Inequalities and Target Objectives for Metaheuristic Search – Part I: Mixed Binary Optimization

Fred Glover

University of Colorado, Boulder, CO 80309-0419, USA Fred.Glover@Colorado.EDU

Summary. Recent adaptive memory and evolutionary metaheuristics for mixed integer programming have included proposals for introducing inequalities and target objectives to guide the search. These guidance approaches are useful in intensification and diversification strategies related to fixing subsets of variables at particular values, and in strategies that use linear programming to generate trial solutions whose variables are induced to receive integer values. We show how to improve such approaches by new inequalities that dominate those previously proposed and by associated target objectives that underlie the creation of both inequalities and trial solutions.

We also propose supplementary linear programming models that exploit the new inequalities for intensification and diversification, and introduce additional inequalities from sets of elite solutions that enlarge the scope of these models. Part I (the present paper) focuses on 0-1 mixed integer programming, and Part II covers the extension to more general mixed integer programming problems. Our methods can also be used for problems that lack convenient mixed integer programming formulations, by generating associated linear programs that encode part of the solution space in mixed binary or general integer variables

Key words: Zero-one Mixed Integer Programming, Adaptive Search, Valid Inequalities, Parametric Tabu Search

1 Notation and Problem Formulation

We represent the mixed integer programming problem in the form

$$\begin{aligned} \text{(MIP) Minimize } & x_0 = fx + gy \\ \text{subject to } & \\ & (x, y) \in Z = \{(x, y) : Ax + Dy \geq b\} \\ & x \text{ integer} \end{aligned}$$

We assume that $Ax + Dy \geq b$ includes the inequalities $U_j \geq x_j \geq 0$, $j \in N = \{1, \ldots, n\}$, where some components of U_j may be infinite. The

linear programming relaxation of (MIP) that results by dropping the integer requirement on x is denoted by (LP). We further assume $Ax+Dy \geq b$ includes an objective function constraint $x_0 \leq U_0$, where the bound U_0 is manipulated as part of a search strategy for solving (MIP), subject to maintaining $U_0 < x_0^*$, where x_0^* is the x_0 value for the currently best known solution x^* to (MIP).

The current paper focuses on the zero-one version of (MIP) denoted by (MIP:0-1), in which $U_j = 1$ for all $j \in N$. We refer to the LP relaxation of (MIP:0-1) likewise as (LP), since the identity of (LP) will be clear from the context.

Several recent papers have appeared that evidence a sudden rekindling of interest in metaheuristic methods for pure and mixed integer programming problems, and especially to problems in zero-one variables. The issue of identifying feasible integer solutions is addressed in Fischetti, Glover and Lodi [4] and Patel and Chinneck [18], and the challenge of solving Boolean optimization problems, which embrace a broad range of classical zero-one problems, is addressed in Davoine, Hammer and Vizvári [3], and Hvattum, Løkketangen and Glover [15]. Metaheuristics for general zero-one problems are examined in Pedroso [19] and in Nediak and Eckstein [16]. The current paper focuses on metaheuristic approaches from a perspective that complements (and contrasts with) the one introduced in Glover [9].

In the following we make reference to two types of search strategies: those that fix subsets of variables to particular values within approaches for exploiting strongly determined and consistent variables, and those that make use of solution targeting procedures. As developed here, the latter solve a linear programming problem LP(x', c')[1] that includes the constraints of (LP) (and additional bounding constraints in the general (MIP) case) while replacing the objective function x_0 by a linear function $v_0 = c'x$. The vector x' is called a *target solution*, and the vector c' consists of integer coefficients c'_j that seek to induce assignments $x_j = x'_j$ for different variables with varying degrees of emphasis.

We adopt the convention that each instance of LP(x', c') implicitly includes the (LP) objective of minimizing the function $x_0 = fx + gy$ as a secondary objective, dominated by the objective of minimizing $v_0 = c'x$, so that the true objective function consists of minimizing $\omega_0 = Mv_0 + x_0$, where M is a large positive number. As an alternative to working with ω_0 in the form specified, it can be advantageous to solve LP(x', c') in two stages. The first stage minimizes $v_0 = c'x$ to yield an optimal solution $x = x''$ (with objective function value $v''_0 = c'x''$), and the second stage enforces $v_0 = v''_0$ to solve the residual problem of minimizing $x_0 = fx + gy$.[2]

[1] The vector c' depends on x'. As will be seen, we define several different linear programs that are treated as described here in reference to the problem LP(x', c').

[2] An effective way to enforce $v_0 = v''_0$ is to fix all non-basic variables having non-zero reduced costs to compel these variables to receive their optimal first stage values throughout the second stage. This can be implemented by masking the columns for these variables in the optimal first stage basis, and then to continue

A second convention involves an interpretation of the problem constraints. Selected instances of inequalities generated by approaches of the following sections will be understood to be included among the constraints $Ax+Dy \geq b$ of (LP). In our definition of LP(x', c') and other linear programs related to (LP), we take the liberty of representing the currently updated form of the constraints $Ax + Dy \geq b$ by the compact representation $x \in X = \{x : (x, y) \in Z\}$, recognizing that this involves a slight distortion in view of the fact that we implicitly minimize a function of y as well as x in these linear programs.[3]

To launch our investigation of the problem (MIP:0-1) we first review previous ideas for generating guiding inequalities for this problem in Section 2 and associated target objective strategies Section 3. We then present new inequalities in Section 4 that improve on those previously proposed. Section 5 describes models that can take advantage of these new inequalities to achieve intensification and diversification of the search process. The fundamental issue of creating the target objectives that can be used to generate the new inequalities and that lead to trial solutions for (MIP: 0-1) is addressed in Section 6. Section 7 shows how to generate additional inequalities by "mining" reference sets of elite solutions to extract characteristics these solutions exhibit in common. Supplemental strategic considerations are identified in Section 8 and concluding remarks are given in Section 9.

2 Inequalities and Sub-Optimization for Guiding Intensification and Diversification Phases for (MIP:0-1)

Let x' denote an arbitrary binary solution, and define the two associated index sets $N'(0) = \{j \in N : x'_j = 0\}$ and $N'(1) = \{j \in N : x'_j = 1\}$. Then it is evident that the inequality

$$\sum_{j \in N'(0)} x_j + \sum_{j \in N'(1)} (1 - x_j) \geq 1 \tag{1}$$

or equivalently

$$\sum_{j \in N'(0)} x_j - \sum_{j \in N'(1)} x_j \geq 1 - |N'(1)| \tag{2}$$

---

the second stage from this starting basis while ignoring the masked variables and their columns. (The masked non-basic variables may incorporate components of both x and y, and will generally include slack variables for some of the inequalities embodied in $Ax + Dy \geq b$.) The resulting residual problem for the second stage can be significantly smaller than the first stage problem, allowing the problem for the second stage to be solved very efficiently.

[3] In some problem settings, the inclusion of the secondary objective x_0 in $v_{00} = Mv_0 + x_0$ is unimportant, and in these cases our notation is accurate in referring to the explicit minimization of $v_0 = c'x$.

eliminates the assignment $x = x'$ as a feasible solution, but admits all other binary x vectors. The inequality (2) has been used, for example, to produce 0-1 "short hot starts" for branch and bound by Spielberg and Guignard [22] and Guignard and Spielberg [13].

Remark 1. Let x denote an arbitrary binary solution, and define the norm L1 of x as

$$||x|| = ex, \text{ where } e = (1, \ldots, 1)$$

Note that the Hamming distance from the binary vectors x and x' can be expressed as

$$d(x, x') = ||x - x'|| = (e - x')x + (e - x)x'$$

Hence the constraint (2) can be written in the following form:

$$d(x, x') = (e - x')x + (e - x)x' \geq 1.$$

Remark 2. The constraint (2) is called *canonical cut* on the unit hypercube by Balas and Jeroslow [1]. The constraint (2) has been used also by Soyster, Lev and Slivka [21], Hanafi and Wilbaut [14] and Wilbaut and Hanafi [24].

To simplify the notation, we find it convenient to give (2) an alternative representation. Let e' denote the vector given by

$$e'_j = 1 - 2x'_j, \; j \in N$$

or equivalently

$$e' = 1 - 2x',$$

hence

$$e'_j = 1 \text{ if } x'_j = 0 \text{ and } e'_j = -1 \text{ if } x'_j = 1.$$

Then, letting $n'(1) = |N'(1)|$, we can also write (2) in the form

$$e'x \geq 1 - n'(1) \tag{3}$$

More generally, for any positive integer e'_0 satisfying $n \geq e'_0 \geq 1$, the binary vectors x that lie at least a Hamming distance e'_0 from x' are precisely those that satisfy the inequality

$$e'x \geq e'_0 - n'(1) \tag{4}$$

The inequality (4) has been introduced within the context of adaptive memory search strategies (Glover [6] to compel new solutions x to be separated from a given solution x' by a desired distance. In particular, upon identifying a reference set $R = \{x^r, r \in R\}$, which consists of elite and diverse solutions generated during prior search, the approach consists of launching a diversification strategy that requires new solutions x to satisfy the associated set of inequalities

$$e^r x \geq e^r_0 - n^r(1), \quad r \in R. \tag{5}$$

This system also gives a mechanism for implementing a proposal of Shylo [20][4] to separate new binary solutions by a minimum specified Hamming distance from a set of solutions previously encountered.

The inequalities of (5) constitute a form of *model embedded memory* for adaptive memory search methods where they are introduced for two purposes: (a) to generate new starting solutions and (b) to restrict a search process to visiting solutions that remain at specified distances from previous solutions. A diversification phase that employs the strategy (b) operates by eventually reducing the e_0^r values to 1, in order to transition from diversification to intensification. One approach for doing this is to use tabu penalties to discourage moves that lead to solutions violating (5). We discuss another approach in the next section.

A more limiting variant of (5) arises in the context of exploiting strongly determined and consistent variables, and in associated adaptive memory *projection* strategies that iteratively select various subsets of variable to hold fixed at specific values, or to be constrained to lie within specific bounds (Glover [6]). This variant occurs by identifying sub-vectors $x^{r_1}, x^{r_2}, \ldots,$ of the solutions x^r (thus giving rise to associated sub-vectors $e^{r_1}, e^{r_2}, \ldots,$ of e^r) to produce the inequalities

$$e^{r_h} \geq e_0^{r_h} - n^{r_h}(1), \quad r \in R, \;\; h = 1, 2, \ldots \tag{6}$$

The inequalities of (6) are evidently more restrictive than those of (5), if the values $e_0^{r_h}$ are chosen to have the same size as the values e_0^r (i.e., if $e_0^{r_h} \geq e_0^r$ for each r and h).

The inequalities (6) find application within two main contexts. The first occurs within a diversification segment of alternating intensification and diversification phases, where each intensification phase holds certain variables fixed and the ensuing diversification divides each x^r into two sub-vectors x^{r_1} and x^{r_2} that respectively contain the components of x^r held fixed and the components permitted to be free during the preceding intensification phase.

The second area of application occurs in conjunction with frequency memory by choosing three sub-vectors x^{r_1}, x^{r_2} and x^{r_3} (for example) to consist of components of solution x^r that have received particular values with high, middle and low frequencies, relative to a specified set of previously visited solutions. (The same frequency vector, and hence the same way of sub-dividing the x^r vectors, may be relevant for all x^r solutions generated during a given phase of search.)[5] Our following ideas can be implemented to enhance these adaptive memory projection strategies as well as the other strategies previously described.

[4] See also Pardalos and Shylo [17] and Ursulenko [23].

[5] The formulas of Glover [6] apply more generally to arbitrary integer solution vectors.

3 Exploiting Inequalities in Target Solution Strategies

We begin by returning to the simple inequality (3) given by

$$e'x \geq 1 - n'(1)$$

and show how to exploit it in a somewhat different manner. The resulting framework also makes it possible to exploit the inequalities of (5) and (6) more effectively.

We make use of solutions such as x' by assigning them the role of *target solutions*. In this approach, instead of imposing the inequality (3) we adopt the strategy of first seeing how close we can get to satisfying $x = x'$ by solving the LP problem[6]

$$LP(x') : \min_{x \in X} u_0 = e'x$$

where as earlier, $X = \{x : (x, y) \in Z\}$. We call x' the target solution for this problem. Let x'' denote an optimal solution to LP(x'), and let u_0'' denote the corresponding value of u_0, i.e., $u_0'' = e'x''$. If the target solution x' is feasible for LP(x') then it is also uniquely optimal for LP(x') and hence $x'' = x'$, yielding $u_0'' = -n'(1)$. In such a case, upon testing x' for feasibility in (MIP:0-1) we can impose the inequality (3) as indicated earlier in order to avoid examining the solution again. However, in the case where x' is not feasible for LP(x'), an optimal solution x'' will yield $u_0'' > -n'(1)$ and we may impose the valid inequality[7]

$$e'x \geq \lceil u_0'' \rceil \tag{7}$$

The fact that $u_0'' > -n'(1)$ discloses that (7) is at least as strong as (3). In addition, if the solution x'' is a binary vector that differs from x', we can also test x'' for feasibility in (MIP:0-1) and then redefine $x' = x''$, to additionally append the constraint (3) for this new x'. Consequently, regardless of whether x'' is binary, we eliminate x'' from the collection of feasible solutions as well as obtaining an inequality (7) that dominates the original inequality (3).

