INFORMS Journal on Computing
Vol. 16, No. 3, Summer 2004, pp. 241–254
ISSN 0899-1499 | EISSN 1526-5528 | 04 | 1603 | 0241

DOI 10.1287/ijoc.1030.0040
© 2004 INFORMS

Exploiting Tabu Search Memory in Constrained Problems

Sadan Kulturel-Konak
Management Information Systems, Penn State Berks-Lehigh Valley College, Reading, Pennsylvania 19610, USA, sadan@psu.edu

Bryan A. Norman
Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania 15261, USA, banorman@engr.pitt.edu

David W. Coit
Department of Industrial and Systems Engineering, Rutgers University, Piscataway, New Jersey 08854, USA, coit@rci.rutgers.edu

Alice E. Smith
Department of Industrial and Systems Engineering, Auburn University, Auburn, Alabama 36849, USA, aesmith@eng.auburn.edu

This paper puts forth a general method to optimize constrained problems effectively when using tabu search. An adaptive penalty approach is used that exploits the short-term memory structure of the tabu list along with the long-term memory of the search results. It is shown to be effective on a variety of combinatorial problems with different degrees and numbers of constraints. The approach requires few parameters, is robust to their setting, and encourages search in promising regions of the feasible and infeasible regions before converging to a final feasible solution. The method is tested on three diverse NP-hard problems, facility layout, system reliability optimization, and orienteering, and is compared with two other penalty approaches developed explicitly for tabu search. The proposed memory-based approach shows consistent strong performance.

Key words: constraint optimization; facility layout; reliability optimization; orienteering; tabu search
History: Accepted by Michel Gendreau; submitted March 2000; revised November 2001, March 2003; accepted April 2003.

1. Introduction

Tabu search (TS) has become an effective heuristic method for many combinatorial optimization problems with large and complex search spaces. Glover and Laguna (1997) define the most important distinguishing property of TS as the exploitation of adaptive forms of memory. These take the form of short-term memory strategies (i.e., tabu list and aspiration criteria) and long-term memory strategies (e.g., intensification and diversification). This paper develops a general approach that uses the special memory properties of TS to optimize constrained problems effectively. An adaptive penalty function, which responds to the search history to guide the search to promising regions of both the feasible and infeasible regions of the space, is developed and tested. While TS has been applied to many constrained problems, a general penalty methodology that specifically exploits the memory structure of TS has not been previously presented in the literature.

Although its roots go back to the late 1960s and early 1970s, TS was proposed in its current form in the late 1980s by Glover (1986). Together with simulated annealing (SA) and genetic algorithms (GA), TS has been chosen by the Committee on the Next Decade of Operations Research (1988) as "extremely promising" for the future treatment of practical applications. Although it is difficult to represent a "canonical form" of TS, most versions of TS can be characterized by the following two properties: (i) complementing local search, and (ii) prohibiting moves that have been previously selected through use of a tabu list. For this second aspect, there are alternative tabu-list management approaches in which certain types of cyclic behavior are excluded. Some methods determine a tabu status based on sequential relationships between the selected moves, rather than the structure of the move itself. Examples from this group are the *cancellation sequence method* (Dammeyer et al. 1991) and the *reverse elimination method* (Dammeyer and Voss 1993, Glover 1990). Further information about TS is available in Glover and Laguna (1997), Glover (1989, 1990), and Glover et al. (1993).

2. Constraints and Tabu Search

Most optimization problems contain constraints. Some of these are easy to satisfy through problem-specific encodings and operators so that infeasible solutions are not considered. However, it is often difficult to enforce solution feasibility. Moreover, it may be hard to identify even a single feasible solution for some highly constrained problems. Even for problems where feasibility can be maintained by discarding infeasible solutions without consideration, it may not be efficient or effective to do so (Coit and Smith 1996a). General heuristics, such as TS, SA, and GA are especially problematic when dealing with such problems because they initiate search (generally) with a random solution and apply operators that may not be able to guarantee feasibility even when operating on a feasible solution. Furthermore, because these are general approaches (i.e., meta-heuristics), it is desirable to employ a general method for dealing with constraints that minimizes or eliminates the need for problem-specific information.

Much of the literature on handling constraints involves the use of penalty functions. Rather than enforcing feasibility, a penalty is applied to an infeasible solution to worsen its objective function value. This may be as simple as a constant penalty for any infeasible solution to more complex functions that depend on the solution itself, the search history, or other user-defined criteria. Schwefel (1995) classified penalty functions according to their severity as follows:

- barrier methods where no infeasible solution is considered (also known as the death penalty);
- partial penalty functions where a penalty is applied near the feasibility boundary;
- global penalty functions where a penalty is applied over the entire infeasible region.

