Chapter 19

ADAPTIVE MEMORY PROJECTION METHODS FOR INTEGER PROGRAMMING

Fred Glover
Leeds School of Business, University of Colorado, Boulder, CO 80309-0419, USA,
fred.glover@colorado.edu

Abstract: Projection methods, which hold selected variables fixed while manipulating others, have a particularly useful role in metaheuristic procedures, especially in connection with large scale optimization and parallelization approaches. This role is enriched by adaptive memory processes of tabu search, which provide a collection of easily stated strategies to uncover improved solutions during the course of the search. Within the context of pure and mixed integer programming, we show that intensification and diversification processes for adaptive memory projection can be supported in several ways, including the introduction of pseudo-cut inequalities that additionally focus the search. We describe how the resulting procedures can be embedded in constructive multi-start methods as well as in progressive improvement methods, and how they can benefit by the application of target analysis.

Keywords: Adaptive Memory, Restricted Search Space, Cutting Planes, Tabu Search

1. Introduction

An old and recurring idea in optimization is to generate solutions by iteratively holding selected subsets of variables fixed at particular values while varying the values of other variables. The simplex method of linear programming is a familiar special case of this idea, where only a single independent variable, designated nonbasic, is allowed to vary at a time, to identify moves between adjacent vertices of the feasible solution polyhedron. The idea also surfaces in the area of cutting plane methods for integer programming by the strategy called lifting, which assigns the role of variables to cutting plane coefficients. A common form of lifting approach, for

example, successively identifies values to assign to coefficients that are free, given the values presently assigned to fixed coefficients, until a complete cutting plane is generated. More generally, the underlying concept resides at the heart of projection mappings, which are pervasively used in nonlinear and mixed integer programming.

The realm of metaheuristics affords an opportunity to apply this idea in new ways, invoking an enriched set of strategic considerations. Confronted with complex optimization problems where exact methods often fail, the issues of iteratively choosing fixed and free variables, and of controlling the processes for assigning values to variables that are free, become challenging on levels not encountered in more classical settings.

This paper adopts the theme of basing such a solution approach on the principles of tabu search, to create an *adaptive memory projection (AMP) method* for pure and mixed integer programming.

Three main concepts of tabu search provide the starting point for adaptive memory projection methods.

(1) strongly determined and consistent variables
(2) intensification/diversification tradeoffs
(3) persistent attractiveness

While the following discussion will be framed in terms of assigning values to variables, the concepts also apply directly to choosing moves in neighborhood spaces. In this case, "values assigned" may be considered to be the same as attributes imparted to solutions as a consequence of selecting a move. The basic approach can be viewed from the perspective of decomposition, since each projection "breaks apart" the problem by partitioning the variables into the fixed and free classes.

1.1 Strongly Determined and Consistent Variables

The notion of strongly determined and consistent variables (Glover, 1977) is to keep track of variables that receive particular value assignments (or ranges of assignments) within some interval of frequency over high quality solutions. The concept also applies to keeping track of such assignments in solutions where changing these assignments would significantly modify the structure or quality of the solutions.[1] To identify and take advantage of such variables involves:

[1] We emphasize that in the metaheuristic context, a solution is defined relative to the neighborhoods employed. Infeasible solutions produced from neighborhoods that admit such alternatives are as relevant as any others. This is notably true in applications where solutions are produced by strategic oscillation (which may purposely construct infeasibilities) and by solving surrogate or Lagrangean or LP relaxations.

(a) Segregating sets of good solutions over which the variables and their assignments are identified (including the use of clustering to achieve this segregation);
(b) Applying measures of frequency to determine which variables qualify as *consistent* (Such measures underlie the type of frequency memory used in intensification strategies in tabu search, noting that the meaning of "consistency" in the present case is determined by reference to a subset of good solutions.)
(c) Applying criteria of change to determine which variables qualify as *strongly determined.* (This relates to the notion of influence in tabu search, which has an important role in diversification strategies.)

The variables identified by the foregoing mechanisms are exploited by constraining them to receive their preferred values (or lie in their preferred ranges), and then searching more intensively over the remaining variables. As expressed in Glover (1977), the key premises underlying the approach are:

- such variables are likely to receive their preferred values in other high quality solutions, including optimal and near-optimal solutions;
- constraining the variables to their preferred values creates a "combinatorial implosion" effect that accelerates the solution of the remaining problem;
- once selected variables are constrained in this manner, then other variables may also become strongly determined or consistent, allowing them to be exploited similarly.