Upon generating the inequality (7) (and an associated new form of (3) if x'' is binary), we continue to follow the policy of incorporating newly generated inequalities among the constraints defining X, and hence those defining Z of (MIP:0-1). Consequently, we assure that X excludes both the original x' and the solution x''. This allows the problem LP(x') to be re-solved, either for x' as initially defined or for a new target vector (which can also be x'' if the latter is binary), to obtain another solution x'' and a new (7).

It is worthwhile to use simple forms of tabu search memory based on recency and frequency in such processes to decide when to drop previously introduced inequalities, in order to prevent the collection of constraints from

[6] This strategy is utilized in the parametric branch and bound approach of Glover [5] and in the feasibility pump approach of Fischetti, Glover and Lodi [4].

[7] For any real number z, $\lceil z \rceil$ and $\lfloor z \rfloor$ respectively identify the least integer $\geq z$ and the greatest integer $\leq z$.

becoming unduly large. Such approaches can be organized in a natural fashion to encourage the removal of older constraints and to discourage the removal of constraints that more recently or frequently have been binding in the solutions to the LP(x') problems produced (see, e.g., Glover and Laguna [11]). Older constraints can also be replaced by one or several surrogate constraints.

The strategy for generating a succession of target vectors x' plays a critical role in exploiting such a process. The feasibility pump approach of Fischetti, Glover and Lodi [4] applies a randomized variant of nearest neighbor rounding to each non-binary solution x'' to generate the next x', but does not make use of associated inequalities such as (3) and (7). In subsequent sections we show how to identify more effective inequalities and associated target objectives to help drive such processes.

3.1 Generalization to Include Partial Vectors and More General Target Objectives

We extend the preceding ideas in two ways, drawing on ideas of parametric branch and bound and parametric tabu search (Glover [5, 7]). First we consider *partial x vectors* that may not have all components x_j determined, in the sense of being fixed by assignment or by the imposition of bounds. Such vectors are relevant in approaches where some variables are compelled or induced to receive particular values, while others remain free or are subject to imposed bounds that are not binding.

Relative to a given vector x' that may contain both assigned and unassigned (free) components, define $N'(0) = \{j \in N : x'_j = 0\}$, $N'(1) = \{j \in N : x'_j = 1\}$ and $N'(\Phi) = \{j \in N : x'_j = \Phi\}$, where $x'_j = \Phi$ signifies that x'_j is not assigned a value (i.e., is not subject to a binding constraint or target affecting its value). Accompanying the vector x' we introduce an associated target objective $c'x$ where c' is an integer vector satisfying the condition

$$\begin{array}{l} c'_j > 0 \ \text{ if } \ j \in N'(0), \\ c'_j < 0 \ \text{ if } \ j \in N'(1), \\ c'_j = 0 \ \text{ if } \ j \in N'(\Phi). \end{array}$$

The vector e', given by $e'_j = 1$ for $j \in N'(0)$ and $e'_j = -1$ for $j \in N'(1)$, evidently constitutes a special case. We couple the target solution x' with the associated vector c' to yield the problem

$$\text{LP}(x', c') : \min_{x \in X} v_0 = c'x.$$

An optimal solution to LP(x', c'), as a generalization of LP(x'), will likewise be denoted by x'', and we denote the corresponding optimum v_0 value by v''_0 ($= c'x''$). Finally, we define $c'_0 = \lceil v''_0 \rceil$ to obtain the inequality

$$c'x \geq c'_0. \tag{8}$$

By an analysis similar to the derivation of (7), we observe that (8) is a valid inequality, i.e., it is satisfied by all binary vectors that are feasible for (MIP:0-1) (and more specifically by all such vectors that are feasible for LP(x', c')), with the exception of those ruled out by previous examination. We address the crucial issue of how to generate the target objectives and associated target solutions x' to produce such inequalities that aid in guiding the search after first showing how to strengthen the inequalities of (8).

4 Stronger Inequalities and Additional Valid Inequalities from Basic Feasible LP Solutions

Our approach to generate inequalities that dominate those of (8) is also able to produce additional valid inequalities from related basic feasible solution to the LP problem LP(x', c'), expanding the range of solution strategies for exploiting the use of target solutions. We refer specifically to the class of basic feasible solutions that may be called *y-optimal* solutions, which are dual feasible in the continuous variables y (including in y any continuous slack variables that may be added to the formulation), disregarding dual feasibility relative to the x variables. Such y-optimal solutions can be easily generated in the vicinity of an optimal LP solution by pivoting to bring one or more non-basic x variables into the basis, and then applying a restricted version of the primal simplex method that re-optimizes (if necessary) to establish dual feasibility relative only to the continuous variables, ignoring pivots that would bring x variables into the basis. By this means, instead of generating a single valid inequality from a given LP formulation such as LP(x', c'), we can generate a collection of such inequalities from a series of basic feasible y-optimal solutions produced by a series of pivots to visit some number of such solutions in the vicinity of an optimal solution.

As a foundation for these results, we assume x'' (or more precisely, (x'', y'')) has been obtained as a y-optimal basic feasible solution to LP(x', c') by the bounded variable simplex method (see, e.g., Dantzig [2]). By reference to the linear programming basis that produces x'', which we will call the x'' basis, define $B = \{j \in N : x_j \text{ is basic}\}$ and $NB = \{j \in N : x_j \text{ is non-basic}\}$. We subdivide NB to identify the two subsets $NB(0) = \{j \in NB : x''_j = 0\}$, $NB(1) = \{j \in NB : x''_j = 1\}$. These sets have no necessary relation to the sets $N'(0)$ and $N'(1)$, though in the case where x'' is an optimal basic solution[8] to LP(x', c'), we would normally expect from the definition of c' in relation to the target vector x' that there would be some overlap between $NB(0)$ and $N'(0)$ and similarly between $NB(1)$ and $N'(1)$.

The new inequality that dominates (8) results by taking account of the reduced costs derived from the x'' basis. Letting rc_j denote the reduced cost

[8] We continue to apply the convention of referring to just the x-component x'' of a solution (x'', y''), understanding the y component to be implicit.

for the variable x_j, the rc_j values for the basic variables satisfy

$$rc_j \text{ for } j \in B$$

and the rc_j values for the non-basic variables assure optimality for x'' under the condition that they satisfy

$$\begin{aligned} rc_j &\geq 0 \text{ for } j \in NB(0) \\ rc_j &\leq 0 \text{ for } j \in NB(1). \end{aligned}$$

Associated with $NB(0)$ and $NB(1)$, define

$$\begin{aligned} \Delta_j(0) &= \lfloor rc_j \rfloor \quad \text{for } j \in NB(0) \\ \Delta_j(0) &= \lfloor -rc_j \rfloor \quad \text{for } j \in NB(1). \end{aligned}$$

Finally, to identify the coefficients of the new inequality, define the vector d' and the scalar d'_0 by

$$\begin{aligned} d'_j &= c'_j && \text{if } j \in B \\ d'_j &= c'_j - \Delta_j(1) && \text{if } j \in NB(1) \\ d'_j &= c'_j + \Delta_j(1) && \text{if } j \in NB(1) \\ d'_0 &= c'_0 + \sum_{j \in NB(1)} \Delta_j(1). \end{aligned}$$

We then express the inequality as

$$d' \geq d'_0 \tag{9}$$

We first show that (9) is valid when generated from an arbitrary y-optimal basic feasible solution, and then demonstrate in addition that it dominates (8) in the case where (8) is a valid inequality (i.e., where (8) is derived from an optimal basic feasible solution). By our previously stated convention, it is understand that X (and (MIP:0-1)) may be modified by incorporating previously generated inequalities that exclude some binary solutions originally admitted as feasible.

Our results concerning (9) are based on identifying properties of basic solutions in reference to the problem

$$LP(x', d') : \min_{x \in X} z_0 = d'x$$

Proposition 1. *The inequality (9) derived from an arbitrary y-optimal basic feasible solution x'' for LP(x', c') is satisfied by all binary vectors $x \in X$, and excludes the solution $x = x''$ when v''_0 is fractional.*

Proof. We first show that the basic solution x'' for LP(x', c') is an optimal solution to LP(x', d'). Let rd_j denote the reduced cost for x_j when the objective function $z_0 = d'x$ for LP(x', d') is priced out relative to the x'' basis, thus yielding $rd_j = 0$ for $j \in B$. From the definitions of the coefficients d'_j,

and in particular from $d'_j = c'_j$ for $j \in B$, it follows that the relation between the reduced costs rd'_j and rc'_j for the non-basic variables is the same as that between the coefficients d'_j and c'_j; i.e.,

$$\begin{aligned} rd'_j &= rc'_j - \Delta_j(0) \ \text{ if } j \in NB(0) \\ rd'_j &= rc'_j + \Delta_j(1) \ \text{ if } j \in NB(1) \end{aligned}$$

The definitions $\Delta_j(0) = \lfloor rc_j \rfloor$ for $j \in NB(0)$ and $\Delta_j(1) = \lfloor -rc_j \rfloor$ for $j \in NB(1)$ thus imply

$$\begin{aligned} rd_j &\geq 0 \ \text{ if } j \in NB(0) \\ rd_j &\leq 0 \ \text{ if } j \in NB(1). \end{aligned}$$

This establishes the optimality of x'' for $\mathrm{LP}(x', d')$. Since the d'_j coefficients are all integers, we therefore obtain the valid inequality

$$d'x \geq \lceil z''_0 \rceil.$$

The definition of d' yields

$$d'x'' = c'x'' + \sum_{j \in NB(1)} \Delta_j(1)$$

and hence

$$z''_0 = v''_0 + \sum_{j \in NB(1)} \Delta_j(1).$$

Since the $\Delta_j(1)$ values are integers, z''_0 is fractional if and only if v''_0 is fractional, and we also have

$$\lceil z''_0 \rceil = \lceil v''_0 \rceil + \sum_{j \in NB(1)} \Delta_j(1).$$

The proposition then follows from the definitions of c'_0 and d'_0. □

Proposition 1 has the following novel consequence.

Corollary 1. *The inequality (9) is independent of the c'_j values for the non-basic x variables. In particular, for any y-feasible basic solution and specified values c'_j for $j \in B$, the coefficients d'_0 and d'_j of d' are identical for every choice of the integer coefficients $c'_j, j \in NB$.*

Proof. The Corollary follows from the fact that any change in the value of c'_j for a non-basic variable x_j (which must be an integer change) produces an identical change in the value of the reduced cost rc_j and hence also in the values $\Delta_j(0)$ and $-\Delta_j(1)$. The argument of the Proof of Proposition 1 thus shows that these changes cancel out, to produce the same final d'_0 and d' after implementing the changes that existed before the changes. □

In effect, since Corollary 1 applies to the situation where $c'_j = 0$ for $j \in NB$, it also allows each d'_j coefficient for $j \in NB$ to be identified by reference to the quantity that results by multiplying the vector of optimal dual values by the corresponding column A_j of the matrix A defining the constraints of (MIP), excluding rows of A corresponding to the inequalities $1 \geq x_j \geq 0$. (We continue to assume this matrix is enlarged by reference to additional inequalities such as (8) or (9) that may currently be included in defining $x \in X$.)

Now we establish the result that (9) is at least as strong as (8).

