied to each of the \( K_5 \) best feasible integer solutions found in Phase 2, working with one solution at a time. If Phase 3 is successful, stopping criteria F1 and F3 could be applied to terminate Phase 3. However, since these criteria are not satisfied
too frequently, we continue reapplying Phase 3 to the result of the previous iteration until Phase 3 no longer makes progress. Other untried possibilities are (1) to change the first component in the opposite direction from what is currently done, as in Hillier (1969a), and (2) to change the first component over a range of integer values \([-K_0, K_0]\), as in Echols and Cooper (1968), increasing the amount of time spent in this phase over that spent by the current method by a factor of \(2K_0\).

6. Computational Experience

This section presents our computational experience with the Heuristic Ceiling Point Algorithm, using the relevant parts of Crowder, Dembo, and Mulvey (1978, Appendix) 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 6.2 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 Heuristic Ceiling Point Algorithm are reported in subsection 6.3.

6.1. 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 Lustig (1987) 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.
6. Computational Experience

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 Heuristic 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.

6.2. Experimental Design

The main objective of our computational testing was to assess whether or not the heuristic methods for enumerating 1-ceiling points described in this report constitute a practical approach for approximately solving general integer linear programming problems. To assess the effectiveness of the Heuristic 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 II(a) and II(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
Table II(a). Realistic Test Problem Characteristics.

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

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

$^{(b)}$ Gives the duality gap in % terms: $100 \times (c^T \bar{x} - c^T x^*)/c^T \bar{x}$. 
Table II(b). Randomly Generated Test Problem Characteristics.

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

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

$^{(b)}$ Gives the duality gap in % terms: $100 \times (\tilde{z} - z^*)/\tilde{z}$. 
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 (1969). 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 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 published by Haldi (1964) 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 (1969b) and are fully specified in Hillier (May 1969). 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 III. 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 Markowitz and Manne (1957), II-M was solved as a general integer problem in Land and Doig (1960) as it is here.

Table III. Coefficient Ranges for Randomly Generated Test Problems.

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>I</th>
<th>II</th>
<th>III</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_j )</td>
<td>-20, 79</td>
<td>0, 99</td>
<td>0, 99</td>
</tr>
<tr>
<td>( a_{ij} )</td>
<td>-40, 59</td>
<td>0, 99</td>
<td>0, 1</td>
</tr>
<tr>
<td>( b_i )</td>
<td>500,999</td>
<td>1000,1999</td>
<td>1</td>
</tr>
<tr>
<td>( x_j )</td>
<td>general</td>
<td>general</td>
<td>binary</td>
</tr>
</tbody>
</table>

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 II(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.

It seems appropriate to evaluate heuristic algorithms based on two indicators of performance: the quality of the best solution found (\( x_H \)) and the CPU time spent in locating \( x_H \). Measuring quality without considering CPU time, or vice-versa, is misleading because an algorithm which spends a large amount of time to find a good suboptimal solution is not necessarily an efficient algorithm nor is one which finds a low-quality solution rather quickly. As a measure of the quality of a reported solution \( y \), we often use the normalized deviation of \( y \) from an optimal solution \( x^* \), i.e.,

\[
D(y, x^*) \equiv (c^T x^* - c^T y)/\|c\|_2,
\]

where \( \|c\|_2 \) is the Euclidean norm of \( c \). The normalized deviation measures the Euclidean distance of \( y \) to the hyperplane \( c^T x = c^T x^* \).
6. Computational Experience

For algorithms which solve the $(LPR)$ 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 $(LPR)$. 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 $(LPR)$. 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.

6.3. Results with the Heuristic Ceiling Point Algorithm

Tables IV(a) and IV(b) describe the performance of the various phases of the Heuristic Ceiling Point Algorithm on the realistic and randomly generated test problems, respectively. The second, third and fourth columns of each table give the fraction of the total CPU time spent in solving $(LPR)$, in Phase 2 and in Phase 3, respectively, of the Heuristic Ceiling Point Algorithm, where solving $(LPR)$ may be thought of as Phase 1. Recall that Phase 2 seeks feasible 1-CP(i)'s by moving along the surface of a constraint hyperplane binding at $\bar{z}$ and rounding to nearby integer solutions. The Phase 3 procedures alter either one or two components of a feasible integer solution found in Phase 2 in an attempt to locate a better 1-CP(FR). The last two columns give the quality of the best solution known by the end of Phases 2 and 3, respectively, as measured by their normalized deviation from optimality.

The Heuristic Ceiling Point Algorithm performed quite well on all three classes of realistic problems, locating an optimal solution for over half of the difficult FC and IBM test problems, and for all of the AL problems. Altogether, the Heuristic Ceiling Point Algorithm found an optimal solution for 16 of the 22 test problems which did not possess
Table IV(a). Performance of Ceiling Point Heuristic Algorithm.
on Realistic Problems.

<table>
<thead>
<tr>
<th>Problem</th>
<th>$L_{P_{R}}$</th>
<th>Phase 2</th>
<th>Phase 3</th>
<th>CPU time</th>
<th>Total Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC-1</td>
<td>66.7</td>
<td>14.3</td>
<td>19.0</td>
<td>0.21</td>
<td>0.000</td>
</tr>
<tr>
<td>FC-2</td>
<td>72.7</td>
<td>18.2</td>
<td>9.1</td>
<td>0.22</td>
<td>0.000</td>
</tr>
<tr>
<td>FC-3</td>
<td>81.3</td>
<td>12.5</td>
<td>6.2</td>
<td>0.16</td>
<td>0.000</td>
</tr>
<tr>
<td>FC-4</td>
<td>81.2</td>
<td>6.3</td>
<td>12.5</td>
<td>0.16</td>
<td>0.000</td>
</tr>
<tr>
<td>FC-5</td>
<td>48.4</td>
<td>35.5</td>
<td>16.1</td>
<td>0.31</td>
<td>1.154</td>
</tr>
<tr>
<td>FC-6</td>
<td>51.7</td>
<td>41.4</td>
<td>6.9</td>
<td>0.29</td>
<td>1.732</td>
</tr>
<tr>
<td>FC-7</td>
<td>46.4</td>
<td>32.1</td>
<td>21.4</td>
<td>0.28</td>
<td>1.154</td>
</tr>
<tr>
<td>FC-8</td>
<td>51.7</td>
<td>41.4</td>
<td>6.9</td>
<td>0.29</td>
<td>1.732</td>
</tr>
<tr>
<td>FC-9</td>
<td>77.3</td>
<td>18.2</td>
<td>4.5</td>
<td>0.22</td>
<td>0.000</td>
</tr>
<tr>
<td>FC-10</td>
<td>32.6</td>
<td>61.8</td>
<td>5.6</td>
<td>0.89</td>
<td>0.816</td>
</tr>
<tr>
<td>IBM-1</td>
<td>65.6</td>
<td>18.8</td>
<td>15.6</td>
<td>0.32</td>
<td>1.134</td>
</tr>
<tr>
<td>IBM-2</td>
<td>57.9</td>
<td>7.9</td>
<td>34.2</td>
<td>0.38</td>
<td>0.378</td>
</tr>
<tr>
<td>IBM-3</td>
<td>68.2</td>
<td>13.6</td>
<td>18.2</td>
<td>0.22</td>
<td>0.075</td>
</tr>
<tr>
<td>IBM-4</td>
<td>24.2</td>
<td>49.8</td>
<td>26.0</td>
<td>2.69</td>
<td>0.258</td>
</tr>
<tr>
<td>IBM-5</td>
<td>23.8</td>
<td>10.9</td>
<td>65.2</td>
<td>2.56</td>
<td>1.033</td>
</tr>
<tr>
<td>AL-55</td>
<td>27.5</td>
<td>64.2</td>
<td>8.3</td>
<td>1.09</td>
<td>0.375</td>
</tr>
<tr>
<td>AL-60</td>
<td>64.4</td>
<td>24.4</td>
<td>11.1</td>
<td>0.45</td>
<td>0.000</td>
</tr>
<tr>
<td>AL-65</td>
<td>46.9</td>
<td>18.8</td>
<td>34.4</td>
<td>0.64</td>
<td>0.125</td>
</tr>
<tr>
<td>AL-70</td>
<td>43.3</td>
<td>17.9</td>
<td>38.8</td>
<td>0.67</td>
<td>0.250</td>
</tr>
<tr>
<td>AL-75</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.28</td>
<td>0.000</td>
</tr>
<tr>
<td>AL-80</td>
<td>46.8</td>
<td>22.0</td>
<td>32.2</td>
<td>0.62</td>
<td>0.025</td>
</tr>
<tr>
<td>AL-85</td>
<td>46.9</td>
<td>20.3</td>
<td>32.8</td>
<td>0.64</td>
<td>0.075</td>
</tr>
<tr>
<td>AL-90</td>
<td>45.2</td>
<td>21.0</td>
<td>33.8</td>
<td>0.62</td>
<td>0.200</td>
</tr>
<tr>
<td>AL-100</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.29</td>
<td>0.000</td>
</tr>
</tbody>
</table>

(a) $D(x_2, x^*) = (c^T x_2 - c^T x^*)/||c||_2$, where $x_2$ is best Phase 2 solution.

(b) $D(x_H, x^*) = (c^T x_H - c^T x^*)/||c||_2$, where $x_H$ is best Phase 3 solution.
Table IV(b). Performance of Ceiling Point Heuristic Algorithm.
on Randomly Generated Problems.

<table>
<thead>
<tr>
<th>Problem</th>
<th>% of Total CPU time</th>
<th>Total</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$L_P^R$</td>
<td>Phase 2</td>
<td>Phase 3</td>
</tr>
<tr>
<td>I-1</td>
<td>32.9</td>
<td>20.1</td>
<td>47.0</td>
</tr>
<tr>
<td>I-2</td>
<td>13.9</td>
<td>13.7</td>
<td>72.4</td>
</tr>
<tr>
<td>I-5</td>
<td>11.1</td>
<td>24.8</td>
<td>64.1</td>
</tr>
<tr>
<td>I-6</td>
<td>22.1</td>
<td>17.6</td>
<td>60.3</td>
</tr>
<tr>
<td>II-1</td>
<td>30.8</td>
<td>14.0</td>
<td>55.2</td>
</tr>
<tr>
<td>II-2</td>
<td>43.7</td>
<td>16.5</td>
<td>39.8</td>
</tr>
<tr>
<td>II-3</td>
<td>41.0</td>
<td>13.9</td>
<td>45.1</td>
</tr>
<tr>
<td>II-4</td>
<td>36.9</td>
<td>17.1</td>
<td>46.0</td>
</tr>
<tr>
<td>II-5</td>
<td>34.8</td>
<td>14.4</td>
<td>50.9</td>
</tr>
<tr>
<td>II-6</td>
<td>18.4</td>
<td>8.6</td>
<td>73.0</td>
</tr>
<tr>
<td>II-7</td>
<td>41.7</td>
<td>17.5</td>
<td>40.8</td>
</tr>
<tr>
<td>II-8</td>
<td>44.7</td>
<td>18.4</td>
<td>36.9</td>
</tr>
<tr>
<td>II-9</td>
<td>31.3</td>
<td>12.0</td>
<td>56.7</td>
</tr>
<tr>
<td>II-10</td>
<td>22.8</td>
<td>9.7</td>
<td>67.5</td>
</tr>
<tr>
<td>II-11</td>
<td>21.1</td>
<td>8.8</td>
<td>70.1</td>
</tr>
<tr>
<td>II-12</td>
<td>34.4</td>
<td>14.2</td>
<td>51.4</td>
</tr>
<tr>
<td>II-13</td>
<td>22.9</td>
<td>22.6</td>
<td>54.4</td>
</tr>
<tr>
<td>II-14</td>
<td>10.3</td>
<td>9.3</td>
<td>80.4</td>
</tr>
<tr>
<td>II-M</td>
<td>30.5</td>
<td>24.2</td>
<td>45.3</td>
</tr>
<tr>
<td>III-2</td>
<td>49.0</td>
<td>22.0</td>
<td>29.0</td>
</tr>
<tr>
<td>III-3</td>
<td>37.5</td>
<td>55.0</td>
<td>7.5</td>
</tr>
<tr>
<td>III-4</td>
<td>25.3</td>
<td>56.0</td>
<td>18.7</td>
</tr>
<tr>
<td>III-5</td>
<td>56.3</td>
<td>32.4</td>
<td>11.3</td>
</tr>
<tr>
<td>III-8</td>
<td>52.9</td>
<td>25.0</td>
<td>22.1</td>
</tr>
</tbody>
</table>

$^{(a)} D(x_2, x^*) \equiv (c^T x_2 - c^T x^*) / ||c||_2$, where $x_2$ is best Phase 2 solution.

