TARGET ANALYSIS TO IMPROVE A TABU SEARCH METHOD
FOR MACHINE SCHEDULING

Fred Glover* and
Manuel Laguna†

Center for Applied Artificial Intelligence,
Graduate School of Business,
University of Colorado, Boulder, Colorado, U.S.A.

ABSTRACT

This paper explores the integrated Artificial Intelligence/Operations Research approach known as target analysis in application to tabu search. Target analysis is designed to give heuristic and optimization procedures the ability to learn what rules are best for solving particular classes of problems. We present the process involved in embedding the target analysis methodology in a tabu search method which is tailored for the solution of a class of single machine scheduling problems.

* Address for correspondence:
US West Chair in Systems Science, Center for Applied Artificial Intelligence, Graduate School of Business, Box 419, University of Colorado, Boulder, Colorado 809-0419, U.S.A.

† This work was done during the author's summer internship at the Advanced Knowledge Systems Group of US West Advanced Technologies.
1. INTRODUCTION

Target analysis is a procedure for creating improved problem solving methods [1–3]. A previous application of target analysis, for the scheduling of future energy production and refueling in a nuclear power plant operation [4], disclosed the advantage of incorporating an artificial intelligence component into a standard optimization technique. The purpose of our study is to show how the target analysis methodology can be embedded within the heuristic solution framework provided by tabu search, as a means of generating more effective decision rules. The particular tabu search procedure used for our experiment is one that was developed for the solution of a class of single machine scheduling problems [5]. Our study is characterized by the following innovations:

- Introducing scoring procedures to measure the goodness of a move based on elements that can be manipulated to create an assured path to a target solution.
- Characterizing "ghost schedule" evaluations embodied in matrix records that are adaptively generated during the solution process, and which are capable of being approximated by a target schedule to evaluate certain sequences of moves.
- Employing a relaxed aspiration level criterion, based on the tabu tenure of moves, to yield improved move choices.
- Introducing a secondary tabu list, as a complement to a standard tabu list, designed to prevent repetitions rather than reversals of moves.
- Developing an event-dependent criterion for tabu list membership, which is activated (both for recording and for preventing moves) only on the event of taking disimproving moves away from a local optimum.

The problem which served as a basis for developing and testing these ideas may be described as follows. A collection of N jobs, each arriving at time zero, is to be scheduled on a continuously available machine with the goal of minimizing the sum of the setup costs and linear delay penalties. Each job i (i = 1, 2, ..., N) requires \( t_i \) units of time on the machine and a penalty \( p_i \) is charged for each unit of time that job commencement is delayed after time zero; \( s_{ij} \) is the setup cost of scheduling job j immediately after job i. Two dummy jobs, 0 and \( N + 1 \), are included in every schedule, where \( t_0 = t_{N+1} = 0 \) and \( p_0 = p_{N+1} = 0 \). The costs \( s_{0j} \) and \( s_{N+1,j} \) are considered to be an initial setup charge and a cleanup cost, respectively. A schedule has the form:

\[
\Pi = \{0, \pi(1), \pi(2), \ldots, \pi(N), N + 1\}
\]

where \( \pi(i) \) is the index of the job in position i of the schedule. Since no precedence constraints are enforced and preemption is allowed, any permutation of the N jobs becomes a feasible schedule. The objective is to minimize the sum of the delay and setup costs for all jobs. In mathematical terms, we desire to

Minimize \( F(\Pi) = D(\Pi) + S(\Pi) \)

where

\[
D(\Pi) = \sum_{i=1}^{N} d_{\pi(i)} p_{\pi(i)}
\]

\[
S(\Pi) = s_{0,\pi(1)} + \sum_{i=1}^{N-1} s_{\pi(i),\pi(i+1)} + s_{\pi(N),N+1}
\]

\[
(\text{P1})
\]

and

\[
d_{\pi(i)} = \sum_{j=1}^{i-1} t_{\pi(j)}, \text{ for } i = 2, \ldots, N \text{ and } d_{\pi(1)} = 0
\]

In [5], four tabu search (TS) methods for the solution of P1 were discussed and compared. The computational experiments undertaken in that study allowed the identification of the best of the candidate TS procedures for the solution of P1 in terms of its ability to consistently find high quality solutions. An overview of this procedure is given in section 2. In section 3, we describe the target analysis methodology as applied to the selected procedure and report results of computational experiments designed to compare the "old" and the "improved" methods. Finally in section 4, we draw conclusions and discuss the relevance of our findings.

2. THE TABU SEARCH METHOD

Tabu search is a metastrategy for solving combinatorial optimization problems (see, e.g. [1, 6–8]). Broadly speaking, this technique is designed to be superimposed on any procedure whose operation can be characterized as performing a sequence of moves which lead the procedure from one trial solution (or solution state) to another. Each move is selected
from a set of currently available alternatives and is susceptible to being evaluated by one or more functions that measure its relative attractiveness in some local sense. When the solution produced by the move is feasible, the objective function value provides one such measure. The well-known hill climbing heuristics fall within this class of procedures. In general a hill climbing heuristic progresses from an initial feasible solution along a path that changes the objective function value in a uniformly descending or ascending direction (for minimization or maximization respectively) until no further improvement of the objective function is possible by means of the available moves. At the stopping point, the solution obtained is a local optimum which, for combinatorial problems, very rarely is also global. In this context, tabu search provides a guiding framework for exploring the solution space beyond points of local optimality.