Proposition 2. *If the basic solution x'' for LP(x', c') is optimal, and thus yields a valid inequality (8), then the inequality (9) dominates (8).*

Proof. We use the fact that x'' is optimal for LP(x', d') as established by Proposition 1. When x'' is also optimal for LP(x', c'), i.e., the x'' is dual feasible for the x variables as well as being y-optimal, the reduced costs rc_j satisfy $rc_j \geq 0$ for $j \in NB(0)$ and $rc_j \leq 0$ for $j \in NB(1)$. The definitions of $\Delta_j(0)$ and $\Delta_j(1)$ thereby imply that these two quantities are both non-negative. From the definitions of d'_j and d'_0 we can write the inequality $d'x \geq d'_0$ as

$$\sum_{j \in B} c'_j x_j + \sum_{j \in NB(0)} (c'_j - \Delta_j(0)) x_j + \sum_{j \in NB(1)} (c'_j + \Delta_j(1)) x_j \geq c'_0 + \sum_{j \in NB(1)} \Delta_j(1) \tag{10}$$

From $\Delta_j(0), \Delta_j(1) \geq 0$, and from $1 \geq x_j \geq 0$, we obtain the inequalities $\Delta_j(0) x_j \geq 0$ and $-\Delta_j(1) x_j \geq -\Delta_j(1)$. Hence

$$\sum_{j \in NB(0)} \Delta_j(0) x_j + \sum_{j \in NB(1)} -\Delta_j(1) x_j \geq \sum_{j \in NB(1)} -\Delta_j(1) \tag{11}$$

Adding the left and right sides of (11) to the corresponding sides of (10) and clearing terms gives

$$\sum_{j \in N} c'_j x_j \geq c'_0$$

Consequently, this establishes that (9) implies (8). □

As in the use of the inequality (7), if a basic solution x'' that generates (9) is a binary vector that differs from x', then we can also test x'' for feasibility in (MIP:0-1) and then redefine $x' = x''$, to additionally append the constraint (3) for this new x'.

The combined arguments of the proofs of Propositions 1 and 2 lead to a still stronger conclusion. Consider a linear program LP(x', h') given by

$$\text{LP}(x', h') : \min_{x \in X} h_0 = h',$$

where the coefficients $h'_j = d'_j$ (and hence $= c'_j$) for $j \in B$ and, as before, B is defined relative to a given y-optimal basic feasible solution x''. Subject

to this condition, the only restriction on the h'_j coefficients for $j \in NB$ is that they be integers. Then we can state the following result.

Corollary 2. *The x'' basis is an optimal LP basis for LP(x', h') if and only if*

$$h'_j \geq d'_j \text{ for } j \in NB(0)$$
$$h'_j \leq d'_j \text{ for } j \in NB(1)$$

and the inequality (9) dominates the corresponding inequality derived by reference to LP(x', h').

Proof. Immediate from the proofs on Propositions 1 and 2. □

The importance of Corollary 2 is the demonstration that (9) is the strongest possible valid inequality from those that can be generated by reference to a given y-optimal basic solution x'' and an objective function that shares the same coefficients for the basic variables.

It is to be noted that if (MIP:0-1) contains an integer valued slack variable s_i upon converting the associated inequality $A_i x + D_i y \geq b_i$ of the system $Ax + Dy \geq b$ into an equation – hence if A_i and b_i consist only of integers and D_i is the 0 vector – then s_i may be treated as one of the components of the vector x in deriving (9), and this inclusion serves to sharpen the resulting inequality. In the special case where all slack variables have this form, i.e., where (MIP:0-1) is a pure integer problem having no continuous variables and all data are integers, then it can be shown that the inclusion of the slack variables within x yields an instance of (9) that is equivalent to a fractional Gomory cut, and a stronger inequality can be derived by means of the foundation-penalty cuts of Glover and Sherali [12]. Consequently, the primary relevance of (9) comes from the fact that it applies to mixed integer as well as pure integer problems, and more particularly provides a useful means for enhancing target objective strategies for these problems. As an instance of this, we now examine methods that take advantage of (9) in additional ways by extension of ideas proposed with parametric tabu search.

5 Intensification and Diversification Based on Strategic Inequalities

5.1 An Intensification Procedure

Consider an indexed collection of inequalities of the form of (9) given by

$$d^p x \geq d^p_0, \quad p \in P. \tag{12}$$

We introduce an intensification procedure that makes use of (12) by basing the inequalities indexed by P on a collection of high quality binary target solutions x'. Such solutions can be obtained from past search history or from

approaches for rounding an optimal solution to a linear programming relaxation (LP) of (MIP:0-1), using penalties to account for infeasibility in ranking the quality of such solutions. The solutions x' do not have to be feasible to be used as target solutions or to generate inequalities. In Section 6 we give specific approaches for creating such target solutions and the associated target objectives $c'x$ that serve as a foundation for producing the underlying inequalities.

Our goal from an intensification perspective is to find a new solution that is close to those in the collection of high quality solutions that give rise to (12). We introduce slack variables $s_p, p \in P$, to permit the system (12) to be expressed equivalently as

$$d^p x - s_p = d_0^p, \quad s_p \geq 0, \; p \in P \tag{13}$$

Then, assuming the set X includes reference to the constraints (13), we create an *Intensified LP Relaxation*

$$\min_{x \in X} s_0 = \sum_{p \in P} w_p s_p$$

where the weights w_p for the variables s_p are selected to be positive integers.

An important variation is to seek a solution that minimizes the maximum deviation of x from solutions giving rise to (12). This can be accomplished by introducing the inequalities

$$s_0 \geq d_0^p - d^p x, \quad p \in P. \tag{14}$$

Assuming these inequalities are likewise incorporated into X, [9] the Min(Max) goal is achieved by solving the problem

$$\min_{x \in X} s_0$$

An optimal solution to either of these two indicated objectives can then be used as a starting point for an intensified solution pass, performing all-at-once or successive rounding to replace its fractional components by integers. [10]

5.2 A Diversification Analog

To create a diversification procedure for generating new starting solutions, we seek an objective function to drive the search to lie as far as possible from

[9] The inclusion of (13) and (14) is solely for the purpose of solving the associated linear programs, and these temporarily accessed constraints do not have to be incorporated among those defining Z.

[10] Successive rounding normally updates the LP solution after rounding each variable in order to determine the effects on other variables and thereby take advantage of modified rounding options.

solutions in the region defined by (12). For this purpose we introduce the variables s_p as in (13), but utilize a maximization objective rather than a minimization objective to produce the problem

$$\max_{x \in X} s_0 = \sum_{p \in P} w_p s_p.$$

The weights w_p are once again chosen to be positive.

A principal alternative in this case consists of maximizing the minimum deviation of x from solutions giving rise to (12). For this, we additionally include the inequalities

$$s_0 \leq d_0^p - d^p x, \quad p \in P \tag{15}$$

giving rise to the problem

$$\max_{x \in X} s_0.$$

The variable s_0 introduced in (15) differs from its counterpart in (14). In the case where the degree of diversification provided by this approach is excessive, by driving solutions too far away from solutions expected to be good, control can be exerted through bounding X with other constraints, and in particular by manipulating the bound U_0 identified in Section 1.

6 Generating Target Objectives and Solutions

We now examine the issue of creating the target solution x' and associated target objective $c'x$ that underlies the inequalities of the preceding sections. This is a key determinant of the effectiveness of targeting strategies, since it determines how quickly and effectively such a strategy can lead to new integer feasible solutions.

Our approach consists of two phases for generating the vector c' of the target objective. The first phase is relatively simple and the second phase is more advanced.

6.1 Phase 1 – Exploiting Proximity

The Phase 1 procedure for generating target solutions x' and associated target objectives $c'x$ begins by solving the initial problem (LP), and then solves a succession of problems LP(x', c') by progressively modifying x' and c'. Beginning from the linear programming solution x'' to (LP) (and subsequently to LP(x', c')), the new target solution x' is derived from x'' simply by setting $x'_j = \langle x''_j \rangle, j \in N$, where $\langle v \rangle$ denotes the nearest integer neighbor of v. (The value $\langle .5 \rangle$ can be either 0 or 1, by employing an arbitrary tie-breaking rule.)

Since the resulting vector x' of nearest integer neighbors is unlikely to be feasible for (MIP:0-1), the critical element is to generate the target objective

$c'x$ so that the solutions x'' to successively generated problems LP(x', c') will become progressively closer to satisfying integer feasibility. If one or more integer feasible solutions is obtained during this Phase 1 approach, each such solution qualifies as a new best solution x^*, due to the incorporation of the objective function constraint $x_0 = U_0 < x_0^*$.

The criterion of Phase 1 that selects the target solution x' as a nearest integer neighbor of x'' is evidently myopic. Consequently, the Phase 1 procedure is intended to be executed for only a limited number of iterations. However, the possibility exists that for some problems the target objectives of Phase 1 may quickly lead to new integer solutions without invoking more advanced rules. To accommodate this eventuality, we include the option of allowing Phase 1 to continue its execution as long as it finds progressively improved solutions.

Phase 1 is based on the principle that some variables x_j should be more strongly induced to receive their nearest neighbors target values x'_j than other variables. In the absence of other information, we may tentatively suppose that a variable whose LP solution value x''_j is already an integer or is close to being an integer is more likely to receive that integer value in a feasible integer solution. Consequently, we are motivated to choose a target objective $c'x$ that will more strongly encourage such a variable to receive its associated value x'_j. However, the relevance of being close to an integer value needs to be considered from more than one perspective.

The targeting of $x_j = x'_j$ for variables whose values x''_j already equal or almost equal x'_j does not exert a great deal of influence on the solution of the new LP(x', c'), in the sense that such a targeting does not drive this solution to differ substantially from the solution to the previous LP(x', c'). A more influential targeting occurs by emphasizing the variables x_j whose x''_j values are more "highly fractional", and hence which differ from their integer neighbors x'_j by a greater amount. There are evidently trade-offs to be considered in the pursuit of influence, since a variable whose x''_j value lies close to .5, and hence whose integer target may be more influential, has the deficiency that the likelihood of this integer target being the "right" target is less certain. A compromise targeting criterion is therefore to give greater emphasis to driving x_j to an integer value if x''_j lies "moderately" (but not exceedingly) close to an integer value. Such a criterion affords an improved chance that the targeted value will be appropriate, without abandoning the quest to identify targets that exert a useful degree of influence. Consequently, we select values λ_0 and $\lambda_1 = 1 - \lambda_0$ that lie moderately (but not exceedingly) close to 0 and 1, such as $\lambda_0 = 1/5$ and $\lambda_1 = 4/5$, or $\lambda_0 = 1/4$ and $\lambda_1 = 3/4$, and generate c'_j coefficients that give greater emphasis to driving variables to 0 and 1 whose x''_j values lie close to λ_0 and λ_1.

The following rule creates a target objective $c'x$ based on this compromise criterion, arbitrarily choosing a range of 1 to 21 for the coefficient c'_j. (From the standpoint of solving the problem LP(x', c'), this range is equivalent to

any other range over positive values from v to $21v$, except for the necessity to round the c'_j coefficients to integers.)

Procedure 1 – Phase 1 Rule for Generating c'_j

Choose λ_0 from the range $.1 \leq \lambda_0 \leq .4$, and let $\lambda_1 = 1 - \lambda_0$.
if $x'_j = 0$ (hence $x''_j \leq .5$) **then**
  if $x''_j \leq \lambda_0$ **then**
    $c'_j = 1 + 20x''_j/\lambda_0$
  else if $x''_j > \lambda_0$ **then**
    $c'_j = 1 + 20(.5 - x''_j)/(.5 - \lambda_0)$
  end if
else if $x'_j = 1$ (hence $x''_j \geq .5$) **then**
  if $x''_j \leq \lambda_1$ **then**
    $c'_j = -(1 + 20(x''_j - .5)/(\lambda_1 - .5))$
  else if $x''_j > \lambda_1$ **then**
    $c'_j = -(1 + 20(1 - x''_j)/(1 - \lambda_1))$
  end if
end if

Finally, replace the specified value of c'_j by its nearest integer neighbor $\langle c'_j \rangle$.

The absolute values of c'_j coefficients produced by the preceding rule describe what may be called a *batwing* function – a piecewise linear function resembling the wings of a bat, with shoulders at $x''_j = .5$, wing tips at $x''_j = 0$ and $x''_j = 1$, and the angular joints of the wings at $x''_j = \lambda_0$ and $x''_j = \lambda_1$. Over the x''_j domain from the left wing tip at 0 to the first joint at λ_0, the function ranges from 1 to 21, and then from this joint to the left shoulder at .5 the function ranges from 21 back to 1. Similarly, from right shoulder, also at .5, to the second joint at λ_1, the function ranges from 1 to 21, and then from this joint to the right wing tip at 1 the function ranges likewise from 21 to 1. (The coefficient c'_j takes the negative of these absolute values from the right shoulder to the right wing tip.).