$^{(b)} D(x_H, x^*) \equiv (c^T x_H - c^T x^*) / ||c||_2$, where $x_H$ is best Phase 3 solution.
an all-integer $\tilde{x}$ (without proving optimality). In the remaining six problems, the best solution found by the Heuristic Ceiling Point Algorithm at the end of Phase 3 differed in objective function value from optimality by at most two in absolute value, although these gaps appear to be fairly large in terms of normalized deviation. Phase 2 managed to locate an optimal solution for 6 of the problems. For all 16 of the remaining problems, the Phase 3 procedures were effective, locating an optimal solution for 10 of these problems. In terms of CPU time, both Phase 2 and Phase 3 required less time than that required to solve the LP-relaxation on all but four of the problems \{FC-10, IBM-4, IBM-5, AL-55\}.

On the randomly generated problems, the Heuristic Ceiling Point Algorithm performed reasonably well overall, but the level of success varied noticeably with the type of problem. It located an optimal solution for all five of the Type III problems, for six of the fifteen Type II problems, but for none of the more difficult Type I problems. Altogether, the Heuristic Algorithm found an optimal solution for 11 of the 24 randomly generated test problems. In the remaining 13 problems, the normalized deviation from optimality of the best solution found was relatively small on all but one (II-1) of the Type II problems, but rather large on all but one (I-6) of the Type I problems. Here, Phase 2 managed to locate an optimal solution for only 3 of the 24 test problems, all of Type III. Phase 3 proved to be effective on 19 of the 21 remaining problems, locating an optimal solution for 8 of these. While Phase 2 never required more time than that needed to solve \((LP_R)\), at least on Type I and Type II problems, the reverse is true of Phase 3, i.e., Phase 3 always required more CPU time than that needed to solve \((LP_R)\) for Type I and Type II problems. In general, we would expect the fraction of total time spent in Phase 3 to increase as the number of variables \((n)\) increases since the execution time of Phase 3 (which calls upon the two-variable change routine TWOVAR) seems to grow with the square of \(n\), while that of the simplex method and of Phase 2 grow more or less linearly with \(n\).

Kochenberger, McCarl and Wyman (1974) are responsible for the only published results known to the authors of a heuristic algorithm being applied to a majority of the realistic test problems from Trauth and Woolsey (1969). Since only averaged results for each class of problems are presented in Kochenberger, McCarl and Wyman (1974), we decided to run the test problems with a widely available package called the Generalized...
Algebraic Modeling System (GAMS, Version 2.04) developed by Brooke, Kendrick and Meeraus (1988). 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 (1980) 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 the specified values of the GAMS/ZOOM run-time options. The performances of the Pivot & Complement heuristic scheme employed by GAMS/ZOOM and the Heuristic Ceiling Point Algorithm (HCPA) are shown in Table V(a). Entries in the column labeled \(z_H\) give the objective function value of the best solution found by the heuristic algorithm, while those in the column labeled \('%\ Opt.'\) represent the percentage deviation of \(z_H\) from the optimal objective function value \(z^*\), defined to be \(100 \times [1 - |(z^* - z_H)/z^*|]\).

The next table, Table V(b), compares the performance of the Heuristic Ceiling Point Algorithm with that of two other heuristic algorithms on the set of randomly generated problems. The first is again the Pivot & Complement heuristic of GAMS/ZOOM while the second is due to Hillier (1969a). Both the Heuristic Ceiling Point Algorithm and GAMS/ZOOM were executed on the VaxStationII microcomputer whereas Hillier’s algorithm was executed on an IBM-360/67 mainframe computer. A knowledgeable computer scientist informed us that these two machines perform roughly the same number of operations per second, despite vast differences in age and architecture (Saunders, 1988). In contrast to the Heuristic Ceiling Point Algorithm, Hillier’s heuristic procedure (1-2A-1) seeks feasible integer solutions while moving along a path strictly interior to the feasible region. On the Type I problems, Hillier’s procedure appears to enjoy much greater success than the Heuristic Ceiling Point Algorithm both in terms of the quality of its best solution and the speed with which it finds this solution. On the Type II and Type III problems, these two algorithms are about equally successful in terms of the quality of the best solution found. However, based on the ratios of total CPU time to CPU time spent solving \((LP_R)\), Hillier’s procedure is probably much faster than the Heuristic Ceiling Point Algorithm.
Table V(a). Comparison of Heuristic Algorithms on Realistic Problems.

<table>
<thead>
<tr>
<th>Problem</th>
<th>CPU time</th>
<th>$z_H$</th>
<th>% Opt.</th>
<th>CPU time</th>
<th>$z_H$</th>
<th>% Opt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC-1</td>
<td>0.21</td>
<td>7</td>
<td>100.0</td>
<td>1.27</td>
<td>6</td>
<td>85.7</td>
</tr>
<tr>
<td>FC-2</td>
<td>0.22</td>
<td>8</td>
<td>100.0</td>
<td>1.40</td>
<td>7</td>
<td>87.5</td>
</tr>
<tr>
<td>FC-3</td>
<td>0.16</td>
<td>10</td>
<td>100.0</td>
<td>1.27</td>
<td>8</td>
<td>80.0</td>
</tr>
<tr>
<td>FC-4</td>
<td>0.16</td>
<td>8</td>
<td>100.0</td>
<td>1.29</td>
<td>6</td>
<td>75.0</td>
</tr>
<tr>
<td>FC-5</td>
<td>0.31</td>
<td>75</td>
<td>98.7</td>
<td>2.22</td>
<td>66</td>
<td>86.8</td>
</tr>
<tr>
<td>FC-6</td>
<td>0.29</td>
<td>105</td>
<td>99.1</td>
<td>2.08</td>
<td>80</td>
<td>75.5</td>
</tr>
<tr>
<td>FC-7</td>
<td>0.28</td>
<td>75</td>
<td>98.7</td>
<td>2.18</td>
<td>66</td>
<td>86.8</td>
</tr>
<tr>
<td>FC-8</td>
<td>0.29</td>
<td>105</td>
<td>99.1</td>
<td>2.04</td>
<td>80</td>
<td>75.5</td>
</tr>
<tr>
<td>FC-9</td>
<td>0.22</td>
<td>9</td>
<td>100.0</td>
<td>1.58</td>
<td>8</td>
<td>88.9</td>
</tr>
<tr>
<td>FC-10</td>
<td>0.89</td>
<td>15</td>
<td>88.2</td>
<td>4.90</td>
<td>13</td>
<td>76.5</td>
</tr>
<tr>
<td>IBM-1</td>
<td>0.32</td>
<td>-9</td>
<td>87.5</td>
<td>1.52</td>
<td>-12</td>
<td>50.0</td>
</tr>
<tr>
<td>IBM-2</td>
<td>0.38</td>
<td>-7</td>
<td>100.0</td>
<td>1.94</td>
<td>-10</td>
<td>57.1</td>
</tr>
<tr>
<td>IBM-3</td>
<td>0.22</td>
<td>-187</td>
<td>100.0</td>
<td>1.38</td>
<td>-187</td>
<td>100.0</td>
</tr>
<tr>
<td>IBM-4</td>
<td>2.69</td>
<td>-10</td>
<td>100.0</td>
<td>5.86</td>
<td>-12</td>
<td>80.0</td>
</tr>
<tr>
<td>IBM-5</td>
<td>2.56</td>
<td>-15</td>
<td>100.0</td>
<td>4.94</td>
<td>-16</td>
<td>93.3</td>
</tr>
<tr>
<td>AL-55</td>
<td>1.09</td>
<td>50</td>
<td>100.0</td>
<td>0.84</td>
<td>50</td>
<td>100.0</td>
</tr>
<tr>
<td>AL-60</td>
<td>0.45</td>
<td>52</td>
<td>100.0</td>
<td>0.67</td>
<td>52</td>
<td>100.0</td>
</tr>
<tr>
<td>AL-65</td>
<td>0.64</td>
<td>57</td>
<td>100.0</td>
<td>0.93</td>
<td>55</td>
<td>96.5</td>
</tr>
<tr>
<td>AL-70</td>
<td>0.67</td>
<td>62</td>
<td>100.0</td>
<td>0.87</td>
<td>57</td>
<td>91.9</td>
</tr>
<tr>
<td>AL-75</td>
<td>0.28</td>
<td>67</td>
<td>100.0</td>
<td>0.34</td>
<td>67</td>
<td>100.0</td>
</tr>
<tr>
<td>AL-80</td>
<td>0.62</td>
<td>68</td>
<td>100.0</td>
<td>0.68</td>
<td>68</td>
<td>100.0</td>
</tr>
<tr>
<td>AL-85</td>
<td>0.64</td>
<td>70</td>
<td>100.0</td>
<td>0.81</td>
<td>70</td>
<td>100.0</td>
</tr>
<tr>
<td>AL-90</td>
<td>0.62</td>
<td>75</td>
<td>100.0</td>
<td>0.86</td>
<td>72</td>
<td>96.0</td>
</tr>
<tr>
<td>AL-100</td>
<td>0.29</td>
<td>85</td>
<td>100.0</td>
<td>0.31</td>
<td>85</td>
<td>100.0</td>
</tr>
</tbody>
</table>
Table V(b). Comparison of Heuristic Algorithms on Randomly Generated Problems.

<table>
<thead>
<tr>
<th>Problem</th>
<th>HCPA</th>
<th>GAMS/ZOOM</th>
<th>Hillier (1969a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VaxStation II</td>
<td>VaxStation II</td>
<td>IBM-360/67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quality</td>
<td>Ratio</td>
<td>Quality</td>
</tr>
<tr>
<td>I-1</td>
<td>0.524</td>
<td>3.0</td>
<td>2.887</td>
</tr>
<tr>
<td>I-2</td>
<td>0.447</td>
<td>7.2</td>
<td>4.258</td>
</tr>
<tr>
<td>I-5</td>
<td>0.990</td>
<td>9.0</td>
<td>7.447</td>
</tr>
<tr>
<td>I-6</td>
<td>0.018</td>
<td>4.5</td>
<td>0.672</td>
</tr>
<tr>
<td>II-1</td>
<td>0.172</td>
<td>3.3</td>
<td>3.180</td>
</tr>
<tr>
<td>II-2</td>
<td>0.000</td>
<td>2.3</td>
<td>0.955</td>
</tr>
<tr>
<td>II-3</td>
<td>0.000</td>
<td>2.4</td>
<td>2.373</td>
</tr>
<tr>
<td>II-4</td>
<td>0.013</td>
<td>2.7</td>
<td>2.011</td>
</tr>
<tr>
<td>II-5</td>
<td>0.000</td>
<td>2.9</td>
<td>0.502</td>
</tr>
<tr>
<td>II-6</td>
<td>0.000</td>
<td>5.4</td>
<td>0.137</td>
</tr>
<tr>
<td>II-7</td>
<td>0.014</td>
<td>2.4</td>
<td>1.197</td>
</tr>
<tr>
<td>II-8</td>
<td>0.066</td>
<td>2.2</td>
<td>1.640</td>
</tr>
<tr>
<td>II-9</td>
<td>0.036</td>
<td>3.2</td>
<td>0.173</td>
</tr>
<tr>
<td>II-10</td>
<td>0.008</td>
<td>4.4</td>
<td>2.741</td>
</tr>
<tr>
<td>II-11</td>
<td>0.012</td>
<td>4.7</td>
<td></td>
</tr>
<tr>
<td>II-12</td>
<td>0.079</td>
<td>2.9</td>
<td></td>
</tr>
<tr>
<td>II-13</td>
<td>0.000</td>
<td>4.4</td>
<td></td>
</tr>
<tr>
<td>II-14</td>
<td>0.000</td>
<td>9.7</td>
<td></td>
</tr>
<tr>
<td>II-M</td>
<td>0.111</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>III-2</td>
<td>0.000</td>
<td>2.0</td>
<td>0.000</td>
</tr>
<tr>
<td>III-3</td>
<td>0.000</td>
<td>2.7</td>
<td>0.064</td>
</tr>
<tr>
<td>III-4</td>
<td>0.000</td>
<td>4.0</td>
<td>0.170</td>
</tr>
<tr>
<td>III-5</td>
<td>0.000</td>
<td>1.8</td>
<td>0.129</td>
</tr>
<tr>
<td>III-8</td>
<td>0.000</td>
<td>1.9</td>
<td>0.187</td>
</tr>
</tbody>
</table>