The most basic form of tabu search consists of introducing tabu restrictions which classify certain moves as forbidden, together with aspiration criteria capable of overriding (when appropriate) the tabu status of some moves and releasing them from their tabu restrictions. These activities have a time dependent dimension which can be implemented by means of a short term memory function. More elaborate TS procedures include intermediate and long term memory functions to carry out additional strategic operations described in [7]. While the incorporation of one or more of these components may improve performance, we purposely selected for our study a method which contains only the fundamental short term memory function. A sequel to this study examines the use of target analysis in conjunction with longer term considerations. [9]

The previous investigation by Laguna et al. [5] suggested that there is an advantage in embedding more than one class of moves in a TS method designed for the solution of P1. Two classes of moves were shown to be very effective in combination, as measured by their ability to uncover improved solutions when guided by the tabu search framework. The first class of moves consists of the common pairwise exchange of two jobs, which we will call a swap move. If no other restrictions are imposed, such a move can examine \(N^*(N - 1)/2\) neighbor solutions. The second class of moves consists of taking a job out of its current position and inserting it somewhere else in the schedule, which we call a transfer move. This latter type of move is able to examine \(N^*(N - 3) + 2\) additional neighbors. The number of available moves of each kind can be decreased by imposing a restric-

A data structure is required to ensure that the tabu restriction is being properly handled (i.e., the correct moves are being classified as tabu and only during the period governed by the short term memory). For this fundamental (starting) version of our method it is sufficient to make use of two arrays, tabu_list and tabu_state. The first is a circular list that contains the index of the tabu jobs, where the length of the list determines the dimension of short term memory (tabu_size). The second is a counter of the number of times that each job appears in the list, allowing a quick check on the tabu status of any move. The
initialization and updating of these arrays are performed as described in [5]. (Alternative arrays for managing tabu status are also possible – see, e.g., [6, 7].)

The last element in our procedure is the aspiration level criterion. The purpose of this component is to allow “good” tabu moves to be selected if the aspiration level is attained. In the starting version of our method we employed standard aspiration criterion that allows the tabu status of a move to be overridden if a better solution than the best found so far can be obtained. An important property of such an aspiration level is its ability to add flexibility to the search while avoiding cycling. However, a weaker form of cycle avoidance is sometimes preferable, allowing the procedure to return to an earlier solution as long as the path away from this solution is not duplicated. The refinements introduced subsequently were based in part on exploiting (and controlling) the flexibility allowed by the aspiration criterion.

General Description of the Short Term Memory Component of Tabu Search

Drawing on the foregoing definitions, an outline of the TS method used to initiate our study is provided by Figure 1. The method starts by generating an initial feasible solution and then executing swap and transfer moves until no further improvement of the objective function is possible. The pre-processing step is therefore a simple hill climbing procedure whose purpose is to find a local optimum (which becomes the current best solution $T^*$ with objective value $F(T^*)$), and to determine the maximum distance that jobs will be allowed to move at any step during the search. The initialization step includes three main tasks:

1. Calculate the cumulative processing time and delay penalty for each position in the current schedule.
2. Generate and store the move values for swap and transfer moves.
3. Initialize the short term memory function.

At this point, the actual search begins by selecting the best admissible move, which can be either a swap or transfer move. This choice is based on defining the move evaluation (or move value) to be $F(T'') - F(T')$, where $T'$ and $T''$ are the solutions before and after the move. Then a “best move” is defined as an admissible move (swap or transfer) with the least move value.

Upon executing the move, the updating procedure is in charge of modifying the delay time and penalty, move values, and memory function elements so as to reflect the change in the current schedule. The method then checks for a possible improvement and, if it occurs, updates the best solution. The procedure is repeated until the termination criterion is met, i.e., until no improvement is achieved for a specified number of iterations, max_iter.

Three aspects of the procedure outlined in Figure 1 deserve special consideration: pre-processing, initializing, and updating. We devote the remainder of this section to the details of these activities. (Additional considerations relevant to selecting a best admissible move may be found in [5].)

Pre-processing

According to previous applications of tabu search there is no strong evidence that the technique performs better starting from a high quality solution than from a somewhat inferior solution, although there are advantages to starting from a small number of diverse solutions, generated by reference to a long term memory function [7, 10, 11]. However, since our goal is to focus on the short term memory function independent of other considerations, our TS procedure uses only one starting point, and we have chosen to make this a reasonably good feasible solution. Two one-pass heuristics (referred to as heuristics 1 and 2) based on the traveling salesman “nearest unvisited city” rule, were used to generate this solution [12] for details).

Once an initial solution is found, the swap and transfer move neighborhoods are examined and a move is executed if it decreases the value of the cur-

---

Figure 1. Tabu Search Procedure for P1.

Pre-process;
Initialize;
do{
  Select best admissible move;
  Update partial sum arrays;
  Execute move;
  Update current solution value;
  Update best solution (if improved solution found);
} while (iterations without improvement < max_iter);
rent objective function. This step is repeated until no improving swap or transfer move is available. The resulting schedule and corresponding value of the objective function become the starting point for the tabu search procedure.

During this stage we also identify the maximum distance (in number of positions) that any job will be allowed to be displaced from its current position during a single move. The purpose of this strategy is to reduce the number of available moves to be examined at each iteration. Initially, the maximum allowed distances for swap and transfer moves are set to $N - 1$ and $N$ respectively. After the pre-processing search is stopped the maximum swap distance is obtained as follows:

$$\text{swap distance} = \begin{cases} 
\min(20, N/2) & \text{if } S(\Pi) < 0.3*F(\Pi) \\
N - 1 & \text{otherwise.}
\end{cases}$$

The maximum transfer distance is then set to be one larger than the maximum swap distance. The rationale behind these selections is twofold. First, when $S(\Pi)$ is small enough, jobs are not likely to move very far away from the positions they currently occupy. (Frequently, due to the way the initial solution was generated, these positions are also fairly close to their “natural” positions.) Second, when a job preferably should be moved farther from its current position than the limiting distance, there generally arises a stage at which the job becomes attractive to move part of the way toward its destination, and then later to move the rest (or an additional part) of the way. By performing a succession of such steps the limiting distance does not pose a true barrier.