1.2 Intensification/Diversification Tradeoffs

Although strongly determined and consistent variables were initially proposed for application in an intensification context, the tabu search emphasis on balancing intensification with diversification suggests the possibility of augmenting the use of these variables by periodically imposing restrictions or incentives to drive the solution into new regions. A coordinated treatment of such considerations is crucial for an adaptive memory projection method, and we subsequently discuss key elements for doing this in detail. It is to be noted that the idea of generating solutions by iteratively holding subsets of variables fixed at particular values while varying the values of other variables is also known in the constraint programming community as "Large Neighborhood Search" (LNS), whose terminology was introduced in Shaw (1998). Still more recently, the theme of identifying and exploiting variables by the criteria of being strongly determined and/or consistent, although again making use of different terminology, appears as a fundamental component of the framework proposed in Ahuja et al. (2002) for large scale neighborhood search, and also appears in

the innovative metaheuristic design proposed by Danna et al. (2003) for mixed integer programming problems.

1.3 Persistent Attractiveness

The notion of persistent attractiveness (e.g., Glover, 2000) underscores the utility of identifying assignments of values to variables that receive high evaluations, but which are not executed by a search method (as where, for example, other assignments may receive still higher evaluations). Assignments that often receive high evaluations are *persistently attractive*, and deserve to be the focus of strategies that impose such assignments just as if the variables qualified as "consistent" in the sense identified earlier. It is also relevant to keep track of variables that are *conditionally attractive*, i.e., that may rarely receive high evaluations, but that receive very high evaluations at critical points. These represent a form of strongly determined variables, because the fact that they rapidly became less attractive is a sign that, had they been given their high-evaluation assignments, they would have been disposed to change the structure of the solution in a significant way.

2. Fundamentals of an Adaptive Memory Projection Method

Adaptive Memory Projection Methods have several versions. We begin by examining a version that is easy to implement, and that employs the proposal of coordinating TS with the application of exact optimization procedures over sub-problems that are kept small enough that such problems can be solved efficiently by these procedures. An outline of the AMP method is as follows.

2.1 Overview of an Adaptive Memory Projection Approach

Step1. (*Initiation*) Generate a starting solution. (Alternatives for doing this are discussed later.)

Step2. (*Search Phase*) Apply a heuristic search for some limited number of iterations. Keep track of the variables whose values are changed, and also keep track of variables (and their assignments) that qualify as persistently and conditionally attractive. (Changes in assignments may be accomplished by multiple types of moves.)

Step3. (*Referent-Optimization Phase*)[2] Choose a subset of the variables recorded in Step 2 that were changed or that qualified as persistently or conditionally attractive. Apply an exact method to solve the *referent-problem* that allows these variables to be free, while remaining problem variables are held fixed at the values they received in the best solution found by the search in Step 2.

Step4. (*Re-launch the Search*) Using recency and frequency memory, and imposing associated restrictions (as subsequently identified), perform a new heuristic search starting from the solution obtained in Step 3.

Step5. (Diversification and Renewed Solution Pass) Drive the search into a new region, using longer term memory and standard TS diversification processes, and repeat Steps 2-4.

We amplify the steps of the preceding outline by several key observations.

(1) A restricted form of the heuristic search of Step 2 can be applied by keeping the number of iterations small enough that all variables recorded (changed or identified as attractive) can be selected to be free in Step 3. If the heuristic search is performed for a larger number of iterations, then a screening step must be performed to select a preferred subset of variables to be treated as free. (The operation of holding certain variables fixed in Step 3 can of course be exploited by temporarily "reducing" the problem, i.e., by adjusting the constraint requirements and dropping the fixed variables from consideration.)

(2) All variables changed in Step 2 are automatically included among candidates to become free variables in the next referent-optimization of Step 3; i.e., if a variable changes and then changes again to return to its original value, it still qualifies to be selected as a free variable of the next referent-optimization.

(3) An advanced and intensive heuristic search may be performed to carry out Step 3 in place of an exact procedure. In such a process, and in successive applications of Step 3, such an approach may keep a longer term frequency memory $f(j)$ that records how often each variable x_j is a free variable for the referent-optimization step. Then the inclusion of x_j as a free variable can be penalized according to the size of $f(j)$ (e.g., as compared to the average $f(j)$ value), to achieve greater diversification.