Quality = $D(x^*, x_H) \equiv (c^T x^* - c^T x_H)/||c||_2$.
Ratio = (Total CPU time)/(CPU time solving $LP_R$).
6. Computational Experience

Algorithm, perhaps by a factor of two or three. Ibaraki, Ohashi and Mine (1974) report achieving good solutions with their interior-path heuristic methods on six of the test problems, although probably at a much greater computational cost than that required by the Heuristic Ceiling Point Algorithm, judging by the ratios of total time to LP solution time.

Average statistics by problem class are shown in Table V(c) for the algorithm of Kochenberger, McCarl and Wyman (1974), as well as those for the Heuristic Ceiling Point Algorithm, the Pivot & Complement heuristic scheme of GAMS/ZOOM and the (1-2A-1) procedure of Hillier. For all three classes of realistic test problems, the Heuristic Ceiling Point Algorithm typically finds higher-quality solutions than does the algorithm of Kochenberger, McCarl and Wyman, but possibly at greater computational effort. Unfortunately, Kochenberger, McCarl and Wyman did not specify the type of computer used in their study. The Heuristic Ceiling Point Algorithm also appears to be more effective than the Pivot & Complement procedure employed by GAMS/ZOOM for all classes of problems (realistic and randomly generated) based on both speed and the quality of solution achieved. In fact, the Pivot & Complement heuristic scheme appears to be competitive with the other algorithms only on the (0-1) AL and Type III problems. Since its performance on the first ten Type II problems was not very strong, no attempt was made to apply GAMS/ZOOM to the five larger Type II problems \{II-1, ..., II-14, II-M\}. On the randomly generated test problems as a whole, Hillier's heuristic procedure seems to be the most effective algorithm judging by both the ratios of total time to time spent solving \((LP_R)\) and solution quality.

7. Summary

In this report, we have described a heuristic algorithm which searches for high-quality feasible 1-ceiling points in the neighborhood of \(\tilde{z}\). In contrast to the heuristic algorithms of Hillier (1969a), Ibaraki, Ohashi and Mine (1974) and Faaland and Hillier (1979), all of which search along paths strictly interior to the feasible region of \((LP_R)\), our search pro-
7. Summary

Table V(c). Summary of Performances by Heuristic Algorithms.

<table>
<thead>
<tr>
<th>Class</th>
<th>HCPA</th>
<th>GAMS/ZOOM</th>
<th>Kochenberger, et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VaxStation II</td>
<td>VaxStation II</td>
<td>(1974)</td>
</tr>
<tr>
<td>FC</td>
<td>0.30</td>
<td>98.4</td>
<td>2.02</td>
</tr>
<tr>
<td>IBM</td>
<td>1.23</td>
<td>97.5</td>
<td>3.13</td>
</tr>
<tr>
<td>AL</td>
<td>0.59</td>
<td>100.0</td>
<td>0.68</td>
</tr>
</tbody>
</table>

ceeds by moving away from \( \bar{x} \) along the surface of the feasible region, periodically rounding from a continuous solution to a nearby integer solution. Once a feasible integer solution is found, one or two components of this solution are modified (sometimes repeatedly) in an attempt to find a feasible 1-ceiling point better than this solution.

In our computational experience, the Heuristic Ceiling Point Algorithm was generally quite successful in finding high-valued solutions. For 16 of the 22 realistic test problems taken from the literature which did not possess an all-integer \( \bar{x} \), our algorithm located an optimal solution (without verifying its optimality), usually in about the same amount of time as that required to solve the LP-relaxation. For all but two of the 20 Type II and Type III randomly generated test problems, an optimal or very high quality solution was found. However, a really good solution was identified for only one of the four Type I problems. Averaged over the classes of randomly generated test problems, the ratios of total CPU time to time spent solving the LP-relaxation ranged from 2.5 to 5.9. On the realistic test problems, the Heuristic Ceiling Point Algorithm typically found better solutions than both the 0-1 Pivot & Complement procedure employed by GAMS/ZOOM and the general integer algorithm of Kochenberger, McCarl and Wyman (1974) and certainly did so more quickly than GAMS/ZOOM. On the randomly generated test problems, the Heuristic Ceil-
7. Summary

ing Point Algorithm again dominated the performance of Pivot & Complement; however, considering both average speed and solution quality, it was outperformed by the general integer method of Hillier (1969a). Overall, we feel that the Heuristic Ceiling Point Algorithm does hold potential as a practical approach for approximately solving pure, general integer linear programming problems. A subsequent report will show how several aspects of this algorithm can be incorporated into an exact algorithm for solving (ILP).


Lee, J. S. and M. Guignard, "An Approximate Algorithm for Multidimensional Zero-One
402-410.

Lustig, I., "Comparisons of Composite Simplex Algorithms," Technical Report SOL 87-8,

Markowitz, H., and A. Manne, "On the Solution of Discrete Programming Problems,"


Saltzman, R., and F. Hillier, "The Role of Ceiling Points in General Integer Linear Pro-
University, Stanford, Calif., August 1988.


Trauth, C., and R. Woolsey, "Integer Linear Programming: A Study in Computational
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.

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

* 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 $z_H = c^T x_H$. 

Listing of Fortran Code Implementation of the Heuristic 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),XO(nn),signac(mm,nn),
- dirs(nn,nn),CTXPT(nn,mm),BFCX0(mm),PRICES(mm)
common/clpl/A,B,C,INDCT,INDOBJ,M,N,
- ABAR,ABAL,ibasis,nonbas,indbas,NPIV,XBAR,XO,20,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,nn+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,CORDER,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
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(nn,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,iomn,iohrun,iocut,iomrun,iolp
character*64 infile,iomn,iohrun,iocut,iomrun,iolp
41
By Robert M. Saltzman

Applies the Heuristic Ceiling Point Algorithm to each problem listed in the file ILPDATA.DAT.

Parenthetical comments with Section numbers refer to parts of:

include 'SDISK2: CSALTZ.ILP1]plist.for'
include '$DISK2: CSALTZ.ILP1]comio.for'
include 'SDISK2: [SALTZ.ILP1]comprt.for'
include 'SDISK2: (SALTZ.ILP1]comxrun3.for'
open( 2, file='SDISK2: [SALTZ.ILP1]ilpdata.dat', status='old')
open( 5, file='SDISK2: [SALTZ.ILP1]outhrun.dat', status='unknown')
open( 6, file='SDISK2: [SALTZ.ILP1]outrun.dat', status='unknown')
open(22, file='SDISK2: [SALTZ.ILP1]switches.dat', status='old')

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

Headlines for Heuristic Summary report, if desired
if (hprint(19) .eq. 1) then
write(5,*)'Heuristic Ceiling Point Algorithm Summary'
write(5,*)'Problem LP Z0 Set+Ph.1 Z '
write(5,*)'Phase2 Z Phase3 Z Total Ratio'
write(5,*)'------ ---- ----- ------ ------- - '
endif

Loop through all problems specified in ILPDATA.DAT
do 8100 ip = 1, 40
Get the name of the input data file, e.g., HALD15.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

call RUN

c write (*,*) '***** Finished problem ***** ', infile
8100 continue
8105 format(All)
8150 continue
stop
end
subroutine RUN

Runs Heuristic Ceiling Point Algorithms for 1 problem
(Overview of entire algorithm given in Section 5.5)

include '$DISK2:SALTZ.ILP1plist.for'
include '$DISK2:SALTZ.ILP1comio.for'
include '$DISK2:SALTZ.ILP1comlp1.for'
include '$DISK2:SALTZ.ILP1comprt.for'
include '$DISK2:SALTZ.ILP1comrun1.for'
include '$DISK2:SALTZ.ILP1comxrun4.for'
integer izz(5)

open(4, file='$DISK2:SALTZ.ILP1outlp.dat', status='unknown')
open(8, file='$DISK2:SALTZ.ILP1outrun.dat', status='unknown')
open(9, file='$DISK2:SALTZ.ILP1outxrun.dat', status='unknown')

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

Solve LP-relaxation of (ILP)
call XTIMER(1,0,zero)
call LPSOLVE
call XTIMER(-1,0,RTIMES(-1))
if (xprint(8) .eq. 0)
   write(*,*),'***> LPSOLVE:','RTIMES(-1),' Z0=',Z0

Check for all-integer LP solution
do 8004 j = 1, n
   if (DABS(XO(j)-IRNDWN(XO(j))) .gt. tol) goto 8008
 8004 continue
write(6,*),'--- > LP solution is all-integer (X*=X0)'
do 8006 j = 1, n
   ixstar(j) = IRNDWN(XO(j))
 8006 continue
goto 8090

Initialize (ALL, X*, ZUP)
8008 call XTIMER(1,0,zero)
ALL = one
ALL = 1 => only search for solns. strictly better than incumbent
ZUP = DBLE(IRNDWN(Z0))
do 8010 j = 1, n
   IXSTAR(j) = -1
 8010 continue
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)

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)

Run Heuristic Ceiling Point Algorithm
call XTIMER(1,0,zero)
call HRUN

if (ixstar(l).lt.0)
  - write(*,*)'*** HRUN failed. Initial Z* =',ZSTAR
call XTIMER(-1,0,rtimes(1))

izz(l) = IRNDWN(ZSTAR)
if (xprint(8).eq.0)
  - write(*,*)'*** HRUN: ',rtimes(1),' Z*=',izz(1)

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),ZO,rtimes(11),rtimes(12),
    rvals(12),rtimes(13),rvals(13),rtimes(14),rpcts(15)
  8015 format(1X,All,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,All,F5.2,6X,F8.2,6X,F8.2,5X,F8.2)
endif

if (DBLE(izz(l)).ge.ZUP)
  - write(6,*)'--- Heuristic alg. found optimal solution'

return
end
subroutine LPSOLVE
C Called by SETUP to solve LP-relaxation of (ILP)
C................................................................................
C  Read in data for this problem: A,B,C,INDCT,INDOBJ
call GETABC
C  Find optimal solution, value: XBAR, X0, and Z0
call Z2PHAS
return
end
C-------------------------------------------------------------------------------
C subroutine GETABC
C Reads in data describing (ILP) and, if desired, reorders vars.
C--------------------------------------------------------------------------------
include '$DISK2:[SALTZ.ILP]plist.for'
include '$DISK2:[SALTZ.ILP]comio.for'
include '$DISK2:[SALTZ.ILP]comprt.for'
include '$DISK2:[SALTZ.ILP]comipl.for'
include '$DISK2:[SALTZ.ILP]comrun1.for'
real*8 ctemp(nn),Atemp(mm,nn)
open(3, file=infile, status='unknown')
C Read in problem data (unformatted)
read (3,*) m,n
do 2 i = 1, m
read (3,*) (A(i,j), j=1,n),B(i),INDCT(i)
2 continue
read(3,*) (C(j),j=1,n),INDOBJ
C Initialize corder and varord
do 4 j = 1, n
  corder(j) = j
  varord(j) = j
4 continue
C Change all problems to maximization form
if (indobj .eq. -1) then
  do 6 j = 1, n
    C(j) = -C(j)
6  continue
  indobj = 1
endif
C Reorder vars. by (1) low c(j) to hi or (2) hi-lo
if (hprint(16) .gt. 0) then
  if (hprint(16) .eq.1) call RSORT1(n,c,corder)
  if (hprint(16) .eq.2) call RSORT2(n,c,corder)
  do 10 j = 1, n
    ctemp(j) = c(corder(j))
    do 8 i = 1, m
      Atemp(i,j) = A(i,corder(j))
7  continue
  do 10 continue
10 continue
do 14 j = 1, n
   c(j) = ctemp(j)
   do 12 i = 1, m
      A(i, j) = Atemp(i, j)
   12 continue
   continue
   14 continue
endif