Initializing

Throughout the search procedure, the move evaluations require (in one way or another) the determination of partial sums of processing times and delay penalties. A simple but efficient way of speeding up these calculations is to create two arrays to store the cumulative value of the processing times and delay penalties for each position. These arrays, which will be referred to as $t_{\text{sum}}$ and $p_{\text{sum}}$, can be initialized as follows:

$$x_{\text{sum}}(k) = \sum_{i=1}^{i} x_{s(i)} \text{ for } x = t \text{ and } p, \text{ and } k = 1, \ldots, N.$$ 

The information in these arrays becomes immediately useful while generating the initial move values. These values are stored in the arrays, $\text{swap}_{\text{move}}(i,j)$ and $\text{tran}_{\text{move}}(i,j)$, where the former is the move value of swapping jobs in positions $i$ and $j$ in the current schedule $(i < j)$, and the latter is the move value of inserting job $\pi(i)$ immediately before the job in position $j$. These array entries are determined by adding the change (increase or decrease) in the setup costs, $\Delta S$, to the change in the delay penalties, $\Delta D$, which together compose the full change in the objective function value. Calculating $\Delta S$ is straightforward and is described in [5]. The use of $t_{\text{sum}}$ and $p_{\text{sum}}$, to accelerate the calculation of $\Delta D$, gives rise to slightly more complex formula. For swap moves, $\Delta D$ is given by:

$$\Delta D = (t_{\text{sum}}(j - 1) - t_{\text{sum}}(i - 1)) * (p_{\pi(i)} - p_{\pi(j)}) + (p_{\text{sum}}(j - 1) - p_{\text{sum}}(i - 1)) * (t_{\pi(i)} - t_{\pi(j)})$$

and for transfer moves the expressions are:

$$\Delta D = (t_{\text{sum}}(j - 1) - t_{\text{sum}}(i)) * p_{\pi(i)} - (p_{\text{sum}}(j - 1) - p_{\text{sum}}(i)) * t_{\pi(i)} \text{ if } i < j$$

$$\Delta D = (p_{\text{sum}}(i - 1) - p_{\text{sum}}(j - 1)) * t_{\pi(i)} - (t_{\text{sum}}(i - 1) - t_{\text{sum}}(j - 1)) * p_{\pi(i)} \text{ if } i < j$$

The last initialization activity, which creates the data structure used to implement the short term memory function, is a direct application of the ideas described earlier and is detailed in [5].

Updating

After a move has been chosen at the current iteration (but before it is actually executed), the values contained in $t_{\text{sum}}$ and $p_{\text{sum}}$ need to be modified. The updating procedure for these arrays, which depends on the kind of move that will be performed, is carried out as follows:

For the transfer move $T(\pi(i), j)$:

$$x_{\text{sum}}(k) = x_{\text{sum}}(k + 1) - x_{\pi(i)} \text{ for } x = t \text{ and } p, \text{ and } k = i, \ldots, j - 2.$$ 

For the swap move $S(\pi(i), \pi(j))$ where $i < j$:

$$x_{\text{sum}}(k) = x_{\text{sum}}(k + 1) + (x_{\pi(i)} - x_{\pi(j)}) \text{ for } x = t \text{ and } p, \text{ and } k = i, \ldots, j - 1.$$ 

The updating of the move values can be now performed by using the new $t_{\text{sum}}$ and $p_{\text{sum}}$ arrays.
discussed in [5] there is only a subset of move values that need to be modified at any iteration. This provides an additional source of efficiency for an otherwise cumbersome updating procedure.

We now show how the target analysis methodology can be applied to our TS procedure characterized in the preceding description.

3. TARGET ANALYSIS

The general target analysis procedure has been described by Glover and Greenberg [2, 3] with the goal of linking the artificial intelligence and operation research perspectives by means of hindsight analysis and pattern recognition techniques, to give heuristic or optimal solution procedures the ability to learn what rules are best to solve a particular class of problems. Many existing solution methods have evolved by settling, a priori, on a somewhat limited characterization of appropriate rules for evaluating decisions. In the TS method previously described, for example, the "best" move to make at each iteration is specified to be the admissible move with the least move value. However, this strategy does not guarantee that the selected move will lead the search in the direction of the optimal solution. The ultimate goal of target analysis is to create some form of "master decision rule" which will assist a heuristic or optimal method in making better choices (or moves) during the solution procedure. In a recent application, for example, target analysis was used to create a decision rule which improved the selection of branching variables and directions at each node of a branch and bound tree [7]. In general terms, this master decision rule is formed by a composite evaluation function and a threshold function value.

The remainder of this section is devoted to the application of target analysis to our TS method. Since the purpose of our study is not only to create an improved method for solving P1 but also to illustrate and test the target analysis methodology, we present details of each of the implementation steps.

3.1 Phase I

The beginning phase is characterized by the identification of a class of problems to be solved, the selection of representative members of this class, and the initial solution of representative problems. This phase consists of the following three steps.

Step 1

In order to apply the target analysis methodology it is necessary to define the class of problems that is desired to be solved. It is also required for the class members to have an appropriate degree of resemblance among each other. In the event that members of the selected class are significantly different (relative to exploitable characteristics that target analysis itself can help to identify), it is convenient to further divide the problems into subclasses. In our application, a class of randomly generated problems can be easily characterized by specifying the underlying distribution functions and parameters used. To provide a basis of comparison with outcomes of previous studies [5, 9, 12], we have chosen to solve a class of problems with processing times, delay penalties, and setup costs uniformly distributed in the ranges given by $1 \leq t_i \leq 12$, $30 \leq p_i \leq 110$, and $500 \leq s_q \leq 1500$ respectively.

Step 2

In this step a collection of representative problems is chosen from the previously selected class. The problems in this collection should exhibit a certain degree of diversity given their structural similarities (e.g., in our study the relationship between $D(\Pi)$ and $S(\Pi)$ is similar for all problems). Also, as a general rule, most (though preferably not all) of these problems will be smaller than those the method is ultimately expected to solve, for two reasons: (i) to reduce the amount of computational effort involved in their solution and (ii) to reduce the amount of information to be compared and classified, in order to facilitate the operation of human insight as well as the application of mathematical analysis.