(4) Each new application of the improving heuristic at Step 4, after running the exact method at Step 3, generates new evaluations for assigning the variables particular values. Variables that receive the highest evaluations to be changed go on the list of variables to be considered as free variables

[2] This terminology comes from Glover and Laguna (1997), where this type of approach is viewed within a broader framework called "referent-domain optimization." See also Mautor and Michelon (1997, 2001).

on the next execution of the exact method. These variables include both those that were selected to be changed and those that were not selected to be changed (but which nevertheless were strong candidates to be selected, e.g., that were strongly or persistently attractive).

The policy of discouraging the selection of variables that were free on the most recent preceding referent-optimization should also affect the evaluations on the current heuristic solution step. Thus, if a variable is tabu to change, its evaluation is considered to be unattractive, unless the associated move satisfies an aspiration criterion permitting its inclusion on the list of candidates to become free variables.

(5) A diversification effect can be introduced within Step 3 by including one or more *pseudo-cuts* that assure a solution will be found that is different from the one found by the preceding heuristic solution effort.[3]

To illustrate, consider just two solutions, the one that initiates the search of Step 2 (or Step 4) and the one that is the best solution found during this step. (Subsequent comments address the situation where these solutions may be the same.) Let $x_1,...,x_p$ be the variables whose values were increased, and let $y_1,...,y_q$ be the variables whose values were decreased, in going from the initial solution to the best solution. Also, let $x_1^{'},...,x_p^{'}$ and $y_1^{'},...,y_q^{'}$ be the values of these variables in the initial solution.

Let *Increase* denote the amount by which the x variables increased and let *Decrease* denote the amount by which the y variables decreased. (Both of these are positive numbers.) For the values given to x and y in the best solution found during the search, the variables therefore satisfied:

$$(x_1 - x_1^{'}) + (x_2 - x_2^{'}) + \cdots + (x_p - x_p^{'}) = increase$$

and

$$(y_1^{'} - y_1) + (y_2^{'} - y_2) + \cdots + (y_q^{'} - y_q) = Decrease$$

The pseudo-cut that compels these variables to produce a change at least k less than the sum of *Increase* and *Decrease* is:

[3] Pseudo-cuts (inequalities that may not be satisfied by optimal solutions) are also used within tabu search methods for mixed integer programming as a basis for generating moves. The tabu search processes that allow the foundations of previous moves to be selectively discarded, and that accordingly allow invalid or unproductive inequalities to be removed, prove especially useful in this context. (See the references cited in Chapter 6 of Glover and Laguna, 1997, particularly in connection with tabu branching.)

$$(x_1 - x_1') + \cdots + (x_p - x_p') + (y_1' - y_1) + \cdots + (y_q' - y_q) \leq \quad \text{(A)}$$
$$Increase + Decrease - k$$

A possible value for k can be $(Increase + Decrease)/2$, for example, although k should normally be restricted from being very large.) In the 0-1 context, $Increase = p$ and $Decrease = q$.

This same idea can be applied to include additional variables in (A), by allowing variables whose values do not change to be classified either as producing a 0 increase or a 0 decrease. Assigning a "no-change" variable to the *Increase* set (treating it as an x variable) tends to strengthen (A), while assigning it to the *Decrease* set (treating it as a y variable) tends to weaken (A). In either case, the expanded inequality is binding so long as k is positive.[4]

The pseudo-cut (A) can also be used in additional strategic ways. Specifically, even without first applying a search to change the initial solution, we can hypothesize that a selected subset of values may increase (or stay the same) and another may decrease (or stay the same), to identify the x and y variables. Then, an inequality for guiding the search can be generated by postulating values for *Increase* and *Decrease*, and choosing an associated value for k. More simply, we can directly choose a value $k^* = Increase + Decrease - k$ that represents the constant term (right hand side) of (A). An extreme example of this occurs for the case of 0-1 variables by specifying the x variables to be those for which $x_j' = 0$ and the y variables to be those for which $y_j' = 1$. Then the inequality for a chosen value of k^* becomes the same as the "local branching" inequality of Fischetti and Lodi (2002). The AMP framework that includes (A) therefore offers a set of strategies that subsume the local branching scheme, and suggests the merit of integrating adaptive memory projection with local branching ideas.[5]

(6) In the situation where the heuristic search of Step 2 (or Step 4) does not identify a solution that is better than the starting solution for this step, the following approach can be used to introduce one or more pseudo-cuts in Step 3 to achieve a diversification effect. This type of approach is useful as well for the case where the starting and best solutions found in Step 3 are the same, and applies particularly to 0-1 problems.