Data echo (after possible reordering), if desired
if (hprint(l) .eq. 1) then
   write(4,*),'---------','infile
   write(4,*),'GETABC: m=',m,' n=',n
   write(4,*),'GETABC: corder=',(corder(j),j=1,n)
   do 16 i = 1, m
      write(4,*) (A(i, j),j=1,n),B(i),INDCT(i)
   16 continue
   write(4,*),'GETABC: C=',(C(j),j=1,n),INDOBJ
endif
return
end

---

subroutine Z2PHAS

Runs 2-phase simplex method on LP-relaxation of (ILP)

---

include 'SDISK2:[SALTZ.ILPl]plist.for'
include 'SDISK2:[SALTZ.ILPl]comprt.for'
include 'SDISK2:[SALTZ.ILPllcompl.for'
logical ARTR(mm),ARTC(l+nn+2*m)

initbv = columns initially in basis
ibasis(j) = index of j-th basic variable
indbas(j) = true => X(j) is basic; false => nonbasic
izrow = objective function row's position in A matrix
izrow = m+1
nge = number of >= constraints
nge = 0
neq = number of = constraints
neq = 0

Count number of >= and = constraints
do 20 i = 1, m
   if (indct(i) .eq. 0) neq = neq + 1
   if (indct(i) .eq. -1) nge = nge + 1
20 continue

irhs = right hand side column
irhs = n+nge+m+1
ARTR = rows with an Artificial var; ARTC = columns w/ art.var.
do 21 j = 1, irhs
   ARTC(j) = .false.
21 continue

do 22 i = 1, m
   artr(i) = .false.
   if (indct(i) .eq. 1) goto 22
   ARTR(i) = .true.
   ARTC(N+nge+i) = .true.
(Re)initialize ABAR (for multirun case)
do 24 i = 1, izrow 22 continue
do 23 j = 1, irhs
    ABAR(i,j) = zero 23 continue
do 25 j = 1, n
    ABAR(izrow,j) = -C(j)
    XBAR(j) = zero 25 continue
do 26 j = 1, nn+2*mm
    indbas(j) = .false. 26 continue
ncolge = n
do 35 i = 1, m
do 30 j = 1, n
    ABAR(i,j) = A(i,j) 30 continue
if (indct(i) .eq. -1) then
    Put in appropriate column from -Identity
    ncolge = ncolge + 1
    ABAR(i,ncolge) = -one
endif
Append appropriate column of +Identity <-> basis vars.
    ABAR(i,n+nge+i) = one
    INITBV(i) = n + nge + i
    ibasis(i) = INITBV(i)
    indbas(n+nge+i) = .true.
    ABAR(i,irhs) = B(i)
    XBAR(n+i) = zero 35 continue
if ( (neq + nge) .eq. 0) goto 45
do 40 j = 1, irhs
    ABAR(izrow,j) = zero
    if (ARTC(j)) goto 40
do 38 i = 1, m
    if (ARTR(i)) ABAR(izrow,j) = ABAR(izrow,j) - ABAR(i,j)
38 continue
40 continue

----- Run Phase I of the simplex method ----- 45
iphase = 1
call ZSOLVE(izrow, iphase, irhs, nge)
if ( hprint(2) .eq. 1) write(4,*' w =',ABAR(izrow, irhs)
if ( DABS(ABAR(izrow, irhs)) .lt. tol) goto 45
if ( hprint(2) .eq. 1) write(4,*' *** Problem INFEASIBLE ***
XO(1) = -one
goto 69

----- Run Phase II of the simplex method ----- 45
iphase = 2
if ( (nge + neq) .eq. 0) goto 58
Calculate objective row for original problem
sum = zero

do 52 i = 1, m
    if (ibasis(i) .gt. n) goto 52
    sum = sum + C(ibasis(i))*ABAR(i,irhs)
52 continue

ABAR(izrow,irhs) = sum

Prevent artificial vars. from entering basis after Phase I

do 53 j = (n+nge+l),irhs-1
    if (ARTC(j).and.(.not.indbas(j))) ABAR(izrow,j) = zero
53 continue

do 55 j = 1, n+nge
    if (.not. indbas(j)) then
        sum = zero
        do 54 i = 1, m
            if (ibasis(i) .gt. n) goto 54
            sum = sum + C(ibasis(i))*ABAR(i,j)
45 continue
        if (j .le. n) sum = sum - C(j)
        ABAR(izrow,j) = sum
    endif
55 continue

call ZSOLVE(izrow,iphase,irhs,nge)

Save Optimal solution, obj. value, and prices

do 60 i = 1, m
    XBAR(ibasis(i)) = ABAR(i,irhs)
    PRICES(i) = ABAR(izrow,n+i)
60 continue

do 62 j = 1, n
    XO(j) = XBAR(j)
62 continue

ZO = ABAR(izrow,irhs)

ipos = 0

do 64 j = 1, (irhs-1)
    if (.not. indbas(j)) then
        ipos = ipos + 1
        nonbas(ipos) = j
    endif
64 continue

if (hprint(2) .eq. 1) then
    write(4,*) 'LP Opt. Soln. XO =', (XO(j),j=1,n)
    write(4,*) 'LP Opt. Value ZO =', ZO
    write(4,*) 'LP Opt. Prices =', (PRICES(L),L=1,m)
    write(4,*) 'indbas ', (indbas(L),L=1,m)
    write(4,*) 'nonbas ', (nonbas(L),L=1,n+nge)
endif

69 continue

return

end
subroutine ZSOLVE(izrow, iphase, irhs, nge)

Called by Z2PHAS to run simplex method for a single phase.

include '$DISK2:[SALTZ.ILP1]plist.for'
include '$DISK2:[SALTZ.ILP1]comprt.for'
include '$DISK2:[SALTZ.ILP1]comipl.for'
integer izrow, iphase, irhs, nge

npiv(1) = 0
npiv(2) = 0

Pivot until optimality or maxit limit reached

do 70 k = 1, maxit
  if (hprint(3) .eq. 1) write(4,*)
     'ZSOLVE: Basis=', (ibasisMi,i=1,m)
  if (ir .eq. 0) goto 80
  call ZPIVOT(izrow,iphase,irhs,nge)
  npiv(iphase) = npiv(iphase) + 1
70 continue

80 if (hprint(3).eq.1) write(4,*) 'No.Pivots=',npiv(iphase)
return

-----------------------------------------------

subroutine ZSETRS(izrow,iphase,irhs,nge)

Called by ZSOLVE to locate pivot row (LBV) and pivot column (EBV)
Assumes problem has been converted to maximization form.

ir = pivot row = argmin {rhs(i)/a(i,j)}
ir = 0
is = pivot column = argmin(reduced costs in izrow)
is = 0

Find EBV by examining reduced costs of all non-basic vars.
cmin = bigm
do 100 j = 1, (irhs-1)
   if (indbas(j)) goto 100
   if (ABAR(izrow,j).ge. cmin) goto 100
   cmin = ABAR(izrow,j)
100 continue
is = j
continue

Optimality check: Is minimum reduced cost nonnegative?
if (ABAR(izrow,is) .gt. -tol) goto 120

Find leaving basic variable (LBV) from min ratio test
rmin = bigm
do 110 i = 1, m
   if (ABAR(i,is) .lt. tol) goto 110
   ratio = ABAR(i,irhs)/ABAR(i,is)
   if (ratio .lt. rmin) then
      rmin = ratio
      ir = i
   endif
110 continue

if (hprint(4) .eq. 1) write(4,*,'(A,A,A)') 'ZSETRS: (ir,is)=',ir,is
if (rmin .eq. bigm) then
   write(4,*,'(A,A,A,A,A)') '***** LP is unbounded *****'
   write(*,*,'(A,A,A,A,A)') '***** LP is unbounded *****'
   STOP
endif
120 continue
return
end

subroutine ZPIVOT(izrow, iphase, irhs, nge)
Called by ZSOLVE to pivot on ABAR(ir,is)

include 'DISK2:[SALTZ.ILP1]plist.for'
include 'DISK2:[SALTZ.ILP1]compl1.for'
integer izrow, iphase, irhs, nge
real*8 rcol(mm)

Save column (unit vector) corresponding to LBV
LBV/EBV => leaving/entering basic variable
do 140 i = 1, m+1
   rcol(i) = ABAR(i,ibasis(ir))
140 continue

Update basis: ibasis = indexes of basic vars
indbas(j) = true => X(j) in basis; false => nonbasic
indbas(ibasis(ir)) = .false.
indbas(is) = .true.
ibasis(ir) = is

Pivot in row of LBV and in NonBasic (NB) columns only
do 150 j = 1, irhs
   if (indbas(j)) goto 150
   ABAR(ir,j) = abar(ir,j)/abar(ir,is)
150 continue
c Pivot in all rows except the one <--> LBV (NB columns only)
do 170 i = 1, m+1
   if (i .eq. ir) goto 170
   do 160 j = 1, irhs
      if (.not. indbas(j))
         - abar(i,j) = abar(i,j) - abar(i,is)*abar(ir,j)
   160 continue
   continue
170 continue
c
c rcol becomes new column corresponding to EBV
do 180 i = 1, m+1
   abar(i,is) = rcol(i)
180 continue
return
end
c subroutine SETUP

c Finds SIGNAC, DIRS (extreme directions of FR emanating from X0),
c CTXPT mapping, BFCXO, and converts all cts. to <= form.
c Assumes that LP-relaxation has been solved already by LPSOLVE.