For our study we have made use of selected members of a set of ten 20-job problems that have served as test problems in [5, 9, 12] and for which optimal solutions are known. In order to identify a proper collection of problems, the following experiment was performed. A solution attempt was made for each of the ten problems using our TS method, which consisted of starting from an initial solution created by heuristic 2 and performing 50 iterations with the short term memory tenure of the tabu list set to 7 moves. This experiment resulted in four problems for which the TS method was not able to find optimal solutions within the specified number of iterations. These problems (labeled 1, 2, 5, and 9) showed enough variety to be considered a suitable representative subset.
Step 3

In this step a massive solution effort is applied to the representative sample of the given class of problems, with the goal of obtaining optimal or exceptionally high quality solutions. Current state-of-the-art procedures are used in order to generate these optimal (or near optimal) solutions. Generally at this step, time is expended and resources are utilized in amounts that go beyond those considered reasonable in normal circumstances. The optimal solutions to the problems in our sample are known as a result of carrying out lengthy calculations with an optimal scheduling algorithm (based on a branch and bound/dynamic programming hybrid approach developed in [12]). Alternatively, with less effort (but still more than we would hope to expend on a routine basis), our TS method could have been used to obtain high quality solutions for the analysis by setting the termination criterion, max_iter, to a number much larger than usual (e.g., 2000). Although the solutions obtained in this way are not guaranteed to be optimal, their use does not diminish the improving effects of applying target analysis. (In fact, we have determined that our TS procedure succeeds in obtaining optimal solutions for the test problems by this means, and in considerably fewer iterations than 2000.)

3.2 Phase II

The most important elements of this phase are embodied in three activities: re-solving the problems, developing provisional choice rules, and gathering information. To achieve these goals it is necessary to trace each of the decisions, or moves, made by the method as well as to observe possible alternatives. Overall, this phase demands a clear understanding of the problem and the proposed solution procedure. In this phase learning is enhanced by a strong reliance on human insight, facilitated by generating comparative information that builds upon the information provided by the earlier efforts of Phase I.

Step 1

This step undertakes to re-solve each of the test problems. The goal is to find the solutions obtained in the first phase, but with the option of inducing the procedure to make "good" moves at every iteration. This has the advantage of reducing overall effort and of generating the types of intermediate solutions we would expect to encounter by creating choice rules that are highly effective. However, as will be seen, there can also be merit in analyzing situations where less desirable moves are employed.

In order to measure the goodness of a move, we introduce a scoring procedure that is based on the knowledge gained in the first phase. The idea is to develop one or more measures of the goodness of a move which are entirely independent of its customary evaluation, and which identify the degree to which the move causes the current solution to progress in the direction of the targeted optimal solution. To do this, we first construct solution scores, designed to measure the extent to which any given solution deviates from the target solution. Then the move scores are derived as the difference between the solution scores obtained before and after applying the moves.

It is important in this approach to assure that the solution scores are based on parameters that can be exploited by the moves available to the procedure, i.e., that make it possible to identify moves which unambiguously and progressively lead to the target solution. This is not possible by scoring solutions on the basis of objective function values, for example, because there generally exists no sequence of available moves that progressively leads to an optimal solution by reference to these values. (Otherwise, the choice rule already employed by the method would directly find such solution.)

We chose therefore to score solutions by reference to certain variables that can be used to characterize these solutions, and which are susceptible to controlled manipulation by the moves. The selected variables in the present case can be shown to correspond to specific integer zero-one variables in integer programming formulations of the scheduling problem, although we define them more directly in terms of natural scheduling parameters.

For our application, we created two different solution scores, as follows. The first is an absolute position score, which simply counts the number of jobs that occupy their targeted optimal positions. (A more refined score can be based on identifying the number of positions each job is displaced from its targeted position, seeking to minimize the sum of these values.) The second solution score is a relative position score which counts the number of jobs that are scheduled immediately after their targeted predecessor jobs. (The fixed positions 0 and N + 1 are referenced in order to obtain a complete count.) From these definitions, the absolute and relative position scores for the targeted optimal solution are N and N
+ 1, respectively, which represent the highest values that can be obtained by any solution. We elected to combine these scores into a single solution score by summing them, this yielding a (maximum) targeted value of $2N + 1$.

From the composite solution score thus determined, the score attributable to each move is therefore the total score of the schedule after the move less the total score of the schedule before the move. By definition, moves with positive scores have the ability to move the schedule "closer to" its target, and higher scoring moves generally produce a faster convergence than lower scoring moves. (However, ambiguities are possible, especially in the case of zero scores, and refinements of the form previously indicated may be relevant for settings more complex than those we encountered.)

The purpose of creating such move scores is to provide functional targets for the target analysis procedure. Specifically, the goal is to identify evaluation functions which can lead to the same decisions that would be provided by the move scores, where these evaluation functions make no use of the knowledge of target solutions obtained in Phase I. Obviously, this goal may be more ambitious than could possibly be achieved, and hence approximations must be sought. For example, one may seek to make decisions that resemble those provided by the scores "as often as possible" or in restricted situations where some other criterion of convergence (such as an improving objective function value) is not operable. The form of the evaluation functions used to approximate such a goal can be exceedingly varied, and for this reason the insight and pattern recognition capabilities of a human problem-solver are quite important at this stage. (Mathematical analysis may be expected to take a larger role as useful general classes of such functions are identified. Presently this stage may be viewed as creating a model, which secondarily admits the application of mathematical analysis in devising good values for its parameters.)