[4] The use of weakened versions of (A) provide interesting possibilities for making the search more flexible.

[5] A more general approach that goes beyond local branching, and that likewise invites integration with adaptive memory projection, is provided by Surrogate Branching, as described in Glover, Fischetti and Lodi (2003).

Redefine $x_1,...,x_p$ to be variables equal to their lower bounds (0), and $y_1,...,y_q$ to be variables at their upper bounds (1) in the starting solution, where we now restrict consideration to those variables that did not change their values, i.e., whose values are the same in the starting solution and in the best solution obtained. (Hence, if the best solution obtained is no different from the starting solution, all variables are candidates to be considered.) We further restrict attention to variables that are included among the free variables for the next referent-optimization iteration. (For example, these can consist of variables that looked attractive to change at some point, although they were not changed. Or they could have been changed but then were changed back.)

Then choose small positive integer values of *Increase* and *Decrease* (where $Increase < p$ and $Decrease < q$) and impose one or both of the pseudo-cuts

$$x_1 + x_2 + \cdots + x_p \geq Increase \qquad \text{(B1)}$$

$$(1 - y_1) + (1 - y_2) + \cdots + (1 - y_q) \geq Decrease \qquad \text{(B2)}$$

An alternative is to impose the pseudo-cut inequality

$$x_1 + x_2 + \cdots + x_p + (1 - y_1) + (1 - y_2) + \cdots + (1 - y_q) \geq Increase + Decrease \qquad \text{(B3)}$$

The upper bound of 1 in these inequalities can be replaced by a more general upper bound value U_j applicable to the associated variable.

We can expand the range of variables included in the foregoing inequalities by incorporating those whose values change, analogous to the previously indicated expansion of (A). We can similarly use these inequalities in additional strategies, by allowing alternative interpretations of the x and y variables.

These same ideas can also be used to assure that new solutions generated in Step 3 are driven away from other solutions previously found, not just those of the immediately preceding execution of Step 2 (or 4).[6]

[6]We add a brief comment to amplify on the use of "persistently attractive" variables in the situation where Step 2 (or Step 4) fails to find an improved solution. In this case, free variables will be chosen from among those that had attractive

(7) To apply Step 4 by re-launching the search, a natural strategy is to apply tabu restrictions to insure that at least one of the variables whose value is changed on any of the first several moves of the search must come from outside the set of variables that were free in the referent-optimization phase of Step 3. In other words, all free variables from this immediately preceding phase are treated as "tabu attributes" whose tenure lasts for a specified number of iterations. (This tenure may be as small as 2-4 iterations, or may be somewhat larger, e.g., 8 – 10 iterations, in a more strongly diversifying approach. Periodic shifts between tenure sizes are relevant, as in an adaptive tenure design.) A move is designated tabu if all of the variables it changes are tabu. A stronger tabu restriction can be imposed for the first few iterations (e.g., 1 to 3 iterations) by designating a move to be tabu if any of the variables it changes are tabu.

(8) For 0-1 mixed integer programming problems, a strategy to establish greater diversification by Step 4 occurs by using "diversity thresholds." Let $x_j, j \in Free$ identify the set of variables that were free in the execution of Step 3 that immediately precedes the execution of Step 4, and let $x_j, j \in Fixed$, identify the associated set of fixed variables. Also, let x_j^* denote the values of these variables in the optimal solution obtained by Step 3 (where x_j^* is simply the fixed value of x_j in the case of a fixed variable).