C---------------------------------------------------------------------------------------

include 'SDISK2:[SALTZ.IP1]plist.for'
include 'SDISK2:[SALTZ.IP1]comprt.for'
include 'SDISK2:[SALTZ.IP1]comlpl.for'

C SIGNAC(i,j)= 1 => A(i,j) and C(j) agree in sign
do 205 i = 1, m
   do 200 j = 1, n
      signac(i,j) = .false.
      if ((a(i,j) .gt. zero) .and. (c(j) .gt. zero))
      - signac(i,j) = .true.
      if ((a(i,j) .lt. zero) .and. (c(j) .lt. zero))
      - signac(i,j) = .true.
   200 continue
   205 continue

C Create ABAL = B-inverse extended to n-dimensions
call BALAS

C if (hprint(6) .eq. 1) then
   do 206 k = 1, n
      write (5,*)' ABAL(k)=',(abal(j,k),j=1,n)
   206 continue
C endif

C Create DIRS = matrix of normalized extreme directions
207 do 220 k = 1, n
   d2norm = V2NORM(n,ABAL(l,k))
   if (hprint(5) .eq. 1) write(5,*)' d2norm 2- I,d2norm
   do 215 j = 1, n
      DIRS(j,k) = -ABAL(j,k)/d2norm
   215 continue
   if (hprint(5) .eq. 0) goto 220
   write(5,*)' Dirs(k)=',(DIRS(j,k),j=1,n)
220 continue

C Create CTXPT = mapping of ext. rays to constraints, i.e.,
CTXPT(k,i) = true => Extreme ray k lies on constraint hp (i)
do 240 i = 1, m
   aix = zero
   do 235 k = 1, n
      aidk = zero
      aix = aix + A(i,k)*X0(k).
      do 230 j = 1, n
         aidk = aidk + A(i,j)*DIRS(j,k)
      230 continue
      CTXPT(k,i) = .false.
      if (DABS(aidk) .lt. tol) CTXPT(k,i) = .true.
   235 continue
   if (hprint(5).eq.1)write(5,*)' CTXPT',(CTXPT(k,i),k=1,n)

52
BFCX0(i) = .true.  => constraint (i) is binding at X0
BFCX0(i) = .false.
diff = B(i) - aix
if (DABS(diff) .lt. tol) BFCX0(i) = .true.
continue
if (hprint(5) .eq. 1) write(5,'(BFCX0=\',(BFCX0(i),i=1,m))

Force all constraints to be in the form Ax <= b
do 250 i = 1, m
  if (indct(i) .eq. 1.) goto 250
  B(i) = -B(i)
  indct(i) = 1.
  do 245 j = 1, n
    A(i,j) = -A(i,j)
  245 continue
  continue
if (indobj .eq. -1) Z0 = -Z0
return
end

-------------------------------------------------------------------------
c subroutine BALAS
  c Creates ABAL = final tableau in Balas' (dictionary) form
  C See paper by Balas [Ba71].
  c------------------------------------------------------------------------
include '$DISK2: ISALTZ.ILP1plist.for'
include 'SDISK2: [SALTZ.ILP1]compl.for'

  c    ibasis = indexes of basic variables
  c    nonbas = indexes of non-basic variables
  c    indbasj= true => Xj in basis, false => nonbasic
  c    nx = no. of nonbasics (+++ will change when NGE > 0)
  c    nx = n
  c
  c Initialize ABAL matrix to all 0's
  do 306 jr = 1, n
    do 305 jc = 1, n
      ABAL(jr,jc) = zero
    305 continue
  306 continue
  c
  Loop through each column, checking whether basic or not
  do 330 j = 1, n
    if (.not. indbas(j)) goto 317
  c
    X(j) is basic: find position of j in basis
    do 310 k = 1, m
      if (ibasis(k) .eq. j) jpos = k
    310 continue
  c
    Move in column of B-inverse
    do 315 icol = 1, nx
      ABAL(j,icol) = ABAR(jpos,nonbas(icol))
    315 continue
  c
  goto 330
  c
  53
c X(j) is non-basic: find position of j in non-basis
317 do 320 k = 1, n
    if (nonbas(k) .eq. j) jpos = k
320 continue

continue

Put appropriate n-dimensional unit vector in this column
ADAL(j,jpos) = -one

continue

return

end

-----------------------------------------------------------------------------

subroutine BOUNDS

Finds Simple Upper Bounds SUB from <= cts. w/ coefs. all >= 0
Called by PRECUT. These are fairly weak bounds in general.

-----------------------------------------------------------------------------

include '$DISK2:SALTZ.ILP1plist.for'
include '$DISK2:SALTZ.ILP1comprt.for'
include '$DISK2:SALTZ.ILP1compl1.for'
include '$DISK2:SALTZ.ILP1comxrun2.for'

logical okrow(mm)

c okrow(i) = .true. => all A(i,j) >= 0 and (i) is <= ct.
do 410 i = 1, m
    okrow(i) = .false.
    if (B(i) .le. zero) goto 410
    do 405 j = 1, n
        if (A(i,j) .lt. zero) goto 410
    405 continue
    okrow(i) = .true.
410 continue

c SUB(j) = minimum over all okrows of { B(i)/A(i,j) }
do 440 j = 1, n
    ubmin = 100.0D0
    do 430 i = 1, m
        if (okrow(i)) then
            if (A(i,j) .eq. zero) goto 430
            ub = B(i)/A(i,j)
            if (ub .lt. ubmin) ubmin = ub
        endif
    430 continue
    SUB(j) = IRNDWN(ubmin)
440 continue

if (xprint(ll) .eq. 1) write(8,*)' SUB=',(SUB(i),i=1,n)
return

end

54
subroutine HRUN

By Robert M. Saltzman
Called by RUN to run Heuristic Ceiling Point Algorithm

include '$DISK2:[SALTZ.ILP1]pplist.for'
include '$DISK2:[SALTZ.ILP1]comprt.for'
include '$DISK2:[SALTZ.ILP1]compl.for'
include '$DISK2:[SALTZ.ILP1]comhrun.for'
include '$DISK2:[SALTZ.ILP1]comrun1.for'

integer ipordr(mm)
real*8 temp(mm), tmax, valist(nn)

Phase 1

if (hprint(19) .eq. 1) call XTIMER(11,0,zero)

ibest = index of best row -> feasible solution in FIS3 matrix
ibest = 1
ffeas = indicates whether any feasible solution yet found
ffeas = .false.

AIMIN(i) = smallest non-0 coef. in row i of A (See IFFEAS/ncp)
do 1102 i = 1, m
   rmin = DABS(A(i,1))
do 1101 j = 2, n
   aa = DABS(A(i,j))
   if (aa .eq. zero) goto 1101
   if (aa .lt. rmin) rmin = aa
1101 continue
1102 continue

if (hprint(7).eq.1) write(5,*) 'AIMIN-', (AIMIN(i),i=1,n)

PCT01 = percentage of coefs in (-1,0,1). Used in FEASCHK/ncp
PCT01 = zero
do 1110 i = 1, m
   do 1105 j = 1, n
      if (DABS(A(i,j)) .lt. 2.) PCT01 = PCT01 + one
1105 continue
1110 continue
PCT01 = PCT01/(m*n)
if (hprint(7).eq.1) write(5,*,'(PCT01=',PCT01)

Initialize FIS3 = (FIS1, FIS2, FIS3) with row i = (value, FIS)
do 1115 i = 1, m
   FIS3(i,1) = -bigm
   FIS3(i,2) = -one
1115 continue

HROUNDWRT returns result in the first row of FIS3
call HROUND
if (hprint(7).eq.1) write(5,*,'(FIS3=',(fis3(i),iq=1,m))
c RATES(k) = rate of obj. change of k-th extreme direction
C (See Section 4.3.1: "rho(k)"

do 1124 k = 1, n
     RATES(k) = VDOT(n,c,dirs(1,k))
1124 continue
    if (hprint(7).eq.1) write(5,* 'RATES=',(RATES(ik),ik=1,n)

c CTVAL(i) = Sum of rates(k) for all ext. dirs. lying on (i)
do 1128 i = 1, m
    for those cts. not binding at X0, set CTVAL to -inf.
    CTVAL(i) = -bign
    if (.not. BFCX0(i)) goto 1128
    CTVAL(i) = zero
    do 1126 k = 1, n
         if (CTXPT(k,i)) CTVAL(i) = CTVAL(i) + RATES(k)
    1126 continue
1128 continue
    if (hprint(7).eq.1) write(5,* 'CTVAL=',(CTVAL(i),i=1,m)

C.............................. Phase 2 ..............................
C
    if (hprint(19).eq.1) then
       call XTIMER(-11,0,rtimes(11))
       call XTIMER(12,0,zero)
    endif

C nhps = number of constraint hyperplanes to search
C (See "K4" in Section 4.5)
    if (hprint(20).eq.0) then
       nhps = AINT(SQRT(REAL(N)))
    else
       Set nhps = number of constraints binding at X0
       nhps = 0
       do 1130 i = 1, m
            if (BFCX0(i)) nhps = nhps + 1
       1130 continue
    endif

C nis2 = max. number of integer solutions to seek in Phase 2
C nis2 = MIN0(m,nhps)
C nis3 = max. number of integer solutions used to launch Phase 3
C (See "K1" in Section 4.5)
C nis3 = nis2 + 1

C Loop through constraints, searching for 1-ceiling points

do 1140 ip = 2, nis3

C Pick a search constraint (ihp)
    ihp = IHPICK(m)
    if (hprint(7).eq.1) write(5,* 'hp = ',ihp,' nis3 = ',nis3

C Calculate a search direction along this search ct.
call HSDIR(ihp)
    if (hprint(7).eq.1) write(5,* 'Sdir ',(sdir(j),j=1,n)
Use Phase 2 method to locate feasible integer solution(s)
call FINDFS(ip,ihp)
c
if (.not. FFEAS) goto 1140
if (FIS3(ip,1).GT.FIS3(ibest,1)) ibest = ip
if (hprint(7).eq.1) then
  write(5,*)'FFEAS=',FFEAS
  write(5,*)'FIS3=',(fis3(ip,iq),iq-l,n+l)
  write(5,*)'Index of best pt. = ',ibest
endif
c
Exit if found high-valued ceiling point, i.e.,
if (ncp .and. (FIS3(ip,1).ge.FIS3(ibest,1))) then
c  Save information prior to exiting
  if (hprint(19).eq.1) then
      call XTIMER(-12,0,rtimes(12))
      rvals(12) = FIS3(ibest,1)
      call XTIMER(13,0,zero)
  endif
goto 1190
cendif

1140 continue
c
if (FIS3(ibest,1).gt.-bigm) goto 1150
if (hprint(7).eq.1) write(5,*)'FINDFS failed to find FIS'
c
c............................... Phase 3 ............................
c
(See Section 4.4)
cPursue in Phase3 only the NKEEP best FIS's found in Phase2
cFirst column of FIS3 is objective function value of point
c
1150 if (hprint(19).eq.1) then
  call XTIMER(-12,0,rtimes(12))
  rvals(12) = FIS3(ibest,1)
  call XTIMER(13,0,zero)
endif
c
NKEEP = 2
c(See "K5" in Section 4.5)
doi 1152 jj = 1, nis3
temp(jj) = FIS3(jj,1)
1152 continue
doi 1156 ii = 1, NKEEP
tmax = -bigm
CFind the point with the iiith largest objective value
doi 1154 jj = 1, nis3
  if (temp(jj).gt.tmax) then
    tmax = temp(jj)
    ipordr(ii) = jj
  endif
1154 continue
temp(ipordr(ii)) = -bigm
1156 continue
doi (hprint(7).eq.1) write(5,*)'ipordr:',(ipordr(j),j=1,NKEEP)
Phase 3: try to improve upon a feasible integer solution

\begin{verbatim}
do 1170 iip = 1, NKEEP
  ip = ipordr(iip)
  Skip point if it is infeasible
  if (FIS3(ip,2) .lt. zero) goto 1170
  O/W, add to list of obj. values (PRIOR to Phase 3)
  valist(iip) = FIS3(ip,1)
  if (iip .eq. 1) go to 1165
  if (FIS3(ip,1) .eq. valist(iip-1)) goto 1170
  Repeatedly call Phase3 with FIS(ip,) until no improvement
  1165 valold = FIS3(ip,1)
  call PHASE3(ip)
  Phase3 result overwrites previous FIS3(ip,)
  valnew = FIS3(ip,1)
  if (valnew .gt. valold) goto 1165
  if (FIS3(ip,1) .gt. FIS3(ibest,1)) ibest = ip
  continue
  if (hprint(7).eq.1) write(5,*)'Post3 best:',FIS3(ibest,1)
  Return the best solution as (Z*,X*)
  1190 ZSTAR = FIS3(ibest,1)
  do 1195 j = 1, n
    IXSTAR(j) = IRNDWN(FIS3(ibest,1+j))
  1195 continue
  if (hprint(19) .eq. 1) then
    call XTIMER(-13,0,rtimes(13))
    rvals(13) = ZSTAR
  Combine times of SETUP (rtimes(0)) and Phase 1
  rtimes(11) = rtimes(0) + rtimes(11)
  Save LP time
  rtimes(10) = rtimes(-1)
  Get each phase's percentage of total
  rtimes(14) = rtimes(10)+rtimes(11)+rtimes(12)+rtimes(13)
  do 1196 ii = 10, 13
    rpcts(ii) = 100.DO*rtimes(ii)/rtimes(14)
  1196 continue
  Get total/LP ratio
  rpcts(15) = rtimes(14)/rtimes(10)
  Printing of this information done in RUN
endif
if (hprint(7).eq.1) then
  write(5,*)'Heuristic best value  Z*=',ZSTAR
  write(5,*)'Heuristic best solution X*',(IXSTAR(j),j=1,n)
endif
return
\end{verbatim}
C-----------------------------------------------------------------------------------
include '$DISK2:[SALTZ.ILP1]plist.for'
include '$DISK2:[SALTZ.ILP1]comhrun.for'
C
C Select the remaining constraint with the smallest CTVAL(i)
valmax = -bigm
do 1210 i = 1, ncts
   if (CTVAL(i) .gt. valmax) then
      valmax = CTVAL(i)
      imax = i
   endif
1210 continue
C
C Prevent this constraint from being selected in the future
CTVAL(imax) = -bigm
IHPICK = imax
return
end
C-----------------------------------------------------------------------