The use of tabu search to guide the solution process adds a further dimension to this step by compounding ordinary evaluation possibilities through its exploitation of history (as embedded in the use of the memory processes for implementing tabu restrictions and aspiration criteria). Thus target analysis has the option of manipulating these memory processes and their associated criteria of "move admissibility" as an additional set of functions for evolving an evaluation model.

To keep the amount of information to be processed within manageable bounds, we elected to examine only a subset of admissible moves, restricting the attention to those that received high evaluations by the customary criterion of objective function value change. Our choice was based on the assumption that a collection of high evaluation moves was more likely to include high scoring moves than a randomly generated collection of the same size. (Such an assumption underlies the use of the "highest evaluation" choice rule by tabu search, with the proviso that if the assumption can be demonstrated false, then the basis of evaluation deserves to be changed.) Operationally, therefore, we defined the available moves at each step to be those admissible swap and transfer moves with the ten best move values (i.e., yielding a collection of ten moves of each type). High scoring moves were in fact found within this set, supporting the basis for its use.

As noted earlier, the re-solution step is also concerned with the creation of provisional choice rules. At this point in the methodology, however, it is more important that the moves selected to provide inputs for creating the evaluation functions include those that qualify as good, and in our application we allowed this to dominate the goal of designing provisional choice rules to classify such moves more accurately than the evaluation function customarily used.

In order to illustrate the re-solution process, Table 1 shows a list of the sequence of "best" moves made while re-solving problem 2. The first column indicates the iteration number, the second to fifth columns contain the sequence of "best" moves and their corresponding move values, move scores, and objective function values after the move. The last columns show the information related to the actual moves made during the normal TS solution procedure.

The first row of Table 1 (for iteration 0) shows the total score and objective function value of the starting solution. The scores in subsequent rows correspond to the moves made in each iteration. The score of the current schedule can be calculated by adding all the scores from row 0 up to the corresponding row. For 20-job problem 2, heuristic 2 yielded a initial schedule with an objective function value of 79 870. After one swap and one transfer move in the pre-processing step, a local optimum with an objective function value of 79 169 was reached, resulting in the following schedule as the starting point:

$$\Pi_0 = \{1 2 3 4 8 6 5 9 7 10 13 17 15 12 16 11 18 14 20 19\}$$
Table 1. Re-Solution Process for Problem 2.

<table>
<thead>
<tr>
<th>Iter</th>
<th>Move</th>
<th>Value</th>
<th>Score</th>
<th>Objective</th>
<th>Move</th>
<th>Value</th>
<th>Score</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>31</td>
<td>79 169</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>T(11, 11)</td>
<td>391</td>
<td>2</td>
<td>79 560</td>
<td>T(3, 1)</td>
<td>88</td>
<td>-6</td>
<td>79 257</td>
</tr>
<tr>
<td>2</td>
<td>S(13, 15)</td>
<td>516</td>
<td>0</td>
<td>80 076</td>
<td>T(19, 16)</td>
<td>228</td>
<td>-1</td>
<td>79 948</td>
</tr>
<tr>
<td>3</td>
<td>S(17, 12)</td>
<td>-5</td>
<td>2</td>
<td>80 071</td>
<td>T(18, 21)</td>
<td>202</td>
<td>-4</td>
<td>79 687</td>
</tr>
<tr>
<td>4</td>
<td>T(16, 14)</td>
<td>-937</td>
<td>6</td>
<td>79 134</td>
<td>S(1, 4)</td>
<td>236</td>
<td>-1</td>
<td>79 923</td>
</tr>
</tbody>
</table>

It is important to point out that the sequence of “best” moves shown in Table 1 is by no means unique. In fact, there are a number of ways (possibly more efficient in terms of the number of iterations) in which the optimal solution of 79 134 could have been found starting from Π₀. However, more efficient moves (those with higher scores) may not be available for consideration as a result of the policy of restricting attention to a subset of moves with attractive move values.

The use of the scoring scheme during the re-solution procedure makes it possible to identify good moves at each iteration and to generate effective move sequences in the search process. In order to measure the performance of the evaluation functions to be created, we seek a means to relate this performance to the move scores, not simply on a one by one basis, but across a reasonably sized collection of moves made by the solution process as it would normally operate, without taking advantage of the move scores (since available information may not allow even the best of evaluation functions to pick a high scoring move at each step). A natural way of achieving this is by calculating the average move score for the moves made in a given solution attempt. In this way, a large average will often correspond to choosing a high percentage of moves with good scores (although it could be due to a few moves with very large scores).

Step 2

Before provisional choice rules are proposed, we undertake to measure the performance of the tabu search method operating under its current choice rule. This rule always selects the move with the least (best) move value from the set that qualifies as admissible (according to the tabu restrictions and aspiration criteria). Although this rule often correctly identifies moves that would be classified as good by the scoring scheme, there are instances (like the one shown in Table 1) where moves with good scores do not necessarily have attractive move values.

We then performed an experiment by re-solving each of the problems of our chosen collection, recording the score for each of the 200 executed moves (50 moves per problem). Independent of this score, moves were classified according to their move values as improving (if strictly less than zero) and non-improving (if greater than or equal to zero). The absolute and relative position scores for improving and non-improving moves were recorded separately, and are presented in Table 2. The average scores shown in the first two columns of Table 2 were calculated by dividing the sum of absolute and relative position scores, respectively, by the number of improving moves. Likewise, the second and third column average scores are the result of dividing the sum of the absolute and relative position scores for non-improving.

Table 2. Average Move Scores for Improving and Non-Improving Moves.