In general, if we let Tip, $Joint$ and $Shoulder$ denote the $|c'_j|$ values to be assigned at these junctures (where typically $Joint > Tip, Shoulder$), then the generic form of a batwing function results by replacing the four successive c'_j values in the preceding method by

$$
\begin{aligned}
c'_j &= Tip + (Joint - Tip)x''_j/\lambda_0, \\
c'_j &= Shoulder + (Joint - Shoulder)(.5 - x''_j)/(.5 - \lambda_0), \\
c'_j &= -(Shoulder + (Joint - Shoulder)(x''_j - .5)/(\lambda_1 - .5)), \\
c'_j &= -(Tip + (Joint - Tip)(1 - x''_j)/(1 - \lambda_1)),
\end{aligned}
$$

The image of such a function more nearly resembles a bat in flight as the value of Tip is increased in relation to the value of $Shoulder$, and more nearly resembles a bat at rest in the opposite case. The function can be turned into a

piecewise convex function that more strongly targets the values λ_0 and λ_1 by raising the absolute value of c'_j to a power $p > 1$ (affixing a negative sign to yield c'_j over the range from the right shoulder to the right wing tip). Such a function (e.g., a quadratic function) more strongly resembles a bat wing than the linear function.[11]

Design of the Phase 1 Procedure

We allow the Phase 1 procedure that incorporates the foregoing rule for generating c'_j the option of choosing a single fixed λ_0 value, or of choosing different values from the specified interval to generate a greater variety of outcomes. A subinterval for λ_0 centered around .2 or .25 is anticipated to lead to the best outcomes, but it can be useful to periodically choose values outside this range for diversification purposes.

We employ a stopping criterion for Phase 1 that limits the total number of iterations or the number of iterations since finding the last feasible integer solution. In each instance where a feasible integer solution is obtained, the method re-solves the problem (LP), which is updated to incorporate both the objective function constraint $x_0 \leq U_0 < x_0^*$ and inequalities such as (9) that are generated in the course of solving various problems LP(x', c'). The instruction "Update the Problem Inequalities" is included within Phase 1 to refer to this process of adding inequalities to LP(x', c') and (LP), and to the associated process of dropping inequalities by criteria indicated in Section 3.

Procedure 2 – Phase 1

1. Solve (LP). (If the solution x'' to the first instance of (LP) is integer feasible, the method stops with an optimal solution for (MIP:0-1).)
2. Apply the Rule for Generating c'_j, to each $j \in N$, to produce a vector c'.
3. Solve LP(x', c'), yielding the solution x''. Update the Problem Inequalities.
4. If x'' is integer feasible: update the best solution $(x^*, y^*) = (x'', y'')$, update $U_0 < x_0^*$, and return to Step 1. Otherwise, return to Step 2.

A preferred variant of Phase 1 does not change all the components of c' each time a new target objective is produced, but changes only a subset consisting of k of these components, for a value k somewhat smaller than n. For example, a reasonable default value for k is given by $k = 5$. Alternatively, the procedure may begin with $k = n$ and gradually reduce k to its default value. Within Phase 2, as subsequently noted, it can be appropriate to reduce k all the way to 1.

[11] Calibration to determine a batwing structure, either piecewise linear or nonlinear, that proves more effective than other alternatives within Phase 1 would provide an interesting study.

This variant of Phase 1 results by the following modification. Let c^0 identify the form of c' produced by the Rule for Generating c'_j, as applied in Step 2 of the Phase 1 Procedure. Re-index the x_j variables so that $|c^0_1| \geq |c^0_2| \geq \ldots \geq |c^0_n|$, and let $N(k) = \{1, \ldots, k\}$, thus identifying the variables $x_j, j \in N(k)$, as those having the k largest $|c^0_j|$ values. Then Phase 1 is amended by setting $c' = 0$ in Step 1 and then setting $c'_j = c^0_j$ for $j \in N(k)$ in Step 2, without modifying the c'_j values for $j \in N - N(k)$. Relevant issues for research involve the determination of whether it is better to better to begin with k restricted or to gradually reduce it throughout the search, or to allow it to oscillate around a preferred value. Different classes of problems will undoubtedly afford different answers to such questions, and may be susceptible to exploitation by different forms of the batwing function (allowing different magnitudes for the *Tip*, *Joint* and *Shoulder*, and possibly allowing the location of the shoulders to be different than the .5 midpoint, with the locations of the joints likewise asymmetric).

6.2 Phase 2 – Exploiting Reaction and Resistance

Phase 2 is based on exploiting the mutually reinforcing notions of *reaction* and *resistance*. The term "reaction" refers to the change in the value of a variable as a result of creating a target objective $c'x$ and solving the resulting problem LP(x', c'). The term "resistance" refers to the degree to which a variable fails to react to a non-zero c'_j coefficient by receiving a fractional value rather than being driven to 0 or 1.

To develop the basic ideas, let NF identify the set of variables that receive fractional values in the solution x'' to the problem LP(x', c'), given by $NF = \{j \in N : 0 < x''_j < 1\}$, and let $N'(0,1)$ identify the set of variables that have been assigned target values x'_j, given by $N'(0,1) = N'(0) \cup N'(1)$ (or equivalently, $N'(0,1) = N' - N'(\Phi)$). Corresponding to the partition of N' into the sets $N'(\Phi)$ and $N'(0,1)$, the set NF of fractional variables is partitioned into the sets $NF(\Phi) = NF \cap N'(\Phi)$ and $NF(0,1) = NF \cap N'(0,1)$.

We identify two different sets of circumstances that are relevant to defining reaction, the first arising where none of the fractional variables x_j is assigned a target x'_j, hence $NF = NF'(\Phi)$, and the second arising in the complementary case where at least one fractional variable is assigned a target, hence $NF(0,1) \neq \emptyset$. We start by examining the meaning of reaction in the somewhat simpler first case.

Reaction When No Fractional Variables Have Targets

Our initial goal is to create a measure of reaction for the situation where $NF = NF'(\Phi)$, i.e., where all of the fractional variables are unassigned (hence, none of these variables have targets). In this context we define reaction to be

measured by the change in the value x''_j of a fractional variable x_j relative to the value x^0_j received by x_j in an optimal solution x^0 to (LP), as given by [12]

$$\Delta_j = x^0_j - x''_j.$$

We observe there is some ambiguity in this Δ_j definition since (LP) changes as a result of introducing new inequalities and updating the value U_0 of the inequality $x_0 \leq U_0$. Consequently, we understand the definition of Δ_j to refer to the solution x^0 obtained by the most recent effort to solve (LP), though this (LP) may be to some extent out of date, since additional inequalities may have been introduced since it was solved. For reasons that will become clear in the context of resistance, we also allow the alternative of designating x^0 to be the solution to the most recent problem LP(x', c') preceding the current one; i.e., the problem solved before creating the latest target vector c'.

The reaction measure Δ_j is used to determine the new target objective by re-indexing the variables $x_j, j \in NF = NF'(\Phi)$, so that the absolute values $|\Delta_j|$ are in descending order, thus yielding $|\Delta_1| \geq |\Delta_2| \geq \ldots$. We then identify the k-element subset $N(k) = \{1, 2, \ldots, k\}$ of NF that references the k largest $|\Delta_j|$ values, where $k = \min(|NF|, k_{max})$. We suggest the parameter k_{max} be chosen at most 5 and gradually decreased to 1 as the method progresses.

The c'_j coefficients are then determined for the variables $x_j, j \in N(k)$, by the following rule. (The constant 20 is the same one used to generate c'_j values in the Phase 1 procedure, and $\langle v \rangle$ again denotes the nearest integer neighbor of v.)

$NF'(\Phi)$ Rule for Generating c'_j and $x'_j, j \in N(k)$ (*for* $N(k) \subset NF = NF'(\Phi)$):

$$\begin{array}{l} \text{If } \Delta_j \geq 0, \text{ set } c'_j = \quad 1 + \langle 20\Delta_j/|\Delta_1| \rangle \text{ and } x'_j = 0 \\ \text{If } \Delta_j \leq 0, \text{ set } c'_j = -1 + \langle 20\Delta_j/|\Delta_1| \rangle \text{ and } x'_j = 1 \end{array}$$

When $\Delta_j = 0$, a tie-breaking rule can be used to determine which of the two options should apply, and in the special case where $\Delta_1 = 0$ (hence all $\Delta_j = 0$), the c'_j assignment is taken to be 1 or -1 for all $j \in N(k)$.

To determine a measure of reaction for the complementary case $NF(0,1) \neq \emptyset$, we first introduce the notion of resistance.

Resistance

A *resisting variable* (or *resistor*) x_j is one that is assigned a target value x'_j but fails to satisfy $x_j = x'_j$ in the solution x'' to LP(x', c'). Accordingly the index set for resisting variables may be represented by $NR = \{j \in N'(0,1) : x''_j \neq x''_j\}$. If x''_j is fractional and $j \in N'(0,1)$ then clearly $j \in NR$ (i.e., $NF(0,1) \subset NR$). Consequently, the situation $NF(0,1) \neq \emptyset$ that was previously identified as complementary to $NF = NF(\Phi)$ corresponds to the presence of at least one fractional resistor.

[12] These Δ_j values are not to be confused with the $\Delta_j(0)$ and $\Delta_j(1)$ of Section 4.

If a resistor x_j is not fractional, i.e., if the value x''_j is the integer $1 - x'_j$, we say that x_j *blatantly resists* its targeted value x'_j. Blatant resistors x_j are automatically removed from NR and placed in the unassigned set $N'(\Phi)$, setting $c' = 0$. (Alternatively, a blatant resistor may be placed in $N'(1 - x'_j)$ by setting $c'_j = -c'_j$ and $x'_j = 1 - x'_j$.) After executing this operation, we are left with $NR = NF(0, 1)$, and hence the condition $NF(0, 1) \neq \emptyset$ (which complements the condition $NF = NF'(\Phi)$) becomes equivalent to $NR \neq \emptyset$.

Let V_j identify the amount by which the LP solution value $x_j = x''_j$ violates the target assignment $x_j = x'_j$; i.e, $V_j = x''_j$ if $x'_j = 0$ (hence if $c'_j > 0$) and $V_j = 1 - x''_j$ if $x'_j = 1$ (hence if $c'_j < 0$). We use the quantity V_j to define a *resistance measure* RM_j for each resisting variable $x_j, j \in NR$, that identifies how strongly x_j resists its targeted value x'_j. Two simple measures are given by $RM_j = V_j$, and $RM_j = |c_j|V_j$.

The resistance measure RM_j is used in two ways: (a) to select specific variables x_j that will receive new x'_j and c'_j values in creating the next target objective; (b) to determine the relative magnitudes of the resulting c'_j values. For this purpose, it is necessary to extend the notion of resistance by making reference to *potentially resisting* variables (or *potential resistors*) $x_j, j \in N'(0, 1) - NR$, i.e., the variables that have been assigned target values x'_j and hence non-zero objective function coefficients c'_j, but which yield $x''_j = x'_j$ in the solution x'' to LP(x', c'). We identify a resistance measure RM^0_j for potential resistors by reference to their reduced cost values rc_j (as identified in Section 4):

$$RM^0_j = -rc_j \text{ for } j \in N'(0) - NR \text{ and } RM^0_j = rc_j \text{ for } j \in N'(1) - NR.$$

We note that this definition implies $RM^0_j \leq 0$ for potentially resisting variables. (Otherwise, x_j would be a non-basic variable yielding $x''_j = 1$ in the case where $j \in N'(0)$, or yielding $x''_j = 0$ in the case where $j \in N'(1)$, thus qualifying as a blatant resistor and hence implying $j \in NR$.) The closer that RM^0_j is to 0, the closer x_j is to qualifying to enter the basis and potentially to escape the influence of the coefficient c'_j that seeks to drive it to the value 0 or 1. Thus larger values of RM^0_j indicate greater potential resistance. Since the resistance measures RM_j are positive for resisting variables x_j, we see that there is an automatic ordering whereby $RM_p > RM^0_q$ for a resisting variable x_p and a potentially resisting variable x_q.

Combining Measures of Resistance and Reaction

The notion of reaction is relevant for variables x_j assigned target values x_j ($j \in N'(0, 1)$) as well as for those not assigned such values ($j \in N'(\Phi)$). In the case of variables having explicit targets (hence that qualify either as resistors or potential resistors) we combine measures of resistance and reaction to determine which of these variables should receive new targets x'_j and new coefficients c'_j.