C subroutine HSDIR(ihp)
C
C Called by HRUN to calculate search direction SDIR.
C Result: SDIR (= "d" in Section 4.3.2).
C-----------------------------------------------------------------------------------
include '$DISK2:[SALTZ.ILP1]plist.for'
include 'SDISK2:[ESALTZ.ILP1]comprt.for'
include '$DISK2:[SALTZ.ILP1]compl.for'
include '$DISK2:[ESALTZ.ILP1]comhrun.for'
INTEGER ihp
C
C Add extreme directions lying on constraint hyperplane (ihp)

   do 1220 j = 1, n
      sdir(j) = zero
      do 1215 k = 1, n
         if (ctxpt(k,ihp)) sdir(j) = sdir(j) + dirs(j,k)
      1215 continue
   1220 continue
C
C Normalize the search direction
   d2norm = V2NORM(n,sdir)
   if (hprint(8).eq.1) write(5,*)'Sdirnorm: ',d2norm
   do 1224 j = 1, n
      sdir(j) = sdir(j)/d2norm
   1224 continue
return
end
subroutine FINDFS(ip,ihp)

c Called by HRUN to find a Feasible Integer Solution

c Result: stored in global variable FIS3(ip,)

c---------------------------------------------------------------------------------

include 'SALTZ.ILP1/plist.for'
include 'SALTZ.ILP1/comprt.for'
include 'SALTZ.ILP1/compl.for'
include 'SALTZ.ILP1/comhrun.for'
integer ip,ihp
real*8 ahp(nn), siprev, sicurr, gap(mm), r(nn), rprev(nn),
- rbest(nn), fs(nn), dx(nn)
if (hprint(9) .eq.1) write(5,*) 'FINDFS: hp,ip=',ihp,ip

ffeas = .false.

nchk = max. number of consecutive infeasibility increases
nchk = "K2" in Section 4.5
nchk = 1+aist(sqrt(real(n)))
ncisi = counter for number of consecutive infeas. increases
ncisi = 0
sicurr= SINF of current solution
sicurr= zero
siprev= SINF of previous solution
siprev= sicurr

SINF = Abbreviation for Sum of Infeasibilities

v = -bigm
vbest = -bigm

do 1240 j = 1, n
   ahp(j) = A(ihp,j)
   r(j) = -one
   fs(j) = X0(j)
   FIS3(ip,j+1) = -one
1240 continue
d

do 1250 i = 1, m
   gap(i) = B(i)
   do 1245 j = 1, n
      gap(i) = gap(i) - A(i,j)*r(j)
1245 continue
1250 continue
d

(Section 4.3.3)

vprev = v
count the number of repetitions of same solution
nreps = 0
call FIS2HP(n,r,ahp,sdir,fs)

Check whether this iteration differs from previous one
do 1260 j = 1, n
   if (r(j) .ne. rprev(j)) goto 1261
1250 continue
nreps = nreps + 1
if (nreps .lt. nchk) goto 1259
if (hprint(9).eq.l)write (5,*) 'EXIT:', nchk, ' same its'
goto 1390

Find DX = change in R from previous iteration RPRESS
1261 do 1262 j = 1, n
   r(j) = DMAX1(zero, r(j))
   dx(j) = r(j) - rprev(j)
1262 continue

Update gaps
do 1264 j = 1, n
   gap(i) = gap(i) - A(i,j)*dx(j)
1264 continue

Determine if the solution is feasible
ifeas = IFFEAS(m,n,gap)
if (ifeas .ne. 1) goto 1300

Feasible solution: calc. obj. value, check vs. vbest
ncisi = 0
siprev= zero
v = VDOT(n,c,r)
if (v .le. vbest) goto 1270
if (hprint(9).eq.1) then
   write(5,*) 'New incumbent value',v
   write(5,*) 'New incumbent soln. ',(r(j), j=1,n)
endif
vbest = v
1268 continue
if (ncp) goto 1390
1270 if (v .ge. vprev) goto 1380
if (hprint(9).eq.1) write(5,*) 'EXIT: V declining'
goto 1390

Solution infeasible: sum the linfeasibilities
sicurr = zero
1300 do 1310 i = 1, m
   if (gap(i) .lt. zero) sicurr = sicurr - gap(i)
1310 continue

61
Check for \texttt{nchk} consecutive increases in \texttt{SINF}
\begin{verbatim}
if (sicurr .gt. siprev) then
  ncisi = ncisi + 1
  siprev = sicurr
  if (ncisi .ge. nchk) goto 1390
else
  ncisi = 0
  siprev = zero
endif
\end{verbatim}

\begin{verbatim}
1380 continue
if (hprint(9).eq.1) write(5,*,'(A,1X,F9.1)') 'FINDFS: max. iter. limit'
1390 if (vbest .eq. -bigm) goto 1399
  FIS3(ip,1) = vbest
  do 1395 j = 1,n
    FIS3(ip,1+j) = rbestr(j)
  1395 continue
1399 continue
if (hprint(9).eq.1) write(5,*,'(A,1X,F9.1)') 'FINDFS: final vbest=',vbest
return
\end{verbatim}

\begin{verbatim}
subroutine FIS2HP (ndim, res, ai, dir, xfs)
C Called by FINDFS to find a Feasible Integer Solution wrt HP
C (See Section 4.3.3) (ndim=N, res=r, ai=ahp, dir=SDIR, xfs=fs)
C................................................................
include '$DISK2:[SALTZ.ILP1]plist.for'
include 'SDISK2:[SALTZ.ILP1]comprt.for'
c integer ndim
real*8 res(nn),ai(nn),dir(nn),xfs(nn),xfrac(nn),alpha(nn)
if (hprint(10).eq.1) write(5,*,'(A,1X,F9.1)') 'FIS2HP: ...........
C Find first component to reach next integer value
C (xfrac = "$f(j)" in Section 4.3.3)
alpmin = bigm
do 1410 j = 1, ndim
  res(j) = zero
  if (dir(j)) 1402,1410,1404
C dir(j) < 0: xfrac = distance to next lower integer
  xfrac(j) = xfs(j) - DINT(xfs(j))
  if (xfrac(j) .eq. zero) xfrac(j) = one
  goto 1408
C dir(j) > 0: xfrac = distance to next higher integer
  xfrac(j) = (one + DINT(xfs(j))) - xfs(j)
1408 alpha(j) = xfrac(j)/DABS(dir(j))
  if (alpha(j) .lt. alpmin) alpmin = alpha(j)
1410 continue
\end{verbatim}
Move to non-int. feas. solution w/ >= 1 integer comp.

(iupdate xfs) and round to integer solution based on A(i,)

do 1420 j = 1, ndim
   xfs(j) = xfs(j) + alpmin*dir(j)
   if (ai(j)) 1412, 1414, 1416
1412 res(j) = one + DINT(xfs(j))
   if (xfs(j) .eq. DINT(xfs(j))) res(j) = xfs(j)
   goto 1420
1414 res(j) = DNINT(xfs(j))
   goto 1420
1416 res(j) = DINT(xfs(j))
1420 continue

if (hprint(10).eq.1) then
   write(5,*) 'fsnew', (xfs(j),j1,ndim)
   write(5,*) 'res', (res(j),j=1,ndim)
endif
return
end

C-----------------------------------------------------------------------------------

integer function IFEFEAS (im, in,gaps)

Called by FINDFS to check the feasibility of a solution

Result: 1 => gaps all non-neg. (feasible)

include '$DISK2:[SALTZ.ILPl1plist.for'
include '$DISK2:[SALTZ.ILP1]comprt.for'
include '$DISK2:[SALTZ.ILP1]comrun1.for'
include '$DISK2:[SALTZ.ILP1]comhrun.for'
integer im, in
real*8 gaps(im)

IFEFEAS = 0
ffeeas = .false.
ncp = .false.
do 1430 i = 1, im
   if (gaps(i) .lt. zero) goto 1440
1430 continue
Feasible point has been found since all gaps are nonnegative
ffeeas = .true.
IFEFEAS = 1
if (in .gt. 8) goto 1435
if (PCT01 .gt. 0.75) goto 1435
Check for n-ceiling point: use sufficient conditon for CP(FR)
if (hprint(21) .eq. 0) then
   do 1432 i = 1, im
      if (gaps(i) .lt. aimin(i)) then
         ncp = .true.
         goto 1435
      endif
1432 continue
endif
end
if (hprint(l1).eq.1) write(5,*) '++Feasible: ncp =', ncp
goto 1450

Infeasible point has been found since a gap is negative
continue
if (hprint(l1).eq.1) write(5,*) '--Infeas:gaps(i) =', gaps(i)

continue
return
end
subroutine PHASE3(ip)

C Called by HRUN to improve upon FIS3(ip,). (See Section 4.4)

include 'SDISK2:[SALTZ.ILP1]plist.for'
include 'SDISK2:[SALTZ.ILP1]compt.for'
include 'SDISK2:[SALTZ.ILP1]compl.for'
include 'SDISK2:[SALTZ.ILP1]comhrun.for'
integer ip
real*8 gaps(mm),IS1(nn),IS2(nn),dk
if (hprint(12).eq.1) write(5,*), 'PHASE3: ip = ',ip
if (hprint(12).eq.1) write(5,*), 'FIS3 = ',(FIS3(ip,L),L=1,n+1)