<table>
<thead>
<tr>
<th>Prob.</th>
<th>Improving Absolute</th>
<th>Improving Relative</th>
<th>Non-Improving Absolute</th>
<th>Non-Improving Relative</th>
<th>Total Absolute</th>
<th>Total Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.259</td>
<td>0.593</td>
<td>-0.304</td>
<td>-0.913</td>
<td>0.000</td>
<td>-0.100</td>
</tr>
<tr>
<td>2</td>
<td>1.261</td>
<td>0.783</td>
<td>-0.963</td>
<td>-0.556</td>
<td>0.060</td>
<td>0.060</td>
</tr>
<tr>
<td>5</td>
<td>0.591</td>
<td>0.318</td>
<td>-0.357</td>
<td>-0.286</td>
<td>0.060</td>
<td>-0.020</td>
</tr>
<tr>
<td>9</td>
<td>0.609</td>
<td>0.609</td>
<td>-0.481</td>
<td>-0.444</td>
<td>0.020</td>
<td>0.040</td>
</tr>
<tr>
<td>Average</td>
<td>0.680</td>
<td>0.575</td>
<td>-0.526</td>
<td>-0.549</td>
<td>0.035</td>
<td>-0.005</td>
</tr>
</tbody>
</table>
ing moves by the number of these moves. The total average scores are the ratio of the sum of all absolute and relative position move scores to the number of moves (i.e., 50).

Table 2 discloses that all average scores for improving moves are positive while all average scores for non-improving moves are negative. Hence a larger percentage of jobs are scheduled in their targeted absolute and/or relative position by means of improving moves than by means of non-improving moves. In fact, negative scores indicate that the non-improving moves have the tendency to move jobs out of their targeted positions (absolute or relative). In problem 2, for example, an improving move results, on the average, in more than one job (1.261) being moved in its targeted absolute position while at the same time a non-improving move results in approximately one job (−0.963) being moved out of its targeted absolute position. These results are highly significant and suggest that the current strategy is quite effective when the admissible move with the highest evaluation is also an improving move, but that a different strategy should be used for selecting moves at points where improving moves are not available.

Step 3

Based on the information generated by target analysis in the preceding steps, this step undertakes to characterize specific evaluation functions that may be used to compose a master decision rule. Such evaluation functions can be classified into two kinds: (i) those that make use of information available only during the current iteration, and (ii) those that make use of past information generated during the search, as a supplement to the type of exploitation of past information that occurs in the short term memory function of tabu search (hence which may include considerations relevant to the use of the intermediate and long term memory functions [4]). For our application we confined attention to the most direct possibilities, choosing one function of the first type and two of the second. Our choices were as follows:

(1) The move value. This function is the one standardly used by the search procedure, which measures the amount of improvement or deterioration that each move, if selected, will cause in the value of the objective function (and which is also used to classify moves as improving and non-improving).

(2) The absolute position record. This function is designed to keep track of the absolute positions occupied by each job in high quality solutions (i.e., solutions whose objective function values are within a certain percentage of the best objective function value found so far). The record is maintained as a matrix of jobs versus positions. The value of a cell (which begins at zero) is incremented each time a job occurs in a given position in a high quality solution, and is also incremented when a job is moved to a given position as a result of an improving move that yields a good solution.

(3) The relative position record. This function is essentially the same as (2) but keeps track of the relative job positions in a matrix of jobs versus jobs.

As previously noted, the evidence summarized in Table 2 indicates that the choice of a best admissible move by criterion (1) is a good strategy during iterations where the set of admissible moves contains at least one improving move. Unfortunately, a similar conclusion cannot be made for points where no improving admissible move is available. The last phase of target analysis undertakes the challenging task of formulating choice rules based on incorporating the other two evaluation functions, which in our case are primarily applicable to handling non-improving situations.

3.3 Phase III

This phase of target analysis is concerned with the creation of a master decision rule. An effective form of such a rule depends on integrating component evaluation functions by means of mathematical and statistical models for determining effective parameters and thresholds [2–4]. Moreover, an additional avenue for determining a good master rule arises by taking advantage of the scoring procedure we have introduced for evaluating moves. Such a procedure makes it possible to obtain information about potential contributions of proposed evaluation functions which can be used to analyze preferred combinations and ranges of application.

To accomplish this, we propose the inclusion of two variants of the absolute and relative position records, in addition to the forms previously described, which have useful properties to supplement those of the original records (2) and (3), and which result by
normalizing the entries of the associated matrices. In particular, for each record a row normalization is created by dividing each row of the matrix by the sum of the entries in that row, and a column normalization is created by dividing each column by the sum of the entries in that column. (Such normalizations may be maintained implicitly, by storing row and column sums, rather than updating separate matrices.)

The row normalization for the absolute position record may be interpreted as creating probabilities for assigning a given job to various positions, while the column normalization may be interpreted as creating corresponding probabilities for assigning a given position to various jobs. Similar interpretations apply to the normalizations of the relative position matrix, involving probabilities relating to the identity of job successors and predecessors. It should be noted that there may be implicit conflicts in the information available from these interpretations, and particularly between the information derived from the absolute position record versus the relative position record (since each implies a preferred structure for the other that may not be fully compatible with the information recorded). These conflicts may be a source of insight about schedule segments that require special attention in the determination of good choice rules.

Both the original and the normalized absolute and relative position records may be used to evaluate moves in a manner analogous to that proposed earlier for scoring moves. Specifically, a given record can be applied to evaluate the solutions obtained before and after applying a move, wherein the associated move evaluation can be obtained from the difference of the two solution evaluations. To evaluate a solution, it is not only necessary to examine each job in the schedule to determine the position it occupies (or the job that follows it), and to consult the corresponding entry of the appropriate matrix record to identify the value to be accumulated as part of the total evaluation. (Computational shortcuts result by restricting attention to the segments of a schedule that are changed by a move.)