Let x^0 refer to the solution x'' to the instance of the problem LP(x', c') that was solved immediately before the current instance;[13] hence the difference between x_j^0 and x_j'' identifies the reaction of x_j to the most recent assignment of c_j' values. In particular, we define this reaction for resistors and potential resistors by

$$\delta_j = x_j'' - x_j^0 \text{ if } x_j' = 0 \ (j \in N'(0))$$
$$\delta_j = x_j^0 - x_j'' \text{ if } x_j' = 1 \ (j \in N'(1)).$$

If we use the measure of resistance $RM_j = V_j$, which identifies how far x_j lies from its target value, a positive δ_j implies that the resistance of x_j has decreased as a result of this assignment. Just as the resistance measure RM_j is defined to be either V_j or $V_j|c_j'|$, the corresponding reaction measure $R\delta_j$ can be defined by either $R\delta_j = \delta_j$ or $R\delta_j = \delta_j|c_j'|$. Based on this we define a composite resistance-reaction measure RR_j for resisting variables as a convex combination of RM_j and $R\delta_j$; i.e., for a chosen value of $\lambda \in [0, 1]$:

$$RR_j = \lambda RM_j + (1 - \lambda)R\delta_j, \quad j \in NR.$$

Similarly, for implicitly resisting variables, we define a corresponding composite measure RR_j^0 by

$$RR_j^0 = \lambda RM_j^0 + (1 - \lambda)R\delta_j, \quad j \in N'(0, 1) - NR.$$

In order to make the interpretation of λ more consistent, it is appropriate first to scale the values of RM_j, RM_j^0 and $R\delta_j$. If v_j takes the role of each of these three values in turn, then v_j may be replaced by the scaled value $v_j = v_j/|Mean(v_j)|$ (bypassing the scaling in the situation where $|Mean(v_j)| = 0$).

To give an effective rule for determining RR_j and RR_j^0, a few simple tests can be performed to determine a working value for λ, as by limiting λ to a small number of default values (e.g., the three values 0, 1 and .5, or the five values that include .25 and .75). More advanced methods for handling these issues are described in Section 8, where a linear programming post-optimization process for generating stronger evaluations is given in Section 8.4, and a target analysis approach for calibrating parameters and combining choice rules more effectively is given in Section 8.5.

Including Reference to a Tabu List

A key feature in using both RR_j and RR_j^0 to determine new target objectives is to make use of a simple tabu list T to avoid cycling and insure a useful degree of variation in the process. We specify in the next section a procedure for creating and updating T, which we treat both as an ordered list and as a set. (We sometimes speak of a variable x_j as belonging to T, with the evident interpretation that $j \in T$.) It suffices at present to stipulate that we

[13] This is the "alternative definition" of x^0 indicated earlier.

always refer to non-tabu elements of $N'(0,1)$, and hence we restrict attention to values RR_j and RR_j^0 for which $j \in N'(0,1) - T$. The rules for generating new target objectives make use of these values in the following manner.

Because RR_j and RR_j^0 in general are not assured to be either positive or negative, we treat their ordering for the purpose of generating c'_j coefficients as a rank ordering. We want each RR_j value (for a resistor) to be assigned a higher rank than that assigned to any RR_j^0 value (for a potential resistor). An easy way to do this is to define a value RR_j for each potential resistor given by

$$RR_j = RR_j^0 - RR_1^0 + 1 - \min_{j \in NR} RR_j, \quad j \in N'(0,1) - NR.$$

The set of RR_j values over $j \in N'(0,1)$ then satisfies the desired ordering for both resistors ($j \in NR$) and potential resistors ($j \in N'(0,1) - NR$). (Recall that $NR = NF(0,1)$ by having previously disposed of blatant resistors.)

For the subset $N(k)$ of k non-tabu elements of $N'(0,1)$ (hence of $N'(0,1) - T$) that we seek to generate, the ordering over the subset $NR - T$ thus comes ahead of the ordering over the subset $(N'(0,1) - NR) - T$, This allows both resistors and potential resistors to be included among those elements to be assigned new coefficients c'_j and new target values x'_j, where the new c'_j coefficients for resistors always have larger absolute values than the c'_j coefficients for potential resistors. If the set of non-tabu resistors $NR - T$ already contains at least k elements, then no potential resistors will be assigned new c'_j or x'_j values.

Overview of Phase 2 Procedure

The rule for generating the target objective $c'x$ that lies at the heart of Phase 2 is based on carrying out the following preliminary steps, where the value k_{max} is determined as previously indicated: (a) re-index the variables x_j, $j \in N'(0,1) - T$, so that the values RR_j are in descending order, thus yielding $RR_1 \geq RR_2 \geq \ldots$; (b) identify the subset $N(k) = \{1, 2, \ldots, k\}$ of NR that references the k largest RR_j values, where $k = \min(|N'(0,1) - T|, k_{max})$; (c) create a rank ordering by letting $R_p, p = 1, \ldots, r$ denote the distinct values among the $RR_j, j \in N(k)$, where $R_1 > R_2 > \ldots R_r, (r \geq 1)$.

Then the rule to determine the c'_j and x'_j values for the variables $x_j, j \in N(k)$, is given as follows:

$N'(0,1) - T$ Rule for Generating c'_j and $x'_j, j \in N(k)$ (*for* $NR = NF(0,1) \neq \emptyset$):

If $x'_j = 1$, and $RR_j = R_p$, set $c'_j = \quad \langle 1 + 20(r + 1 - p)/r \rangle$ and re-set $x'_j = 0$
If $x'_j = 0$, and $RR_j = R_p$, set $c'_j = -\langle 1 + 20(r + 1 - p)/r \rangle$ and re-set $x'_j = 1$

We see that this rule assigns c'_j coefficients so that the $|c'_j|$ values are the positive integers $\langle 1 + 20(1/r)\rangle, \langle 1 + 20(2/r)\rangle, \ldots, \langle 1 + 20(r/r)\rangle = 21$.

We are now ready to specify the Phase 2 procedure in overview, which incorporates its main elements except for the creation and updating of the tabu list T.

Procedure 3 - Phase 2 Procedure in Overview

1. Solve (LP). (Stop if the first instance of (LP) yields an integer feasible solution x'' which therefore is optimal for (MIP:0-1).) (If the solution x'' to the first instance of (LP) is integer feasible, the method stops with an optimal solution for (MIP:0-1).)
2. There exists at least one fractional variable ($NF \neq \emptyset$). Remove blatant resistors if any exist, from NR and transfer them to $N'(\Phi)$ (or to $N'(1 - x'_j)$) so $NR = NF(0, 1)$.
   (a) If $NF = NF(\Phi)$ (hence $NR = \emptyset$), apply the $NF(\emptyset)$ Rule for Generating c'_j and $x'_j, j \in N(k)$, to produce the new target objective $c'x$ and associated target vector x'.
   (b) If instead $NR \neq \emptyset$, then apply the $N'(0, 1) - T$ Rule for Generating c'_j and $x'_j, j \in N(k)$, to produce the new target objective $c'x$ and associated target vector x'.
3. Solve LP(x', c'), yielding the solution x''. Update the Problem Inequalities.
4. If x'' is integer feasible: update the best solution $(x^*, y^*) = (x'', y'')$, update $U_0 < x^*_0$, and return to Step 1. Otherwise, return to Step 2.

6.3 Creating and Managing the Tabu List T – Phase 2 Completed

We propose an approach for creating the tabu list T that is relatively simple but offers useful features within the present context. As in a variety of constructions for handling a recency-based tabu memory, we update T by adding a new element j to the first position of the list when a variable x_j becomes tabu (as a result of assigning it a new target value x'_j and coefficient c'_j), and by dropping the "oldest" element that lies in the last position of T when its tabu status expires.

Our present construction employs a rule that may add and drop more than one element from T at the same time. The checking of tabu status is facilitated by using a vector $Tabu(j)$ that is updated by setting $Tabu(j) = true$ when j is added to T and by setting $Tabu(j) = false$ when j is dropped from T. (Tabu status is often monitored by using a vector $TabuEnd(j)$ that identifies the last iteration that element j qualifies as tabu, without bothering to explicitly store the list T, but the current method of creating and removing tabu status makes the indicated handling of T preferable.).

We first describe the method for the case where $k = 1$, i.e., only a single variable x_j is assigned a new target value (and thereby becomes tabu) on

a given iteration. The modification for handling the case $k > 1$ is straightforward, as subsequently indicated. Two parameters T_{min} and T_{max} govern the generation of T, where $T_{max} > T_{min} \geq 1$. For simplicity we suggest the default values $T_{min} = 2$ and $T_{max} = n^6$. (In general, appropriate values are anticipated to result by selecting T_{min} from the interval between 1 and 3 and T_{max} from the interval between n^5 and n^7.)[14]

The target value x'_j and coefficient c'_j do not automatically change when j is dropped from T and x_j becomes non-tabu. Consequently, we employ one other parameter $AssignSpan$ that limits the duration that x_j may be assigned the same x'_j and c'_j values, after which x'_j is released from the restrictions induced by this assignment. To make use of $AssignSpan$, we keep track of when x_j most recently was added to T by setting $TabuAdd(j) = iter$, where $iter$ denotes the current iteration value (in this case, the iteration when the addition occurred). Then, when $TabuAdd(j) + AssignSpan < iter$, x_j is released from the influence of x'_j and c'_j by removing j from the set $N'(0, 1)$ and adding it to the unassigned set $N'(\Phi)$. As long as x_j is actively being assigned new x'_j and c'_j values, $TabuAdd(j)$ is repeatedly being assigned new values of $iter$, and hence the transfer of j to $N'(\Phi)$ is postponed. We suggest a default value for $AssignSpan$ between $1.5 \times T_{max}$ and $3 \times T_{max}$; e.g. $AssignSpan = 2 \times T_{max}$.

To manage the updating of T itself, we maintain an array denoted $TabuRefresh(j)$ that is initialized by setting $TabuRefresh(j) = 0$ for all $j \in N$. Then on any iteration when j is added to $T, TabuRefresh(j)$ is checked to see if $TabuRefresh(j) < iter$ (which automatically holds the first time j is added to T). When the condition is satisfied, a *refreshing operation* is performed, after adding j to the front of T, that consists of two steps: (a) the list T is reduced in size to yield $|T| = T_{min}$ (more precisely, $|T| \leq T_{min}$) by dropping all but T_{min} the first elements of T; (b) $TabuRefresh(j)$ is updated by setting $TabuRefresh(j) = iter + v$, where v is a number randomly chosen from the interval $[AssignSpan, 2 \times AssignSpan]$. These operations assure that future steps of adding this particular element j to T will not again shrink T to contain T_{min} elements until $iter$ reaches a value that exceeds $TabuRefresh(j)$. Barring the occurrence of such a refreshing operation, T is allowed to grow without dropping any of its elements until it reaches a size of T_{max}. Once $|T| = T_{max}$, the oldest j is removed from the end of T each time a new element j is added to the front of T, and hence T is stabilized at the size T_{max} until a new refreshing operation occurs.

This approach for updating T is motivated by the following observation. The first time j is added to T (when $TabuRefresh(j) = 0$) T may acceptably be reduced in size to contain not just T_{min} elements, but in fact to contain only 1 element, and no matter what element is added on the next iteration the composition of $N'(0, 1)$ cannot duplicate any previous composition. Moreover,

[14] The small value of T_{min} accords with an intensification focus, and larger values may be selected for diversification. A procedure that modifies T_{max} dynamically is indicated in Section 8.1.

following such a step, the composition of $N'(0,1)$ will likewise not be duplicated as long as T continues to grow without dropping any elements. Thus, by relying on intervening refreshing operations with $TabuRefresh(j) = 0$ and $T_{min} = 1$, we could conceivably allow T to grow even until reaching a size $T_{max} = n$. (Typically, a considerable number of iterations would pass before reaching such a state.) In general, however, by allowing T to reach a size $T_{max} = n$ the restrictiveness of preventing targets from being reassigned for T_{max} iterations would be too severe. Consequently we employ the two mechanisms to avoid such an overly restrictive state consisting of choosing $T_{max} < n$ and performing a refreshing operation that allows each j to shrink T more than once (whenever *iter* grows to exceed the updated value of $TabuRefresh(j)$) The combination of these two mechanisms provides a flexible tabu list that is self-calibrating in the sense of automatically adjusting its size in response to varying patterns of assigning target values to elements.

The addition of multiple elements to the front of T follows essentially the same design, subject to the restriction of adding only up to T_{min} new indexes $j \in N(k)$ to T on any iteration, should k be greater than T_{min}. We slightly extend the earlier suggestion $T_{min} = 2$ to propose $T_{min} = 3$ for $k_{max} \geq 3$.

One further comment is warranted concerning the composition of T. The organization of the method assures $T \subset N'(0,1)$ and typically a good portion of $N'(0,1)$ lies outside T. If exceptional circumstances result in $T = N'(0,1)$, the method drops the last element of T so that $N'(0,1)$ contains at least one non-tabu element.

Drawing on these observations, the detailed form of Phase 2 that includes instructions for managing the tabu list is specified below, employing the stopping criterion indicated earlier of limiting the computation to a specified maximum number of iterations. (These iterations differ from those counted by *iter*, which is re-set to 0 each time a new solution is found and the method returns to solve the updated (LP).)

The inequalities introduced in Sections 3 and 4 provide a useful component of this method, but the method is organized to operate even in the absence of such inequalities. The intensification and diversification strategies proposed in Section 5 can be incorporated for solving more difficult problems.