---

evaluations during the pass (or that belonged to moves having attractive evaluations). Once the method reaches an improving phase, whether or not a solution better than the starting solution is obtained, the moves from this phase may be considered as sources for new free variables. Typically, not all moves that receive high evaluations (relative to alternative moves) will be chosen during an improving phase. Consequently, there can be more variables that qualify as "attractive" than those that belonged to moves that were chosen. (Some of these may have belonged to more than one move on the same iteration, and some may have belonged to different moves on different iterations.) The attractiveness of a variable may be measured by reference to the attractiveness of the moves that contain it. A variable that belongs to one of the most highly attractive moves at a given iteration, but then which vanishes from the attractive category, is also important to consider.

Diversity Threshold Procedure 1

Let $Free(1) = \{j \in Free : x_j^* = 1\}$ and $Free(0) = \{j \in Free : x_j^* = 0\}$.

Record $FreeSum(1) = \sum_{j \in Free(1)} x_j$ and $FreeSum(0) = \sum_{j \in Free(0)} x_j$.[7]

To generate a solution that departs from the solution of Step 3 we seek to avoid the situation where $FreeSum(1) = |Free(1)|$ and $FreeSum(0) = 0$. Define $FreeDiversitySum = |Free(1)| - FreeSum(1) + FreeSum(0)$ Identifying the number of variables $x_j, j \in Free$, such that $x_j \neq x_j^*$. Then we want to assure $FreeDiversitySum \geq FreeDiversityThreshold$ where $FreeDiversityThreshold = 2$ or 3, etc. (larger for more diversification). This can be done by defining a move to be tabu if it will make *FreeDiversitySum* fall below *FreeDiversityThreshold* (For most simple types of moves, the condition only needs to be checked when *FreeDiversitySum* drops as low as $FreeDiversityThreshold + 1$.)

Diversity Threshold Procedure 2

Let

$Fixed(1) = \{j \in Fixed : x_j^* = 1\}$

$Fixed(0) = \{j \in Fixed : x_j^* = 0\}$

Define $FixedSum(1)$ and $FixedSum(0)$ in the apparent way, and then define *FixedDiversitySum* relative to these preceding values, also in the apparent way. We seek to assure

$$FixedDiversitySum \geq FixedDiversityThreshold$$

for a suitable (relatively small) value of $FixedDiversityThreshold$.

[7] These sums can be easily updated at each move by checking if a variable changed by the move is in Free(1) or Free(0). For example, to facilitate the update, keep an array FreeMember(j) that receives the value 1 for j ∈ Free(1), the value 0 for j ∈ Free(0) and the value -1 for j ∈ Fixed.

This second approach is implicitly the basis for the tabu rule discussed in (7). That is, the tabu rule of (7) insures that *FixedDiversitySum* grows by at least 1 on each iteration, for some beginning number of iterations. Variants of these approaches can readily be constructed for problems involving integer variables other than 0-1 variables.

Combined Diversity Threshold Procedure

The two foregoing procedures can be combined to give a less restrictive approach. The combined procedure defines

$$DiversitySum = FreeDiversitySum + FixedDiversitySum$$

Then it is only necessary to select a (relatively small) value *DiversityThreshold* and require that every move assures

$$DiversitySum \geq DiversityThreshold$$

We can also stipulate that initial moves of the method are tabu unless they increase *DiversitySum* by at least 1 at each iteration – i.e., requiring that the initial moves must increase *DiversitySum* by at least 1 until *DiversitySum* reaches some minimum value (for example, 1 or 2 more than *DiversityThreshold*). This approach offers an alternative to the tabu rule of (7) that may possibly yield better results.

(9) The re-launched heuristic search of Step 4 may not find any improving moves to begin, and should operate by the usual tabu search design of selecting best admissible (non-tabu) moves from an intelligently generated candidate list, even if the selected moves cause the solution to deteriorate. The heuristic must be run for enough iterations to give a basis for selecting a proper number of new free variables for the next referent-optimization. As it proceeds, the heuristic continues to create new tabu restrictions in a standard way, to avoid cycling.

If the method is in an improving sequence at the point where enough new free variables can be selected for the next referent-optimization phase, the sequence is allowed to continue its improvement until reaching a new local optimum. (This may be viewed as a special aspiration criterion that accepts all improving moves at this point.)

(10) To spur additional diversification, a "weak tabu condition" can be implemented whenever improving moves exist by discouraging (but not preventing) the choice of improving moves that change the value of some variable to a value it had in the referent-solution. Other tabu restrictions

continue to operate normally in this phase. When a non-improving stage is entered during the search, then the weak tabu condition can be strengthened for a small number of iterations (2-4) to prevent moves that change the value of any variable back to the value it received in the referent-solution.