C Each row of FIS3 contains (value, solution)
do 3000 j = 1, n
   IS1(j) = FIS3(ip,j+1)
   IS2(j) = IS1(j)
3000 continue
V1 = FIS3(ip,1)
V2 = V1

c Find the slack (gaps(i)) of solution wrt each constraint
do 3008 i = 1, m
   gaps(i) = B(i)
   do 3004 j = 1, n
      gaps(i) = gaps(i) - A(i,j)*IS1(j)
3004 continue
3008 continue
j = 0
k = 0
dk = zero

c Try changing 1 component of IS1 to improve upon it
call STAYFS(IS1,gaps,j,k,dk)
c
if (hprint(12).eq.1) write(5,*), 'STAYFS=> k,dk = ',k,dk
if (k.eq.0) goto 3010
IS1(k) = IS1(k) + dk
c
V1 = objective function value of result on STAYFS
V1 = V1 + (dk*C(k))
3010 continue

c Try changing 2 components of IS2 to improve upon it
call TWOVAR(IS2,gaps)
c
V2 = objective function value of result of TWOVAR
V2 = VDOT(n,C,1S2)
c
Replace FIS3(ip,) with the better of IS1 and IS2
if (V1.gt. V2) then
   FIS3(ip,1) = V1
do 3015 j = 1, n
   FIS3(ip,j+1) = IS1(j)
3015 continue
if (hprint(12).eq.1) write(5,*), 'Ph3:1-var best, V=',V1
else
  c
  TWOVAR was better than STAYFS
  3020
  FIS3(ip,1) = V2
  do 3025 j = 1, n
     FIS3(ip,j+1) = IS2(j)
  3025
  continue
     if (hprint(12).eq.1)write(5,*) 'Ph3:2-var best, V=',V2
  endif
  c
  return
end
C--------------------------------------------------------------------------------
C
 subroutine STAYFS (ISOL,gap, ij, ik,deltak)
 C Called by PIIASE3
 C to improve upon ISOL by
 C (See Section 4.4.1)
include '$DISK2: fSALTZ.ILP1]plist.for'
include '$DISK2: (SALTZ.ILP1]comprt.for'
include '$DISK2: (SALTZ.ILP1]compl.for'
include 'SDISK2: (SALTZ.ILPl1comhrun.forI'
include '$DISK2: [SALTZ.ILPl1comrunI.forI
C
integer ij, ik
real*8 ISOL(nn),gap(mm),deltak,d(nn)
if (hprint(13) .eq. 1) then
  write(5,*) 'STAYFS: gap=',(gap(ii-1,m)
endif

vbest = -bigm
ivbest = 1
C
Consider changing every component (iv)
do 3170 iv = 1, n
  d(iv) = zero
  if (iv .eq. ij) goto 3170
Xv is no help if gap small and must decrease to help obj
do 3105 i = 1, m
     if ((gap(i),lt,AIMIN(i)).and.SIGNAC(i,iv))goto 3170
  3105
continue
if (C(iv)) 3110, 3170, 3140
Civ < 0: delv is largest nonpos. change
  3110
d(iv) = -bigm
(delv = "delta(i,j)" in Section 4.4.1)
do 3120 i = 1, m
     if (A(i,iv) .eq. zero) goto 3120
     delv = gap(i)/A(i,iv)
     if (delv .gt. zero) goto 3120
     delv = -DMIN1(ISOL(iv),dabs(delv))
     if (delv .gt. d(iv)) d(iv) = delv
  3120
continue
  d(iv) = DBLE(IRNDUP(d(iv)))
goto 3170
3170
C Civ > 0: delv is smallest nonneg. change
3140 d(iv) = bigm
   do 3150 i = 1, m
      if (A(i,iv) .eq. zero) goto 3150
      delv = gap(i)/A(i,iv)
      if (delv .lt. zero) goto 3150
      if (delv .lt. d(iv)) d(iv) = delv
   3150 continue
   d(iv) = DBLE(IRNDWN(d(iv)))
   continue

3170 Prepare to exit: ik = index of best component to change
if (hprint(13) .eq. 1) write(5,*)'STAYFS:deltas','(d(j),j=1,n)
i = 0
   deltak = zero
   vbest = zero
   do 3180 j = 1, n
      val = C(j)*d(j)
      if (val .gt. vbest) then
         Update the incumbent
         vbest = val
         ik = j
         deltak = d(j)
      endif
   3180 continue
   if (hprint(13) .eq. 1) write(5,*)'STAYFS:k=',ik,', dk=',deltak
return
end

C---------------------------------------------------------------------
C subroutine TWOVAR(ISOL, ogap)
C
C Called by PHASE3 to improve upon ISOL by modifying 2 variables.
C (See Section 4.4.2)
C---------------------------------------------------------------------
include '$DISK2:[SALTZ.ILPljizist.for'
include '$DISK2:LSALTZ.ILP1jcomprt.for'
include '$DISK2:[SALTZ. ILPilcomipi .for'
real*8 ISOL(nn),ogap(mm),tis(nn),IRES(nn),tgap(mm),ud(nn)
if (hprint(14) .eq. 1) write(5,*)'2VAR:ISOL','(isol(L),L=1,n)
C
vbest = zero
C
C Change each first component of ISOL by +1 or -1
   do 3270 j = 1, n
      ud(j) = direction (up(1)/down(-1)) which Xj will go
      if (ISOL(j) .eq. zero) goto 3210
      sum = zero
      do 3205 i = 1, m
         sum = sum + A(i,j)
      3205 continue
      if (sum .gt. zero) ud(j) = -one
      if (sum .lt. zero) ud(j) = one
      if (sum .ne. zero) goto 3210
   3270 continue
do 3220 L = 1, n
   tis(L) = ISOL(L)
continue
   tis(j) = tis(j) + ud(j)
do 3230 i = 1, m
   tgap(i) = ogap(i) - A(i,j)*ud(j)
continue
   k = 0
dk = zero
if (IFFEAS(m,n,tgap) .eq. 0) goto 3240

Changing first component led to feasible solution
call STAYFS(tis,tgap,j,k,dk)
goto 3250

Changing first component led to infeasible solution
continue
call GETFES(tis,tgap,j,k,dk)

if (k .eq. 0) goto 3270

Save result of the changes if improvement
vres = (c(j)*ud(j)) + (c(k)*dk)
if (vres .gt. vbest) then
   vbest = vres
   do 3265 L = 1, n
      ires(L) = ISOL(L)
   continue
   ires(j) = ISOL(j) + ud(j)
   ires(k) = ISOL(k) + dk
endif

continue

Return result in ISOL if it is an improvement
if (vbest .gt. zero) then
   do 3280 L = 1, n
      ISOL(L) = ires(L)
   continue
endif

if (hprint(14).eq.1) write(5,*)'TWOVAR:vbest=',vbest
return
end

----------------------------------------------------------------
subroutine GETFES(ISOL,gap,ij,ik,deltak)
Called by TWOVAR to find FIS by 1-var. change from ISOL(infeas)
(Referred to as "GAINFEAS" in Section 4.4.2)
----------------------------------------------------------------
include '$DISK2:[SALTZ.ILP1]plist.for'
include '$DISK2:[SALTZ.ILP1]comprt.for'
include '$DISK2:[SALTZ.ILP1comlpl.for'
integer ij, ik
real*8 ISOL(nn),gap(mm),deltak
if (hprint(14) .eq. 1) then
    write(5,*) 'GETFES: ij-',ij,' ik-',ik
    write(5,*) 'GETFES: isol-', (isol(L),L1I,n)
    write(5,*) 'GETFES: gap-*, (gap(L),L-l,m)
endif

ik - 0
deltak - zero
vbest -- bigm
0
Allow for C(j)*Del(j) to outweigh C(v)*Delv

check whether or not each component can lead to feas. soln.
do
    iv = 1, n
    if (iv .eq. ij) goto 3370
    do
        i = 1, ms
        if ((gap(i).lt.zero).and.(A(i,iv).eq.zero)) goto 3370
        continue
    c
end do

    iright -IRNDWN(bigm)
    ileft -- iright
    do
        i = 1, m
        if (A(i, iv) .lt. zero) then
            idiv = IRNDUP(gap(i)/A(i,iv))
            if (idiv .gt. ileft) ileft = idiv
        end if
    continue
    c
    Limit decrease (ileft) to avoid ISv + Dv < 0
    ileft = MAXO(ileft, -IRNDWN(ISOL(iv)))
    if (hprint(14) .eq. 1) then
        write(5,*)'GETFES: iv=',iv,' (L,R)=',ileft,iright
    endif
    if (ileft .gt. iright) goto 3370
    d = DBLE(irl)
    c
    if (c(iv) .lt. zero) then
        idiv = IRNDWN(bigm)
        if (idiv .gt. ileft) ileft = idiv
        continue
    continue
    c
    d = DBLE(lire)
    if (c(iv) .gt. zero) then
        idiv = IRNDUP(bigm)
        if (idiv .lt. ileft) ileft = idiv
        continue
    continue
    c
    do
        d = DBLE(idiv)
        if (c(iv) .lt. zero) then
            idiv = IRNDUP(bigm)
            if (idiv .lt. ileft) ileft = idiv
        end if
    continue
    c
    Allow for C(j)*Del(j) to outweigh C(v)*Delv
    do
        iv = 1, n
        if (iv .eq. ij) goto 3370
        do
            i = 1, m
            if (A(i,iv) .lt. zero) then
                idiv = IRNDUP(gap(i)/A(i,iv))
                if (idiv .lt. iright) iright = idiv
            end if
        continue
        c
    end do
    d = DBLE(idiv)
    if (c(iv) .lt. zero) then
        idiv = IRNDWN(bigm)
        if (idiv .gt. ileft) ileft = idiv
    end if
    continue
    c
    Limit decrease (ileft) to avoid ISv + Dv < 0
    ileft = MAXO(ileft, -IRNDWN(ISOL(iv)))
    if (hprint(14) .eq. 1) then
        write(5,*)'GETFES: iv=',iv,' (L,R)=',ileft,iright
    endif
    if (ileft .ge. iright) goto 3370
    d = DBLE(iright)
    c
    if (c(iv) .gt. zero) then
        idiv = IRNDUP(bigm)
        if (idiv .lt. ileft) ileft = idiv
    end if
    continue
    c
    continue
end if
if (hprint(14) .eq. 1) then
    write(5,*) 'GETFES: ik-',ik,' deltak=',deltak
    write(5,*) 'GETFES: vbest- ',vbest
endif
return
subroutine HROUND

Called by HRUN to round XO wrt binding cts. Store best in FIS3(1,)

include '$DISK2:ESALTZ.ILP1/plist.for'
include '$DISK2:ESALTZ.ILP1/comprt.for'
include '$DISK2:ESALTZ.ILP1/comlpl.for'
include '$DISK2:ESALTZ.ILP1/comhrun.for'
integer ISOL(nn)
real*8 gap(mm)

Loop through all problem constraints
do 3470 i = 1, m
   if (.not. BFCX0(i)) goto 3470

Round XO wrt this binding constraint: ISOL is result.
call HRNDPT(X0,i,ISOL)
if (hprint(15).eq.l)write(5,*)'HRNDPT->',(ISOL(L),L=1,n)

Compute the gap or slack of ISOL wrt each constraint
do 3420 irow = 1, m
   gap(irow) = B(irow)
   do 3410 j = 1, n
      gap(irow) = gap(irow) - A(irow,j)*DBLE(ISOL(j))
   3410 continue
   3420 continue

Determine the feasibility of ISOL using gap vector
ifeas = IFFEAS(m,n,gap)
if (hprint(15).eq.1) then
   write(5,*)'HRNDPT: ifeas ',ifeas
   write(5,*)'HRNDPT: gap ',(gap(L),L=1,m)
endif
if (ifeas .eq. 0) goto 3470

ISOL is feasible: Compute its objective function value
val = zero
do 3430 j = 1, n
   val = val + C(j)*DBLE(ISOL(j))
3430 continue

ii (val .gt. FIS3(1,1)) then
   New best feasible solution
   FIS3(1,1) = val
   do 3450 j = 1, n
      FIS3(1,j+1) = DBLE(ISOL(j))
   3450 continue
endif

3470 continue
if (hprint(15).eq.1)
   write(5,*)'HROUND: FIS3(1,)=',(FIS3(1,L),L=1,n+1)
return
end
subroutine HRNDPT(X, ihp, IRES)

Called by HROUND to round X to the feasible side of constraint ihp.
Returns rounded integer solution in IRES. (See Section 4.3.3)

C-----------------------------------------------
include '$DISK2:[SALTZ.ILP]plist.for'
include '$DISK2:[SALTZ.ILP]compl.for'
integer IRES(nn), ihp
real*8 X(nn)

C    Direction to round X(j) depends on sign of A(ihp,j)
Do 3540 j = 1, n
   if (A(ihp,j)) 3510, 3520, 3530
3510    IRES(j) = IRNDUP(X(j))
goto 3540
3520    IRES(j) = IDNINT(X(j))
goto 3540
3530    IRES(j) = IRNDWN(X(j))
3540 continue
return
end
REAL*8 FUNCTION V2NORM(N,X)

Returns the L2-Norm of an n-vector X.