The next section gives another way to increase the power of the foregoing procedure when faced with solving harder problems, by providing a class of additional inequalities that are useful in the context of an intensification strategy.

7 Additional Inequalities for Intensification from an Elite Reference Set

We apply a somewhat different process than the type introduced Section 4 to produce new inequalities for the purpose of intensification, based on a strategy of extracting (or "mining") useful inequalities from a reference set R

Procedure 4 – Complete Phase 2 Procedure

0. Choose the values T_{min} and T_{max} and $AssignSpan$.
1. Solve (LP). (Stop if the first instance of (LP) yields an integer feasible solution x'', which therefore is optimal for (MIP:0-1).) Set $TabuRefresh(j) = 0$ for all $j \in N$ and $iter = 0$.
2. There exists at least one fractional variable ($NF \neq \emptyset$). Remove each blatant resistor x_j, if any exists, from NR and transfer it to $N'(\Phi)$ (or to $N'(1 - x'_j)$), yielding $NR = NF(0, 1)$. If j is transferred to $N'(\Phi)$ and $j \in T$, drop j from T. Also, if $T = N'(0, 1)$, then drop the last element from T.
   (a) If $NF = NF(\Phi)$ (hence $NR = \emptyset$), apply the $NF(\Phi)$ Rule for Generating c'_j and $x'_j, j \in N(k)$.
   (b) If instead $NR = \emptyset$, then apply the $N'(0, 1) - T$ Rule for Generating c'_j and $x'_j, j \in N(k)$.
   (c) Set $iter = iter + 1$. Using the indexing that produces $N(k)$ in (a) or (b), add the elements $j = 1, 2, \ldots, \min(T_{min}, k)$ to the front of T (so that $T = (1, 2, \ldots)$ after the addition). If $TabuRefresh(j) < iter$ for any added element j, set $TabuRefresh(j) = iter + v$, for v randomly chosen between $AssignLength$ and $2 \times AssignSpan$ (for each such j) and then reduce T to at most T_{min} elements by dropping all elements in positions $> T_{min}$.
3. Solve LP(x', c'), yielding the solution x''. Update the Problem Inequalities.
4. If x'' is integer feasible: update the best solution $(x^*, y^*) = (x'', y'')$, update $U_0 < x^*_0$, and return to Step 1. Otherwise, return to Step 2.

of elite solutions. The goal in this case is to generate inequalities that reinforce the characteristics of solutions found within the reference set. The resulting inequalities can be exploited in conjunction with inequalities such as (9) and the systems (12)–(15). Such a combined approach gives an enhanced means for achieving the previous intensification and diversification goals.

The basis for this inequality mining procedure may be sketched as follows. Let $Count_j(v)$, for $v \in \{0, 1\}$, denote the number of solutions in R (or more precisely in an updated instance R' of R), such that $x_j = v$. We make use of sets $J(0)$ and $J(1)$ that record the indexes j for the variables x_j that most frequently receive the values 0 and 1, respectively, over the solutions in R. In particular, at each iteration either $J(0)$ or $J(1)$ receives a new index j^* for the variable x_{j^*} that receives either the value 0 or the value 1 in more solutions of R' than any other variable; i.e., x_{j^*} is the variable having the maximum $Count_j(v)$ value over solutions in R.

Associated with x_{j^*}, we let v^* (= 0 or 1) denote the value v that achieves this maximum $Count_j(v)$ value. The identity of j^* and v^* are recorded by adding j^* to $J(v^*)$. Then $J(v^*)$ is removed from future consideration by dropping it from the current N', whereupon R' is updated by removing all of its solutions x that contain the assignment $x_{j^*} = v^*$. The process repeats until no more solutions remain in R'. At this point we have a minimal, though no necessarily minimum, collection of variables such that every solution in R satisfies the inequality (16) indicated in Step 4 below.

Procedure 5 – Inequality Mining Method (for the Elite Reference Set R)

0. Begin with $R' = R, N' = N$ and $J(0) = J(1) = \emptyset$.
1. Identify the variable $x_{j^*}, j^* \in N'$, and the value $v^* = 0$ or 1 such that

$$Count_{j^*}(v^*) = \max_{j \in N', v \in \{0,1\}} Count_j(v)$$

   Add j^* to the set $J(v^*)$.
2. Set $R' = R' - \{x \in R' : x_{j^*} = v^*\}$ and $N' = N' - \{j^*\}$.
3. If $R' = \emptyset$ or $N' = \emptyset$ proceed to Step 4. Otherwise, determine the updated values of $Count_j(v), j \in N', v \in \{0,1\}$ (relative to the current R' and N') and return to Step 1.
4. Complete the process by generating the inequality

$$\sum_{j \in J(1)} x_j + \sum_{j \in J(0)} (1 - x_j) \geq 1 \tag{16}$$

7.1 Generating Multiple Inequalities

The foregoing Inequality Mining Method can be modified to generate multiple inequalities by the following simple design. Let $n(j)$ be the number of times the variable x_j appears in one of the instances of (16). To initialize these values we set $n(j) = 0$ for all $j \in N$ in an initialization step that precedes Step 0. At the conclusion of Step 4, the $n(j)$ values are updated by setting $n(j) = n(j) + 1$ for each $j \in J(0) \cup J(1)$.

In the simplest version of the approach, we stipulate that each instance of (16) must contain at least one x_j such that $n(j) = 0$, thus automatically assuring every instance will be different. Let L denote a limit on the number of inequalities we seek to generate. Then, only two simple modifications of the preceding method are required to generate multiple inequalities.

(A) The method returns to Step 0 after each execution of Step 4., as long as $n(j) = 0$ for at least one $j \in N$, and as long as fewer than L inequalities have been generated;

(B) Each time Step 1 is visited immediately after Step 0 (to select the first variable x_{j^*} for the new inequality), we additionally require $n(j^*) = 0$, and the method terminates once this condition cannot be met when choosing the first x_{j^*} to compose a given instance of (16). Hence on each such "first execution" of Step 1, j^* is selected by the rule

$$Count_{j^*}(v^*) = \max_{j \in N', n(j)=0, v \in \{0,1\}} Count_j(v).$$

As a special case, if there exists a variable x_j such that $x_j = 1$ (respectively, $x_j = 0$) in all solutions $x \in R$, we observe that the foregoing method will generate the inequality $x_j \geq 1$ (respectively, $x_j \leq 0$) for all such variables.

The foregoing approach can be given still greater flexibility by subdividing the value $n(j)$ into two parts, $n(j:0)$ and $n(j:1)$, to identify the number of times x_j appears in (16) for $j \in J(0)$ and for $j \in J(1)$, respectively. In this variant, the values $n(j:0)$ and $n(j:1)$ are initialized and updated in a manner exactly analogous to the initialization and updating of $n(j)$. The restriction of the choice of j^* on Step 1, immediately after executing Step 0, is simply to require $n(j^*:v^*) = 0$, by means of the rule

$$Count_{j^*}(v^*) = \max_{j \in N', n(j:v)=0, v \in \{0,1\}} Count_j(v).$$

For additional control, the $n(j:0)$ and $n(j:1)$ values (or the $n(j)$ values) can also be constrained not to exceed some specified limit in subsequent iterations of Step 1, in order to assure that particular variables do not appear a disproportionate number of times in the inequalities generated.

7.2 Additional Ways for Exploiting R

It is entirely possible that elite solutions can lie in "clumps" in different regions. In such situations, a more effective form of intensification can result by subdividing an elite reference set R into different components by a clustering process, and then treating each of the individual components as a separate reference set.

One indication that R should be subdivided is the case where the Inequality Mining Method passes from Step 3 to Step 4 as a result of the condition $N' = \emptyset$. If this occurs when $R' \neq \emptyset$, the inequality (16) is valid only for the subset of R given by $R - R'$, which suggests that R is larger than it should be (or that too many inequalities have been generated). Clustering is also valuable in the context of using the inequalities of (12) for intensification, by dividing the target solutions x' underlying these inequalities into different clusters.

There is, however, a reverse consideration. If clustering (or some other construction) produces an R that is relatively small, there may be some risk that the inequalities (16) derived from R may be overly restrictive, creating a form of intensification that is too limiting (and hence that has a diminished ability to find other good solutions). To counter this risk, the inequalities of (16) can be expressed in the form of goal programming constraints, which are permitted to be violated upon incurring a penalty.

Finally, to achieve a greater degree of intensification, the Inequality Mining Method can employ more advanced types of memory to generate a larger number inequalities (or even use lexicographic enumeration to generate all inequalities of the indicated form). Additional variation can be achieved by introducing additional binary variables as products of other variables, e.g., representing a product such as $x_1x_2(1 - x_3)$ as an additional binary variable using standard rules (that add additional inequalities to those composing the

system (16)). These and other advanced considerations are addressed in the approach of *satisfiability data mining* (Glover [8]).

8 Supplemental Strategic Considerations

This section identifies a number of supplemental strategic considerations to enhance the performance of the approaches described in preceding sections.

8.1 Dynamic Tabu Condition for T_{max}

The value of T_{max} can be translated into a tabu tenure that varies within a specified range, and that can take a different value each time a variable x_j (i.e., its index j) is added to the tabu list T. This can be done by employing an array $TabuEnd(j)$ initialized at 0 and updated as follows. Whenever j is added to T, a value v is selected randomly from an interval $[T_a, T_b]$ roughly centered around T_{max}. (For example, if $T_{max} = n^6$ the interval might be chosen by setting $T_a = n^5$ and $T_b = n^7$). $TabuEnd(j)$ is assigned the new value $TabuEnd(j) = v + iter$. Subsequently, whenever j is examined to see if it belongs to T (signaled by $Tabu(j) = true$), if $iter > TabuEnd(j)$ then j is dropped from T.

In place of the rule that removes an element from T by selecting the element at the end of the list as the one to be dropped, we instead identify the element to be dropped as the one having the smallest $TabuEnd(j)$ value. When j is thus removed from T, we re-set $TabuEnd(j) = 0$.

The condition $TabuEnd(j) \geq iter$ could be treated as equivalent to $Tabu(j) = true$, and it would be possible to reference the $TabuEnd(j)$ array in place of maintaining the list T, except for the operation that drops all but the T_{min} first elements of T. Because of this operation, we continue to maintain T as an ordered list, and manage it as specified. (Whenever an element is dropped from T, it is dropped as if from a linked list, so that the relative ordering of elements remaining on T is not disturbed.) Alternatively, T can be discarded if $TabuEnd(j)$ is accompanied by an array $TabuStart(j)$, where $TabuStart(j) = iter$ at the iteration where j becomes tabu, thus making it possible to track the longevity of an element on T.

8.2 Using Model Embedded Memory to Aid in Generating New Target Objectives

We may modify the specification of the c'_j values in Phase 2 by using model embedded memory, as proposed in parametric tabu search. For this, we replace the value 20 in the c'_j generation rules of Section 6 by a value $BaseCost$ which is increased on each successive iteration, thus causing the new $|c'_j|$ values to grow as the number of iterations increases. The influence of these values in

driving variables to reach their targets will thus become successively greater, and targets that have been created more recently will be less likely to be violated than those created earlier. (The larger the absolute value of c'_j the more likely it will be that x_j will not resist its target value x'_j by becoming fractional.)

Consequently, as the values $|c'_j|$ grow from one iteration to the next, the variables that were given new targets farther in the past will tend to be the ones that become resistors and candidates to receive new target values. As a result, the c'_j coefficients produced by progressively increasing $BaseCost$ emulate a tabu search recency memory that seeks more strongly to prevent assignments from changing the more recently that they have been made.

The determination of the c'_j values can be accomplished by the same rules specified in Section 6 upon replacing the constant value 20 by $BaseCost$. Starting with $BaseCost = 20$ in Step 1 of Phase 2, the value of $BaseCost$ is updated each time $iter$ is incremented by 1 in Step 3 to give $BaseCost = \lambda \times BaseCost$ where the parameter λ is chosen from the interval $\lambda \in [1.1, 1.3]$. (This value of λ can be made the same for all iterations, or can be selected randomly from such an interval at each iteration.)

To prevent the $|c'_j|$ values from becoming excessively large, the current $|c'_j|$ values can be reduced once $BaseCost$ reaches a specified limit by the applying following rule.

Reset $BaseCost = 20$ and index the variables $x_j, j \in N'(0,1)$ so that
$|c'_1| \geq |c'_2| \geq \ldots \geq |c'_p|$ where $p = |N'(0,1)|$.
Define $\Delta_j = |c'_j| - |c'_{j+1}|$ for $j = 1, \ldots, p-1$.
Select $\lambda \in [1.1, 1.3]$.
Set $|c'_p| = BaseCost$ and $|c'_j| = \min(|c'_{j+1}| + \Delta_j, \lambda|c'_{j+1}|)$ for $j = p-1, \ldots, 1$.
Let $sign(c'_j) =$"+" if $x'_j = 0$ and $sign(c'_j) =$"−" if $x'_j = 1, j \in N'(0,1)$.
Finally, reset $BaseCost = |c'_1|$ $(=\max_{j \in N'(0,1)} |c'_j|)$.