(11) The preceding approach can be applied by eliminating the use of the heuristic method in Steps 1 and 4. In other words, the procedure can simply use an adaptive memory design from a variant of TS that progressively selects different subsets of variables to be free, and then immediately undertakes the next referent-optimization phase in Step 3. In essence, the heuristic approach is not precisely discarded, but it only identifies variables that may be considered conditionally attractive as a basis for constituting the new set of free variables. (These variables are evaluated as candidates to receive new assignments, but no steps are performed to actually execute such assignments.) The conditionally attractive evaluation can be made from information provided by the exact method itself. For example, if the exact method solves linear programming sub-problems, as in a B&B approach, then a form of sensitivity or post-optimizing "look-ahead" analysis may be performed to help identify new candidates to constitute the next set of free variables.

More general TS diversification approaches can of course also be incorporated into Step 5. An adaptive memory projection approach can readily be embedded within a constructive method for generating a solution, for the case of diversification methods based on re-starting. Consequently, such a projection method can be used to drive a multi-start method, and can also be used (as above) as part of a process that progressively amends a current solution rather than constructing (or re-constructing) a solution from scratch.

3. Constructive Variant of an Adaptive Memory Projection Method

A constructive variant of an adaptive memory projection method, to be incorporated in a multi-start procedure, can be briefly sketched as follows by making use of ideas already discussed.

3.1 Constructive (Multi-Start) Procedure

Step 1. Generate a solution constructively (as by a strategy for successively choosing values to assign to variables), keeping track of persistently and conditionally attractive value assignments, as well as assignments actually made.

Step 2. Select a subset of variables that include some of those selected to receive specific values during the construction process and also some of those that were persistently and conditionally attractive. Then apply an exact method to solve the sub-problem that allows these selected variables to be free, while remaining problem variables are held fixed.

Step 3. Using an adaptive memory design to avoid duplicating sets of variables recently chosen, return to Step 3 to select a new subset of variables and again apply an exact method with these variables treated as free.

Early stages of such an approach can select subsets of variables in Step 2 that were chosen successively to receive their assigned values, and that first became attractive (if they qualify as persistently or conditionally attractive) during this sequence of assignments. Later applications of the step can reasonably mix the subsets to include variables assigned in non-consecutive orders. Once enough values are changed, the method reverts to become the same as the method originally described (which does not specifically make reference to constructive processes).

The observations of Section 2 are relevant for the constructive multi-start variant as well.

4. Adaptive Memory Projection and Target Analysis

Some concluding comments are relevant for applying the AMP approach, when a basic heuristic is used to identify a particular region to explore, and then another more advanced solution procedure is used to examine this region in depth.

Consider the use of a search neighborhood in which the moves consist of flipping values of 0-1 variables. Assume we have a corresponding MIP formulation of the problem to be solved, so that an exact 0-1 MIP method can take the role of the advanced method. Then the AMP approach can be used in a couple of straightforward ways as follows.

(1) First apply the current heuristic method. Maintain a record of k variables (e.g., $k = 20$)[8] that have changed their values in the most recent moves prior to reaching a local optimum, or prior to reaching some other intervention point that seems useful. (Fewer than k moves may be involved in identifying such variables, if a move modifies more than a single variable at a time.)

Also record an additional k variables that received high evaluations to change their assigned values, but not quite high enough to result in choosing the variables to receive such a changed assignment. Then apply the AMP approach by solving an MIP sub-problem over these

[8] In some applications, k may usefully be selected to be much larger, as illustrated by the work of Danna and Perron (2003).

$2k$ variables, which are allowed to be free while the other variables are fixed at the values assigned by the local optimum. This gives a solution at least as good as the one the heuristic found. (Instead of basing the process on a record of most recently changed variables, it can be based on a record of other critical changes following earlier suggestions.)

(2) Further exploit the preceding strategy by means of target analysis (Chapter 9 of Glover and Laguna, 1997), to learn how to improve the basic heuristic. In this case the AMP approach is applied by allowing more variables to be included in the sub-problem solved by the MIP method than would normally be desirable to include in this problem. (For example, this might involve choosing up to $4k$ variables for the sub-problem solved by the MIP method.)