There is an important difference in the evaluations produced by the original and normalized records. The original records give relatively higher evaluations to the jobs that are "Less stable", i.e., that appear to have more than one good position (as a result of moves that produced improved solutions by transferring these jobs to such positions). Thus, the use of this type of evaluation will favor moves involving jobs with a less clearly defined preference for a given position, tending to diversify the schedules produced by encouraging continued movement of these jobs. On the other hand, the row normalized record will remove this bias toward unstable jobs, and will give highest values (closest to 1) for jobs that have rarely strayed from a given position. Thus, this latter type of evaluation will more nearly correspond to applying an intensification strategy. (Both the normalized and unnormalized evaluations are based on reinforcing characteristics identified as good in the past, and hence broadly may be viewed as intensification strategies, since a truly aggressive form of diversification strategy seeks to break away from the past. The balance of these two elements is a fertile area of study. See, e.g., [7, 10, 11]).

In a similar manner, column normalization in the absolute position record diminishes the relative influence of unstable positions (i.e., which have been occupied by various jobs). An even greater emphasis on stability might be pursued by records employing both row and column normalizations. (If only solutions, and not moves, were used to provide the entries of these records, then the normalized and unnormalized forms would be equivalent.)

To link these evaluations with the objective function evaluation to create a master decision rule entails an additional application of phase II of target analysis. We briefly note how this may be accomplished, though a full study of such a linking (which essentially results in nesting phase II within phase III) is beyond the scope of our present investigation. In a design similar to that previously executed in our implementation of phase II, the evaluations produced by each of the proposed functions can be recorded for a subset of the moves that rank highly by the objective function criterion, and then compared against the solution scores to determine which evaluations are more effective. These comparisons can also be used to establish thresholds for the various evaluation functions that determine the ranges in which they best apply. Such comparisons may further be used to identify a second type of threshold, which has the goal of determining when a move is "good enough" to warrant acceptance without an extended search for the best (hence providing a basis for a candidate list strategy).

3.4 An Approximating Implementation of Phase III

Given the major investment of research effort required by a full scale implementation of phase III, we have undertaken to use the information provided by target analysis in phases I and II to create two
alternative implementations capable of being pursued feasibly within the scope of this study. The first involves interpreting the normalized absolute and relative position records as "ghost schedules" which can be approximated by means of a single target schedule vector, while the second involves introducing a new aspiration criterion and associated tabu list to exploit elements of the behavior disclosed by the analysis of phase II.

The Ghost Schedule Approximation

In the row normalization of the absolute position rule, each row entry may be viewed as a "ghost position" for the associated job, in the sense that the job generally is neither entirely present nor entirely absent from this position, but rather occupies the position only with a certain probability. Similarly, in the column normalization, each column entry describes a "ghost job" which the associated position contains only with a certain probability. Viewing the normalized matrices, accordingly, as ghost schedules with probabilistic occupancies, we may conceive of a simpler type of evaluation that approximates these matrix records by a (real) target schedule. A logical choice for this target schedule is the best schedule obtained up to the present iteration. Such a schedule incorporates less information than the ghost schedule, since it is simply a vector that records an isolated snapshot of history rather than a matrix that encompasses more comprehensive information from a longer span. However, the target schedule retains a degree of adaptiveness by means of taking a form that gradually changes (as long as improved schedules continue to be found). Consequently, we undertook to investigate this approximation to the use of the relative and absolute position records, in place of a more extensive effort of creating a master decision rule from these records by embedding an additional application of phase II within phase III.

Figure 2. Comparison Between Tabu Search With and Without the Target Schedule Approach
To test the use of the target schedule, we applied it to evaluate moves only in the situation where no admissible improving moves were available, relying on the choice of the highest objective function evaluation in all other cases. (It may be noted that the evaluation of moves provided by the target schedule is identical to that obtained from an absolute position record with a single 1 in each row, which identifies the position occupied by the associated job in the target schedule.) Within this test environment, we re-solved the set of the ten 20-job problems whose optimal solutions are known. Figure 2 shows a comparison of the number of moves required to optimally solve the problems by the original tabu search method and by the tabu search method with the target schedule strategy. The approach incorporating the target schedule succeeds in finding optimal solutions to all problems in less than 175 moves while the original TS procedure requires more than 250 moves to solve them all. In the absence of a more extensive application of target analysis to determine a threshold for isolating high evaluation moves, the target schedule approach in its present form has the disadvantage of requiring (ten from each class) to be identified by a sorting procedure, which diminishes the attractiveness of the approach for larger problems. Motivated by this, and by the desire to exploit other aspects of the solution behavior uncovered by target analysis, we turned to the development of additional strategies.

Use of an Embedded Aspiration Level and a Secondary Tabu List

The information generated during our application of phase II suggests that under certain circumstances it is preferable to disregard the restrictions imposed on certain tabu moves that are also improving moves. In general, the use of aspiration levels in tabu search is for the purpose of handling just such a condition, and thus a possible response is to replace the aspiration level customarily used by a more relaxed (less stringent) aspiration level.

Such a replacement potentially can have both advantages and disadvantages, i.e., it may result in finding an improved solution, but it may also result in going back to a local optimum previously encountered. Returning to a solution already visited is, in our case, not necessarily a bad strategy since it may provide an opportunity to leave this point by a more effective path than taken before. (The use of a relaxed aspiration level is different from the strategy of recording a local optimum with the intent to return to it if the escape direction does not look promising. The revised trajectory created by a relaxed aspiration level does not compel a return to a specific solution, but simply allows such a return if it proves more attractive than all competing alternatives.)

Clearly, visiting a solution more than once makes little sense unless a different sequence of moves is taken leading away from this solution. To assure this in our application we create a secondary tabu list of jobs. This list contains the indexes of the jobs that changed positions (π(t) and π(j)) for a swap move and π(t) only for a transfer move) immediately upon leaving a local optimum, and has a size of $[N/2]$ jobs (where $[x]$ is the largest integer less than or equal to $x$). The purpose of this list is to avoid, at non-improving situations, all moves that will alter the position of any of its member jobs unless a better solution than the one found so far can be obtained.