The new $|c'_j|$ values produced by this rule will retain the same ordering as the original ones and the signs of the c'_j coefficients will be preserved to be consistent with the target values x'_j.

In a departure for diversification purposes, foregoing rule can be changed by modifying the next to last step to become

Set $|c'_1| = BaseCost$ and $|c'_{j+1}| = \min(|c'_j| + \Delta_{j+1}, \lambda|c'_j|)$ for $j = 1, \ldots, p-1$

and concluding by resetting $BaseCost = |c'_p|$.

8.3 Multiple Choice Problems

The Phase 2 procedure can be specialized to provide an improved method for handling (MIP:0-1) problems that contain multiple choice constraints which

take the form

$$\sum_{j \in N_q} x_j = 1,\ q \in Q$$

where the sets $N_q, q \in Q$, are disjoint subsets of N.

Starting with all $j \in N_q$ unassigned (hence $N_q \subset N'(\Phi)$), the specialization is accomplished by only allowing a single $j \in N_q$ to be transferred from $N'(\Phi)$ to $N'(1)$. Once this transfer has occurred, let $j(q)$ denote the unique index $j \in N'(1) \cap N_q$ and let $x_{j(q)}$ denote a resisting variable, hence $j(q) \in NR$. After disposing of blatant resistors, we are assured that such a resisting variable satisfies $j(q) \in NF$ (i.e., $x_{j(q)}$ is fractional).

We seek a variable x_{j^*} to replace $x_{j(q)}$ by selecting

$$j^* = \underset{j \in NR \cap (N_q - \{j(q)\})}{\arg\max} RR_j - T$$

(note $j \in NR \cap (N_q - \{j(q)\})$ implies $j \in N'(0)$), or if no such index j^* exists, selecting

$$j^* = \underset{j \in N'(\Phi) \cap N_q}{\arg\max} RR_j - T.$$

Then j^* is transferred from its present set, $N'(0)$ or $N'(\Phi)$, to $N'(1)$, and correspondingly $j(q)$ is transferred from $N'(1)$ to either $N'(1)$ or $N'(\Phi)$. After the transfer, $j(q)$ is re-defined to be given by $j(q) = j^*$.

8.4 Generating the targets x'_j and coefficients c'_j post-optimization

More advanced evaluations for generating new target assignments and objectives for the Phase 2 procedure can be created by using linear programming post-optimization. The approach operates as follows.

The procedures described in Section 6 are used to generate a candidate set $N(k)$ of some number k of "most promising options" for further consideration. The resulting variables $x_j, j \in N(k)$, are then subjected to a post-optimization process to evaluate them more thoroughly. We denote the value k for the present approach by $k^\#$, where $k^\#$ can differ from the k used in the component rules indicated in Section 6. ($k^\#$ may reasonably be chosen to lie between 3 and 8, though the maximum value of $k^\#$ can be adapted from iteration to iteration based on the amount of effort required to evaluate the current variables that may be associated with $N(k^\#)$.) By the nature of the process described below, the value of k in Phase 2 will be limited to satisfy $k \leq k^\#$.

As in the Phase 2 procedure, there are two cases, one where no resistors exits and $N(k^\#)$ is composed of fractional variables from the set $NF(\Phi)$, and the other where resistors exist and $N(k^\#)$ is composed of resistors and potential resistors from the set $N'(0,1)$. To handle both of these cases, let c_{max} denote the absolute value of the maximum c'_j coefficient normally assigned on the current iteration, i.e., $c_{max} = |c'_1|$ where c'_1 is identified by the indexing used to create the candidate set $N(k^\#)$. Also, let v''_0 $(= c'x'')$ denote the

objective function value for the current solutions x'' of LP(c', x'), where we include reference to the associated value x''_0 by including x_0 as a secondary objective, as discussed in Section 1. (i.e., implicitly $v''_0 = c'x'' + \varepsilon x''_0$ for some small value ε. The reference to x''_0 is particularly relevant to the case where no resistors exist, since then $c'x'' = 0$.)

We then evaluate the assignment that consists of setting $x'_j = 0$ and $c'_j = c_{max}$ or setting $x'_j = 1$ and $c'_j = -c_{max}$ for each $j \in N(k^{\#})$, to determine the effect of this assignment in changing the value of v''_0 upon solving the new LP(c', x') (i.e., the form of LP(c', x') that results for the indicated new value of x'_j and c'_j). To avoid undue computational expense, we limit the number of iterations devoted to the post-optimization performed by the primal simplex method to solve the new LP(c', x'). (The post-optimization effort is unlikely to be excessive in any event since the new objective changes only the single coefficient c'_j.)

Denote the new x'_j and c'_j values for the variable x_j currently being evaluated by $x^{\#}_j$ and $c^{\#}_j$ and denote the new v''_0 that results by the post-optimization process by $v^{\#}_0$. Then we employ $v^{\#}_0$ to evaluate the merit of the option of assigning $x'_j = x^{\#}_j$ and $c'_j = c^{\#}_j$.

Case 1. $N(k^{\#}) \subset NF(\Phi)$ (and there are no resistors).

Both of the options consisting of setting $x^{\#}_j = 0$ and $c^{\#}_j = c_{max}$ and of setting $x^{\#}_j = 1$ and $c^{\#}_j = -c_{max}$ exist for each $j \in N(k^{\#})$. Denote the quantity $v^{\#}_0$ for each of these two options respectively by $v^{\#}_0(j:0)$ and $v^{\#}_0(j:1)$. Then we prefer to make the assignment $x^{\#}_j = v^{\#}$ where

$$v^{\#} = \arg\min_{v \in \{0,1\}} v^{\#}_0(j:v),$$

and we select the index $j^{\#} \in N(k^{\#})$ for this assignment by

$$j^{\#} = \arg\max_{j \in N(k^{\#})} EV_j,$$

where EV_j is the evaluation given by $EV_j = |v^{\#}_0(j:0) - v^{\#}_0(j:1)|$. (Greater refinement results by stipulating that

$$\min_{v^{\#}_0(j:1)} v^{\#}_0(j:0)$$

equals or exceeds a specified threshold value, such as the average of the $\min(v^{\#}_0(j:0), v^{\#}_0(j:1))$ values over $j \in N(k^{\#})$.)

Case 2. $N(k^{\#}) \subset N'(0,1) - T$ (and resistors exist)

In this case only a single option exists, which consists of setting $x^{\#}_j = 1 - x'_j$ and setting $c^{\#}_j = c_{max}$ or $-c_{max}$ according to whether the resulting $x^{\#}_j$

is 0 or 1. The evaluation rule is therefore simpler than in Case 1: the preferred $j^\#$ for implementing the single indicated option is given simply by

$$j^\# = \arg\min_{j \in N(k^\#)} v_0^\#.$$

In both Case 1 and Case 2, if more than one x_j is to be assigned a new target, the elements of $N(k^\#)$ can be ordered by the indicated evaluation to yield a subset that constitutes the particular $N(k)$ used in Phase 2 (where possibly $k < k^\#$).

The foregoing approach can be the foundation of an aspiration criterion for determining when a tabu element should be relieved of its tabu status. Specifically, for Case 2 the set $N(k^\#)$ can be permitted to include some small number of tabu elements if they would qualify to belong to $N(k^\#)$ if removed from T. ($k^\#$ might be correspondingly increased from its usual value to allow this eventuality.) Then, should the evaluation $v_0^\#(j : x_j^\#)$ be 0 for some such $j \in T$, indicating that all variables achieve their target values in the solution to the associated problem $\mathrm{LP}(c', x')$, and if no $j \in N(k^\#) - T$ achieves the same result, then the identified $j \in T$ may shed its tabu status and be designated as the preferred element $j^\#$ of Case 2.

8.5 Target Analysis

Target analysis is a strategy for creating a supervised learning environment to determine effective parameters and choice rules (see, e.g., Glover and Greenberg [10]; Glover and Laguna [11]). We refer to a target solution in the context of target analysis as an *ultimate* target solution to avoid confusion with the target solutions discussed in preceding sections of this paper. Such an ultimate target solution, which we denote by x^t, is selected to be an optimal (or best known) solution to (MIP:0-1). The supervised learning process of target analysis is applied to identify decision rules for a particular method to enable it to efficiently obtain solutions x^t to a collection of problems from a given domain. The approach is permitted to expend greater effort than normally would be considered reasonable to identify the solutions x^t used to guide the learning process (unless by good fortune such solutions are provided by independent means).

Target analysis can be applied in the context of both the Phase 1 and Phase 2 procedures. We indicate the way this can be done for Phase 2, since the Phase 1 approach is simpler and can be handled by a simplified variant. The target analysis operates by examining the problems from the collection under consideration one at a time to generate information that will then be subjected to a classification method to determine effective decision rules. At any given iteration of the Phase 2 procedure, we have available a variety of types of information that may be used to compose a decision rule for determining whether particular variables x_j should be assigned a target value of $x'_j = 0$ or

$x'_j = 1$. The goal is to identify a classification rule, applied to this available information, so that we can make correct decisions; i.e., so that we can choose x_j to receive its value in the solution x^t, given by $x'_j = x^t_j$.

Denote the information associated with a given variable x_j as a vector I_j ("I" for "information"). For example, in the present setting, I_j can consist of components such as $x''_j, x^0_j, V_j, |c'_j|, \Delta_j, RM_j, \lambda_j, RR_j, R\lambda_j$, and so forth, depending on whether x_j falls in category identified in Step 2(a) or Step 2(b) of the Phase 2 procedure. The advanced information given by the values $v^{\#}_0(j:v)$ discussed in Section 8.4 is likewise relevant to include. For those items of information that can have alternative definitions, different components of I_j can be created for each alternative. This allows the classification method to base its rules on multiple definitions, and to identify those that are preferred. In the case of parameterized evaluators such as RR_j, different instances of the evaluator can be included for different parameter settings (values of λ), thus allowing preferred values of these parameters likewise to be identified. On the other hand, some types of classification procedures, such as separating hyperplane methods, automatically determine weights for different components of I_j and for such methods the identification of preferred parameter values occurs implicitly by reference to the weights of the basic components, without the need to generate multiple additional components of I_j.

From a general perspective, a classification procedure for Phase 2 may be viewed as a method that generates two regions $R(0)$ and $R(1)$, accompanied by a rule (or collection of rules) for assigning each vector of information I_j to exactly one of these regions. The goal is compose $R(0)$ and $R(1)$ and to define their associated assignment rule in such a fashion that I_j will be assigned to $R(0)$ if the correct value for x_j is given by $x^t_j = 0$, and will be assigned to $R(1)$ if the correct value for x_j is given by $x^t_j = 1$. Recognizing that the classification procedure may not be perfect, and that the information available to the procedure may not be ideal, the goal more precisely is to make "correct assignments" for as many points as possible. We will not discuss here the relative merits of different types of classification methods, but simply keep in mind that the outcome of their application is to map points I_j into regions $R(0)$ and $R(1)$. Any reasonable procedure is likely to do a very much better job of creating such a mapping, and hence of identifying whether a given variable x_j should be assigned the value 0 or 1, than can be accomplished by a trial and error process to combine and calibrate a set of provisional decision rules. (An effort to simply "look at" a range of different data points I_j and figure out an overall rule for matching them with the decisions $x_j = 0$ and $x_j = 1$ can be a dauntingly difficult task.)

In one sense, the classification task is easier than in many classification settings. It is not necessary to identify a correct mapping of I_j into $R(0)$ or $R(1)$ for all variables x_j at any given iteration of Phase 2. If we can identify a mapping that is successful for any one of the variables x_j in the relevant category of Step 2(a) or Step 2(b) of the Phase 2 procedure, then the procedure will be able to quickly discover the solution x^t. Of course, the mapping must

be able to detect the fact that assigning a particular I_j to $R(0)$ or $R(1)$ is more likely to be a correct assignment than one specified for another I_j. (For example, if a particular point I_j is mapped to lie "deeply within" a region $R(0)$ (or $R(1)$), then the classification of I_j as implying $x_j^t = 0$ (or $x_j^t = 1$) is presumably more likely to be correct. In the case of a separating hyperplane procedure, for instance, such a situation arises where a point lies far from the hyperplane, hence deeply within one of the two half-spaces defined by the hyperplane.)