Then, target analysis can operate by examining the cases where the MIP gives a better solution than found by the heuristic. These cases yield information about how to improve the heuristic by changing its choice rules to make better decisions.

To illustrate, it can be valuable to know if the variables that allow the MIP method to get better results are primarily those whose values were changed by the heuristic (a) multiple times, (b) at least once, or (c) never, but which received a high evaluation (or high average evaluation) in favor of receiving new value assignments in spite of not being selected for such a purpose. This knowledge provides a foundation for tests to further pinpoint the characteristics of the most important variables.

It is similarly relevant to identify variables that receive different values in the MIP solution than in the heuristic solution, thereby making it possible to investigate whether the heuristic evaluations should be amended by reference to associated historical information.

Such information may be embodied in the frequency that a variable receives an evaluation at a certain basic level in favor of being assigned a given value 0 or 1, and also embodied in the frequency that the variable receives an evaluation at a higher level in favor of such a value assignment. Target analysis is then used to identify whether some mix of these two types of frequencies supports the choice $x_j = 0$ or $x_j = 1$.

In the case where many alternative choices may exist to be evaluated, target analysis can be kept manageable by focusing on a selected subset of key variables. These may be a collection of variables that receive the highest evaluations according to particular decision rules to be analyzed, or they may be divided among variables whose evaluations by such rules predict appropriate values to be assigned and those whose evaluations predict inappropriate values. In this type of process multiple decision rules can be analyzed simultaneously, rather than requiring each one to be run separately. Thus, it may be discovered that one rule is more accurate under certain

conditions and another is more accurate under other conditions – an outcome that would not be discovered except by the use of target analysis, since in standard testing only the quality of the final solution would be known, without providing an ability to assess the merit of component choices (and to do so in relation to the settings in which these choices arise).

It is important to realize that clues of the type indicated only have to give approximate rules for identifying good values to be assigned to the variables. Suppose, for instance, the MIP solution method is applied to a larger sub-problem than normally would be considered, as in the approach described in (1), and gives values to 10 variables that are different than found by the heuristic. A rule may be regarded as only "halfway accurate" if it instead selects 20 variables as candidates to change their values. But if these 20 variables include the 10 that the MIP method determined should be changed, then the rule is extremely powerful – because the MIP method can be applied to a 20 variable problem and still do as well as applying it to a considerably larger problem. In short, if the rule is used in conjunction with the AMP method, then the method will always find the desired solution. Clearly, a good AMP method can result from a rule that is not this effective. But target analysis can be a useful supplement in the design stage, in order to devise a rule for the AMP method that performs better than those that might otherwise be employed.

References

Ahuja, R.K., O. Ergun, J.B. Orlin and A.P. Punnen (2002) " Survey of Very Large-Scale Neighborhood Search Techniques," *Discrete Applied Mathematics* 123:75–102.

Danna, E. and L. Perron (2003) Structured vs. Unstructured Large Neighborhood Search: A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs, ILOG Technical Report, ILOG, S.A.

Danna, E., E. Rothberg and C. Le Pape (2003) Exploring Relaxation Induced Neighborhoods to Improve MIP Solutions. ILOG Technical Report, ILOG, S.A.

Fischetti, M. and A. Lodi (2002) "Local Branching," Research Report, DEI, University of Padova and DEIS, University of Bologna.

Glover, F. (1977) "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, 8(1):156–166.

Glover, F. (2000) Multi-Start and Strategic Oscillation Methods – Principles to Exploit Adaptive Memory. Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, M. Laguna and J.L. Gonzales Velarde, eds., Kluwer Academic Publishers, 1–24.

Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers.

Glover, F., M. Fischetti and A. Lodi (2003) Surrogate Branching Methods for Mixed Integer Programming. Report HCES-04-03, Hearin Center for Enterprise Science, University of Mississippi.

Mautor, T. and P. Michelon (1997) Mimausa: A New Hybrid Method Combining Exact Solution and Local Search. MIC'97, 2nd Methaheuristics International Conference, Sophia Antipolis.

Mautor, T. and P. Michelon (2001) Mimausa: An Application of Referent Domain Optimization. Technical Report, Laboratoire d'Informatique d'Avignon.

Shaw, P. (1998) Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher and J.F. Puget, eds., Proceeding of CP '98, Springer-Verlag, 417–431.