The secondary tabu list is an event-dependent list that is activated only when nonimproving moves are made, and only in the vicinity of a high quality solution. In our implementation, we have defined high quality solutions as those whose objective function values are not more than 0.1% greater than the objective function value of the best solution found so far. We exploit the fact that these solutions often have a common set of jobs that are scheduled in their targeted positions, to enforce the same secondary tabu restrictions in all of them. In general, displacing jobs in this set, as means of leaving the current high quality local optimum, results in moves with good move values but bad move scores (where the latter are based on knowledge of an attractive target solution). Therefore, the secondary tabu list provides a mechanism to record which jobs should not be moved during the path away from a high quality solution. The aspiration level criterion that we employed in combination with the secondary tabu list allows an improving tabu move to be made if it has been in the tabu list for at least $[\text{tabu}_\text{size}/2]$ iterations and if no admissible improving move is available.

We conducted a computational experiment to compare the performance of our original tabu search method and the version incorporating the secondary tabu list. A set of 5 problems for each dimension 40, 60, 80, and 100 were randomly generated for this experiment (using the distributions and parameters in Section 3.1). Two solution attempts of 1000 iterations were made with each method, using heuristics 1 and 2 to provide the two initial solutions, and tabu_size was set to 7. (An improved tabu_size for
Table 3. Summary of Computational Experience.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Before TA</th>
<th>After TA</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Worst</td>
<td>Best</td>
</tr>
<tr>
<td>40-Job</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>193,153</td>
<td>193,337</td>
<td>193,061</td>
</tr>
<tr>
<td>2</td>
<td>210,218</td>
<td>210,218</td>
<td>210,294</td>
</tr>
<tr>
<td>3</td>
<td>218,086</td>
<td>218,128</td>
<td>218,128</td>
</tr>
<tr>
<td>4</td>
<td>263,605</td>
<td>263,706</td>
<td>263,436</td>
</tr>
<tr>
<td>5</td>
<td>219,621</td>
<td>219,648</td>
<td>219,537</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60-Job</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>494,659</td>
<td>495,195</td>
<td>493,877</td>
</tr>
<tr>
<td>2</td>
<td>571,289</td>
<td>571,358</td>
<td>571,573</td>
</tr>
<tr>
<td>3</td>
<td>420,197</td>
<td>420,297</td>
<td>420,146</td>
</tr>
<tr>
<td>4</td>
<td>508,737</td>
<td>509,410</td>
<td>508,682</td>
</tr>
<tr>
<td>5</td>
<td>561,227</td>
<td>560,921</td>
<td>560,916</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80-Job</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>869,692</td>
<td>869,165</td>
<td>868,833</td>
</tr>
<tr>
<td>2</td>
<td>854,439</td>
<td>854,799</td>
<td>854,472</td>
</tr>
<tr>
<td>3</td>
<td>791,591</td>
<td>792,418</td>
<td>791,838</td>
</tr>
<tr>
<td>4</td>
<td>835,783</td>
<td>838,650</td>
<td>835,369</td>
</tr>
<tr>
<td>5</td>
<td>977,948</td>
<td>978,265</td>
<td>977,953</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100-Job</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1,516,995</td>
<td>1,518,445</td>
<td>1,516,863</td>
</tr>
<tr>
<td>2</td>
<td>1,470,531</td>
<td>1,471,651</td>
<td>1,470,432</td>
</tr>
<tr>
<td>3</td>
<td>1,346,453</td>
<td>1,349,847</td>
<td>1,346,824</td>
</tr>
<tr>
<td>4</td>
<td>1,212,163</td>
<td>1,214,125</td>
<td>1,211,716</td>
</tr>
<tr>
<td>5</td>
<td>1,233,147</td>
<td>1,233,345</td>
<td>1,232,164</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

these problems can likely be found by more extensive experimentation.)

Table 3 presents the results of our experiment by showing the best solution to each problem found by each method and the difference between these solutions. A positive difference indicates an improvement in the objective function as a result of using the new approach and a negative difference indicates the reverse. For each set of problems a total difference is computed by adding all individual differences. Table 3 shows that 13 of the 20 best solutions for each problem were found by the tabu search method incorporating the secondary tabu list. Also, 15 of the 20 overall worst solutions are attributable to the original tabu search method. The total best solution differences for each set of problems are also in favor of the tabu search method that employs a secondary tabu list. In sum, the approach that attempts to take advantage of the information provided by target analysis, although constituting an indirect strategy for doing this, yields a more consistent solution method for its ability to provide high quality solutions to P1.

4. CONCLUSIONS

Our research suggests the usefulness of target analysis for developing improved variants of tabu
search. Although we relied on approximations to the types of strategies that may be developed by a more comprehensive application of target analysis, the outcomes of employing these approximations were encouraging. In addition, the nature of the information generated by our application of target analysis was suggestive of more general patterns to be watched for in future investigations. Specifically, the results from our application of phase II disclosed the importance of looking for regional dependence of good decision criteria, which can be characterized in a more refined way than we have undertaken. For example, instead of establishing a dividing line between improving and non-improving moves (according to a particular decision criterion), there may be two or three major dividing lines, each identifying a region that may be exploited most effectively by a particular (individually tailored) rule. A process of successively identifying regions of competence for different decision rules, and using these to characterize new decision rules and new regions, offers an inviting avenue for investigation.

Finally, we note that our experimentation sets the stage for more ambitious implementations of target analysis that embed additional applications of phase II within phase III. Such an approach appears crucial to determining effective uses of evaluation functions such as the relative and absolute position records we have described. Moreover, the generation of information relevant to combining these functions into a master decision rule provides a foundation for mathematical and statistical models to identify effective parameter values (as in [4]), allowing the possibility of additional gains.

ACKNOWLEDGEMENT

This research has been supported in part by the Air Force Office of Scientific Research under contract AFOSR 1538101 with the University of Colorado, Boulder.

REFERENCES


Paper Received 17 February 1990; Revised 4 June 1990.