INTEGER N
REAL*8 X(N),sum

sum = VDOT(N, X, X)
V2NORM = DSQRT(sum)
return
end

REAL*8 FUNCTION V1NORM(N,X)

Returns the L1-Norm of an n-vector X.

integer N,j
REAL*8 X(N),sum

sum = 0.D0
do 9002 j = 1, N
   sum = sum + DABS(X(j))
9002 continue
VINORM = sum
return
end

REAL*8 FUNCTION VDOT(N,X,Y)

Returns the dot product of 2 n-vectors, X & Y.

INTEGER N,j
REAL*8 X(N),Y(N), sum

sum = 0.D0
do 9010 j = 1, n
   sum = sum + x(j)*y(j)
9010 continue
VDOT = sum
return
end
Integer function IRNDUP(x)

Rounds a real number x up to smallest integer \( \geq x \)

#include 'SYSK2:SALTZ.ILP1plist.for'
real*8 x

if (x .lt. 0) go to 9015
For x \( \geq 0 \)
IRNDUP = 1 + IDINT(x)
if ((x-tol) .le. IDINT(x)) IRNDUP = IDINT(x)
goto 9020
For x < 0:
9015 IRNDUP = IDINT(x)
if ((x-tol) .le. (IDINT(x) - 1)) IRNDUP = IDINT(x) - 1
9020 continue
return
end

Integer function IRNDWN(x)

Rounds a real number x down to largest integer \( \leq x \)

#include 'SYSK2:SALTZ.ILP1plist.for'
real*8 x

if (x .lt. 0) go to 9025
For x \( \geq 0 \)
IRNDWN = IDINT(x)
if ((x+tol) .ge. (1+IDINT(x))) IRNDWN = 1+IDINT(x)
goto 9030
For x < 0:
9025 IRNDWN = IDINT(x) - 1
if ((x+tol) .ge. IDINT(x)) IRNDWN = IDINT(x)
9030 continue
return
end
subroutine RSORT1 (in, vec, indexa)

Crude sort of Real vector from low to high. result: indexa

include '$DISK2:[SALTZ.ILP1]plist.for'
integer in, indexa(nn)
real*8 vec(nn), temp(nn)

do 9040 j = 1, in
   temp(j) = vec(j)
9040 continue

do 9050 i = 1, in
   tmin = bigm
   do 9045 j = 1, in
      if (temp(j) .lt. tmin) then
         tmin = temp(j)
         indexa(i) = j
      endif
   9045 continue
   temp(indexa(i)) = bigm
9050 continue
return
end

subroutine RSORT2 (in, vec, indexa)

Crude sort of Real vector from high to low. result: indexa

include '$DISK2:[SALTZ.ILP1]plist.for'
integer in, indexa(nn)
real*8 vec(nn), temp(nn)

do 9060 j = 1, in
   temp(j) = vec(j)
9060 continue

do 9070 i = 1, in
   tmax = -bigm
   do 9065 j = 1, in
      if (temp(j) .gt. tmax) then
         tmax = temp(j)
         indexa(i) = j
      endif
   9065 continue
   temp(indexa(i)) = -bigm
9070 continue
return
end
SUBROUTINE XTIMER(CLOCK, PRTOPT, SECONS)
IMPLICIT REAL*8 (A-H,O-Z)
PARAMETER (NUMTIM = 30)
INTEGER CLOCK, PRTOPT

C This routine turns on or off a selected clock and optionally prints
C statistics regarding all clocks or just the clock chosen.
C
C The procedure for adding a new timer is as follows:
C 1) Change PARAMETER statement at beginning of this subroutine
C 2) Change computed GOTO in TIMOUT subroutine
C 3) Add WRITE statement in TIMOUT subroutine and GOTO 500 statement
C 4) Add FORMAT statement in TIMOUT subroutine
C
C Value of ABS(CLOCK) is which clock to use. If CLOCK is > 0, then the
C clock is reset to start timing at the current time (determined by
C calling the machine dependent subroutine NOWCPU. If CLOCK is < 0, then
C the clock is turned off & the statistic is recorded for the amount of
C time since the clock was turned on.
C CLOCK = 0 resets all clocks and statistics if PRTOPT = 0
C PRTOPT = 0 indicates print nothing
C = 1 indicates print last statistic for this clock,
C only if CLOCK < 0
C = 2 indicates print all statistics for all clocks
C SECONS = CPU Time in seconds for clock number ICLOCK
C
C Currently, the only statistic kept is the mean
C The information is placed in COMMON so it stays around each time the
C subroutine is called. Note that the size of the common is determined
C by the parameter at the beginning of the routine.
C
COMMON /MITIME/ LASTIM(NUMTIM), ISUMTI(NUMTIM), NUMSTA(NUMTIM)
C
INTEGER ICLOCK, ITIME, ISTAT, ILO, IHI
CHARACTER*4 LMEAN, LAST
DATA LMEAN/'MEAN'/, LAST/'LAST'/

ICLOCK = IABS(CLOCK)
IF (ICLOCK .GT. 0) THEN
  ITIME = NOWCPU(0)
  IF (CLOCK .GT. 0) THEN
    LASTIM(ICLOCK) = ITIME
  ELSE
    ISTAT = ITIME - LASTIM(ICLOCK)
    SECONS = ISTAT/100.DO
    ISUMTI(ICLOCK) = ISUMTI(ICLOCK) + ISTAT
    NUMSTA(ICLOCK) = NUMSTA(ICLOCK) + 1
  END IF
ELSE IF (PRTOPT .EQ. 0) THEN
  ITIME = NOWCPU(1)
  ITIME = NOWCPU(0)
  DO 100 I=1, NUMTIM
    LASTIM(I) = ITIME
    ISUMTI(I) = 0
    NUMSTA(I) = 0
  100 CONTINUE
END IF
C Now deal with print options
C
GO TO (200, 300, 400) PRTOPT+1
C Print option 0 and default is do nothing
200 GO TO 500
C Print option 1 is to print statistic for last clock if just turned off
300 IF (CLOCK .LT. 0) THEN
   CALL TIMOUT(LAST, ISTAT, ICLOCK, 1)
END IF
GO TO 500
C Print option 2 is to print all statistics if CLOCK = 0, or print
C statistic for individual clock
400 IF (CLOCK .NE. 0) THEN
   ILO = ICLOCK
   IHI = ICLOCK
ELSE
   ITIME = NOWCPU(-1)
   ILO = 1
   IHI = NUMTIM
END IF
DO 450 I = ILO, IHI
   ISTAT = NUMSTA(I)
   IF (ISTAT .GT. 0) ISTAT = (10*ISUMTI(I))/ISTAT
   CALL TIMOUT(LMEAN, ISTAT, I, 10)
450 CONTINUE
500 RETURN
END
SUBROUTINE TIMOUT(LSTA, ISTAT, ICLOCK, IDIV)
C
C Since Fortran can’t compute the Format statement while executing, we
C need a computed GOTO to write the correct statistic. As statistics
C are added, the computed goto must be modified. The numbering of the
C statements should be obvious
C
CHARACTER*4 LSTA
INTEGER ISTAT, ICLOCK, IDIV
C
C LSTA is 4 characters to print out to tell which type of statistic it is
C ISTAT is the statistic to print out
C ICLOCK selects the correct FORMAT statement
C IDIV is the amount to divide ISTAT by before printing
C
COMMON /MTIMFI/ ITIMFI
DOUBLE PRECISION RESULT
IF (ITIMFI .EQ. 0) GOTO 500
RESULT = DBLE(ISTAT)/DBLE(IDIV)
GOTO (110, 140) ICLOCK
100 GOTO 500
110 WRITE(ITIMFI, 1010) ICLOCK, LSTA, RESULT
   GOTO 500
140 WRITE(ITIMFI, 1040) ICLOCK, LSTA, RESULT
   GOTO 500
500 RETURN
C What follows are the Format statements for each timer. Timer 1 uses C line 1010, Timer 2 uses 1040, etc. They should all start with a place C for an A4 and have a place for an I10 C

1010 FORMAT(Ix,'Clock ',12,iX,A4,' time for entire program ',T50,F15.2, * centiseconds')

1040 FORMAT(Ix,'Clock ',12,iX,A4,' time for inner loop ',T50, * F15.2,' centiseconds')
END
SWITCHES.DAT

0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0
0 0 0 0 0 0 0 1 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

1st row is hprint:
1. getabc (data echo)
2. z2phase
3. zsolve
4. zsetrs, zpivot4
5. setup
6. balas
7. hrun
8. hdir
9. findfs
10. fis2hp
11. iffeas
12. phase3
13. stayfs
14. twovar, getfes
15. hround, hrndpt
16. VarOrder: 1)lo-hi; 2)hi-lo
17. REDCTS: list active cts.
18. REORDR: fixed vars. first
19. HRUN: time/print Phases
20. HRUN: 1 => nhps = No.BFCX0
21. HRUN: 1 => avoid NCP check

2nd row is xprint:
1. Runcut
2. Precut
3. Objcut, Redcts
4. Cutpt, Cutpt2
5. Cuthp
6. Cutsdr
7. Aratio
8. l=>MINIMAL SCREEN PRINTING
9. Ishr
10. Ishrs, Isrend, Mincmp
11. Bounds
12. Xrun
13. Xcp
14. Xcb
15. Ichkbd
16. Ilastv, Incmod
17. Ixpick
18. 1 => STOP AFTER HRUN
19. 1 => STOP AFTER ICUT
20. only use cutpt2
21. Skip HRUN, start RUNCUT w/ Z*=0

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

78
Input Format

\[
m \ n \ a_{11} \ldots a_{1n} \ b_1 \ (1,-1) \ (1 \Rightarrow \leq \text{constraint}; \ -1 \Rightarrow \geq) \\
\vdots \ \vdots \ \vdots \ \vdots \\
\vdots \ \vdots \ \vdots \ \vdots \\
a_{m1} \ldots a_{mn} \ b_m \ (1,-1) \\
c_1 \ldots c_n \ (1,-1) \ (1 \Rightarrow \text{maximize}; \ -1 \Rightarrow \text{min.})
\]

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 8 0 1
18 4 4 18 28 1 6 4 2 9 7 1 100 1
-12 0 0 0 0 0 0 1 0 0 0 0 0 1
-15 0 0 0 0 0 0 1 0 0 0 0 0 1
0 0 -12 0 0 0 0 0 0 1 0 0 0 0 1
0 0 0 -10 0 0 0 0 0 0 1 0 0 0 1
0 0 0 0 -11 0 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
**A Heuristic Ceiling Point Algorithm for General Integer Linear Programming**

- **AUTHOR(S):** Robert M. Saltzman and Frederick S. Hillier

- **PERFORMING ORGANIZATION NAME AND ADDRESS:**
  Department of Operations Research - SOL
  Stanford University
  Stanford, CA 94305-4022

- **CONTROLLING OFFICE NAME AND ADDRESS:**
  Office of Naval Research - Dept. of the Navy
  800 N. Quincy Street
  Arlington, VA 22217

- **REPORT DATE:**
  November 1988

- **NUMBER OF PAGES:**
  79 Pages

- **DISTRIBUTION STATEMENT:**
  This document has been approved for public release and sale; its distribution is unlimited.

- **KEY WORDS:**
  integer linear programming; general integer variables, heuristic algorithm; ceiling points; linear programming relaxation; enumeration algorithms

- **ABSTRACT:**
  (Please see other side)
Abstract

A Heuristic Ceiling Point Algorithm
for General Integer Linear Programming

Robert M. Saltzman and Frederick S. Hillier
Stanford University, 1988

This report describes a heuristic algorithm for the pure, general integer linear programming problem (ILP). In attempting to quickly obtain a near-optimal solution (without concern for establishing optimality), the algorithm searches for a "feasible 1-ceiling point." 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 heuristic approach is based upon the idea that a feasible 1-ceiling point found relatively near the optimal solution for the LP-relaxation is apt to have a high (possibly even optimal) objective function value. Having applied this Heuristic Ceiling Point Algorithm to 48 test problems taken from the literature, it appears that searching for such 1-ceiling points usually does provide a very good solution with a moderate amount of computational effort.