The points I_j to be classified, and that are used to generate the regions $R(0)$ and $R(1)$ (via rules that map the points into these regions), are drawn from multiple iterations and from applications of Phase 2 on multiple problems. Consequently, the number of such data points can potentially be large and discretion may be required to limit them to a manageable number. One way to reduce the number of points considered is to restrict the points I_j evaluated to those associated with variables x_j for $j \in N(k)$. The smaller the value of k_{max}, the fewer the number of points generated at each iteration to become inputs for the classification procedure. Another significant way to reduce the number of points considered derives from the fact that we may appropriately create a different set of rules, and hence different regions $R(0)$ and $R(1)$, for different conditions. A prominent example concerns the conditions that differentiate Step 2(a) from Step 2(b).

Still more particularly, the condition $NF = NF(\Phi)$ of Step 2(a), which occurs when there are no fractional resistors, can receive its own special treatment. In this case we are concerned with determining target values for fractional variables that are not currently assigned such x'_j values. In an ideal situation, we would identify an optimal target value x'_j for some such variable at each step (considering the case for $k_{max} = 1$, to avoid the difficulty of simultaneously identifying optimal x'_j values for multiple variables simultaneously), and Phase 2 would then discover the solution x^t almost immediately. In addition, no resistors would ever arise, and the condition $NF = NF(\Phi)$ would be the only one relevant to consider at any step.

Consequently, to create a mapping and associated sets $R(0)$ and $R(1)$ for the condition $NF = NF(\Phi)$, it is appropriate to control each iteration of Phase 2 for the purpose of target analysis so that only "correct decisions" are implemented at each iteration. Then the data points I_j for $j \in NF(\Phi)$ are based on information that is compatible with reaching the ultimate target x^t at each step. (If an incorrect target x'_j were produced on some step, so that x_j is induced to receive the wrong value, then it could be that the "correct rule" for a new (different) variable x_h would not be to assign it the target $x'_h = x_h^t$, because the target $x'_h = 1 - x_h^t$ might lead to the best solution compatible with the previous assignment $x_j = x'_j$.)

Once such a controlled version of target analysis produces rules for classifying vectors I_j for $j \in N(\Phi)$, then these decision rules can be "locked into" the Phase 2 procedure, and the next step is to determine rules to handle the condition $NR \neq \emptyset$ of Step 2(b). Thus the target analysis will execute Phase 2

for its "best current version" of the rules for determining assignments $x_j = x'_j$ (which for Step 2(b) amounts to determining which variables should reverse their assignments to set $x'_j = 1 - x'_j$). This procedure can also be controlled to an extent to prevent the current collection of x'_j targets from diverging to widely from the x^t_j values. The resulting new rules generated by the classification method can then be embedded within a new version of Phase 2, and this new version can be implemented to repeat the target analysis and thereby uncover still more refined rules.

This process can also be used to identify aspiration criteria for tabu search. Specifically, an additional round of target analysis can be performed that focuses strictly on the tabu variables $x_j, j \in T$ on iterations where Step 2(b) applies. Then the classification procedure identifies a mapping of the vectors I_j for these tabu variables into the regions $R(0)$ and $R(1)$. A version of Phase 2 can then use this mapping to identify vectors I_j for $j \in T$ that lie deeply within $R(0)$ and $R(1)$, and to override the tabu status and drop j from T if I_j lies in $R(v)$ but $x'_j = 1 - v$. This rule can be applied with additional safety by keeping track of how often a tabu variable is evaluated as preferably being assigned a target value that differs from its current assignment. If such an evaluation occurs sufficiently often, then the decision to remove j from T can be reinforced.

8.6 Incorporating Frequency Memory

Tabu search methods typically incorporate frequency memory to improve their efficacy, where the form of such memory depends on whether it is intended to support intensification or diversification strategies.

Frequency memory already implicitly plays a role in the method for handling the tabu list T described in Section 6.3, since a variable that is frequently added to T is automatically prevented from initiating a refreshing operation, and hence T will continue to grow up to its limit of T_{max} elements until a variable that is less frequently added (or more specifically, that has not been added for a sufficient duration) become a member of T and launches an operation that causes T to shrink.

We consider two additional ways frequency memory can be employed within the Phase 2 procedure. The first supports a simple diversification approach by employing an array $Target(j : v)$ to record how many iterations x_j has been assigned the target value $v \in \{0, 1\}$ throughout previous search, or throughout search that has occurred since x^*_0 was last updated. (The type of frequency memory is called residence frequency memory.) The diversification process then penalizes the choice of an assignment $x_j = v$ for variables x_j and associated values v for which $Target(j : v)$ lies within a chosen distance from

$$\max_{j \in N, v \in \{0,1\}} Target(j : v),$$

motivated by the fact that this maximum identifies a variable x_j and value v such that the assignment $x_j = v$ has been in force over a larger span of previous iterations than any other target assignment.

A more advanced form of frequency memory that supports an intensification process derives from parametric tabu search. This approach creates an intensification score $InScore(x_j = x_j^\#)$ associated with assigning x_j the new target value $x_j^\#$, and takes into account the target assignments $x_h = x'_h$ currently active for other variables x_h for $h \in N'(0,1) - \{j\}$. The score is specifically given by

$$InScore(x_j = x_j^\#) = \sum_{h \in N'(0,1) - \{j\}} Freq(x_j = x_j^\#, x_h = x'_h)$$

where $Freq(x_j = x_j^\#, x_h = x'_h)$ denotes the number of times that the assignments $x_j = x_j^\#$ and $x_h = x'_h$ have occurred together in previously identified high quality solutions, and more particularly in the elite solutions stored in the reference set R as described in Section 7. Abstractly, $Freq(x_j = x_j^\#, x_h = x'_h)$ constitutes a matrix with $4n^2$ entries (disregarding symmetry), one for each pair (j, h) and the 4 possible assignments of 0-1 values to the pair x_j and x_h. However, in practice this frequency value can be generated as needed from R, without having to account for all assignments (all combinations of x_j and $x_h = 0$ and 1) since only a small subset of the full matrix entries will be relevant to the set R. The portion of the full matrix relevant to identifying the values $Freq(x_j = x_j^\#, x_h = x'_h)$ is also further limited by the fact that the only variables x_j considered on any given iteration are those for which $j \in N(k)$ where k is a relatively small number. (In the case treated in Step 2(b) of the Phase 2 Procedure, a further limitation occurs since only the single $x_j^\#$ value given by $x_j^\# = 1 - x'_j$ is relevant.)

By the intensification perspective that suggests the assignments that occur frequently over the elite solutions in R are also likely to occur in other high quality solutions, the value $InScore(x_j, x_j^\#)$ is used to select a variable x_j to assign a new target value $x_j^\#$ by favoring those variables that produce higher scores. (In particular, the assignments $x_j = x_j^\#$ for such variables occur more often in conjunction with the assignments $x_h = x'_h, h \in N'(0,1) - \{j\}$ over the solutions stored in R.)

9 Conclusions

Branch-and-bound (B&B) and branch-and-cut (B&C) methods have long considered the methods of choice for solving mixed integer programming problems. This orientation has resulted in attracting contributions to these classical methods from many researchers, and has led to successive improvements in these methods extending over a period of several decades. In recent years,

these efforts to create improved B&B and B&C solution approaches have intensified and have produced significant benefits, as evidenced by the existence of MIP procedures that are appreciably more effective than their predecessors.

It remains true, however, that many MIP problems resist solution by the best current B&B and B&C methods. It is not uncommon to encounter problems that confound the leading commercial solvers, resulting in situations where these solvers are unable to find even moderately good feasible solutions after hours, days, or weeks of computational effort. As a consequence, metaheuristic methods have attracted attention as possible alternatives or supplements to the more classical approaches. Yet to date, the amount of effort devoted to developing good metaheuristics for MIP problems is almost negligible compared to the effort being devoted to developing refined versions of the classical methods.

The view adopted in this paper is that metaheuristic approaches can benefit from a change of perspective in order to perform at their best in the MIP setting. Drawing on lessons learned from applying classical methods, we anticipate that metaheuristics can likewise profit from generating inequalities to supplement their basic functions. However, we propose that these inequalities be used in ways not employed in classical MIP methods, and indicate two principal avenues for doing this: the first by generating the inequalities in reference to strategically created target solutions and target objectives, as in Sections 3 and 4, and the second by embedding these inequalities in special intensification and diversification processes, as in Sections 5 and 7 (which also benefit by association with the targeting strategies).

The use of such strategies raises the issue of how to compose the target solutions and objectives themselves. Classical MIP methods such as B&B and B&C again provide a clue to be heeded, by demonstrating that memory is relevant to effective solution procedures. However, we suggest that gains can be made by going beyond the rigidly structured memory employed in B&B and B&C procedures. Thus we make use of the type of adaptive memory framework introduced in tabu search, which offers a range of recency and frequency memory structures for achieving goals associated with short term and long term solution strategies. Section 6 examines ways this framework can be exploited in generating target objectives, employing both older adaptive memory ideas and newer ones proposed here for the first time. Additional opportunities to enhance these procedures described in Section 8 provide a basis for future research.

Acknowledgment

I am grateful to Said Hanafi for a preliminary critique of this paper and for useful observations about connections to other work. I am also indebted to César Rego for helpful suggestions that have led to several improvements.

References

1. Balas E Jeroslow R (1972) Canonical Cuts on the Unit Hypercube. SIAM Journal of Applied Mathematics, 23(1):60–69.
2. Dantzig G (1963) Linear Programming and Extensions. Princeton University Press, Princeton, NJ.
3. Davoine T Hammer PL Vizviári B (2003). A Heuristic for Boolean Optimization Problems. Journal of Heuristics 9:229–247.
4. Fischetti M Glover F Lodi A (2005) Feasibility Pump. Mathematical Programming – Series A 104:91–104.
5. Glover F (1978) Parametric Branch and Bound. OMEGA, The International Journal of Management Science 6(2):145–152.
6. Glover F (2005) Adaptive Memory Projection Methods for Integer Programming. In: Rego C Alidaee B (eds) Metaheuristic Optimization Via Memory and Evolution: Tabu Search and Scatter Search. Kluwer Academic Publishers.
7. Glover F (2006) Parametric Tabu Search for Mixed Integer Programs. Computers and Operations Research 33(9):2449–2494.
8. Glover F (2006a) Satisfiability Data Mining for Binary Data Classification Problems. Research Report, University of Colorado, Boulder.
9. Glover F (2007) Infeasible/Feasible Search Trajectories and Directional Rounding in Integer Programming. Journal of Heuristics, Kluwer Publishing (to appear).
10. Glover F Greenberg H (1989) New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence. European Journal of Operational Research 39(2):119–130.
11. Glover F Laguna M (1997) Tabu Search. Kluwer Academic Publishers.
12. Glover F Sherali HD (2003) Foundation-Penalty Cuts for Mixed-Integer Programs. Operations Research Letters 31:245–253.
13. Guignard M Spielberg K (2003) Double Contraction, Double Probing, Short Starts and BB-Probing Cuts for Mixed (0,1) Programming. Wharton School Report.
14. Hanafi S Wilbaut C (2006) Improved Convergent Heuristic for 0-1 Mixed Integer Programming. Research Report, University of Valenciennes.
15. Hvattum LM Løkketangen A Glover F (2004) Adaptive Memory Search for Boolean Optimization Problems. Discrete Applied Mathematics 142:99–109.
16. Nediak M Eckstein J (2007) Pivot, Cut, and Dive: A Heuristic for Mixed 0-1 Integer Programming. Journal of Heuristics, Kluwer Publishing (to appear).
17. Pardalos PS Shylo OV (2006) An Algorithm for Job Shop Scheduling based on Global Equilibrium Search Techniques. Computational Management Science (Published online), DOI: 10.1007/s10287-006-0023-y.
18. Patel J Chinneck JW (2006) Active-Constraint Variable Ordering for Faster Feasibility of Mixed Integer Linear Programs. Mathematical Programming (to appear).
19. Pedroso JP (2005) Tabu Search for Mixed Integer Programming. In: Rego C Alidaee B (eds) Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search. Kluwer Academic Publishers.
20. Shylo OV (1999) A Global Equilibrium Search Method. (Russian) Kybernetika I Systemniy Analys 1:74–80.
21. Soyster AL Lev B Slivka W (1978) Zero-One Programming with Many Variables and Few Constraints. European Journal of Operational Research 2(3):195–201.

22. Spielberg K Guignard M (2000) A Sequential (Quasi) Hot Start Method for BB (0,1) Mixed Integer Programming. Mathematical Programming Symposium, Atlanta.
23. Ursulenko A (2006) Notes on the Global Equilibrium Search. Working paper, Texas A&M University.
24. Wilbaut C Hanafi S (2006) New Convergent Heuristics for 0-1 Mixed Integer Programming. Research Report, University of Valenciennes.