Lagrangian relaxation methods (Avriel 1976, Fisher 1981, Reeves 1993) use a somewhat similar approach where the most difficult constraints of the problem are temporarily relaxed by using a modified objective function to prevent a solution from being too far from the feasible region.

A general penalty approach is described below, using the notation of Smith and Coit (1995). A minimization problem is shown, but this can be readily converted to a maximization problem.

$$\begin{aligned} \min\ & f(\mathbf{x}) \\ \text{s.t. } & \mathbf{x} \in A \\ & \mathbf{x} \in B, \end{aligned}$$

where $\mathbf{x}$ is a vector of decision variables, and the constraints "$\mathbf{x} \in A$" are relatively easy to satisfy while the constraints "$\mathbf{x} \in B$" are relatively difficult to satisfy. Using this definition, the problem can be reformulated as:

$$\begin{aligned} \min\ & f(\mathbf{x}) + p(d(\mathbf{x}, B)) \\ \text{s.t. } & \mathbf{x} \in A, \end{aligned}$$

where $d(\mathbf{x}, B)$ is a function denoting the distance of the solution vector $\mathbf{x}$ from the region B, and $p(\cdot)$ is a monotonically nondecreasing penalty function. Two major penalty approaches have been studied in the literature; one based on the number of constraints violated, and one based on the distance from the feasible region, with the latter usually being more effective (Joines and Houck 1994).

Hertz (1992) solves a course-scheduling problem using TS by weighting constraints according to their difficulty to satisfy, where constraints that are more difficult have greater weight. Weights are predetermined by the user and are customized to each problem instance. Taillard et al. (1997) study the vehicle routing problem (VRP) using TS and, in their problem, each vehicle must start and terminate its route within the time window associated with the depot. It is permissible to miss the time windows at some customer locations, but this results in penalties that are added to the objective value. Their objective function, therefore, consists of the total distance traveled on the route plus the amount of lateness multiplied by a lateness-penalty coefficient (α). They assert that by using a large α, the VRP with hard time windows can also be addressed.

Thomas and Salhi (1998) implement a TS heuristic to solve the resource-constrained project-scheduling problem where the objective function is the minimization of project makespan. Their objective function has the property of rejecting moves that have been visited many times previously, and penalizing moves that lead to schedules possessing a high level of resource infeasibility. The penalty function, which is a multiple of the ratio of resource infeasibility to the total amount of that resource available, is added to the objective value.

Glover et al. (1993) describe a "shifting penalty approach" that is an instance of strategic oscillation, one of the basic diversification approaches for TS. The shifting-penalty tactic is used by Hertz and de Werra (1989) and Costa (1990) for timetable-planning problems. First, a penalty function is constructed based on the degree of violation for any given schedule. Then, the global objective function is defined as the weighted sum of these penalty functions, where constraints are weighted according to importance. Weights are dynamic, and the largest weights are reduced after a specified number of iterations. With this method, the more important constraints are satisfied early in the search, then the search turns to satisfying the lesser constraints. A different tact is taken by Scholl and Voss (1996) where the infeasibility aspect

of an assembly-line-balancing problem is handled by beginning with an initial number of stations that is infeasible, and increasing this by one until a feasible solution is found. TS is used at each iteration to search for solutions to the resulting problems.

2.1. The Penalty Function of Nonobe and Ibaraki

A TS approach to the constraint-satisfaction problem (CSP) is studied by Nonobe and Ibaraki (1998) (N&I). They define a weight for each constraint based on its "importance." If the CSP has no feasible solution, an acceptable solution that slightly violates the less important inequalities is identified. They give computational results for problems including graph coloring, generalized assignment, set covering, timetabling, and nurse scheduling.

Using the notation from N&I, starting with an initial solution x, a penalty function, $p(x)$ is defined by

$$p(x) = \sum_{h=1}^{m} w_h p_h(x),$$

where w_h is a weight given to each constraint and

$$p_h(x) = \begin{cases} \max(\sum_{i,j} a_{hij}x_{ij} - b_h,\ 0) \\ \quad \text{if the } h\text{th constraint is } \sum_{i,j} a_{hij}x_{ij} \le b_h, \\ \text{the amount of violation (nonnegative number)} \\ \quad \text{if the } h\text{th constraint is defined differently.} \end{cases}$$

Combining the problem objective function, $f(x)$, with the penalty results in the overall objective function q (for a minimization problem):

$$\min q(x) = w_0\{\max(f(x) - z, 0) + \theta \min(f(x) - z, 0)\} + p(x),$$

where z is the objective-function value, f, of the best feasible solution found thus far (initially set to a large number), $0 \le \theta \le 1$ is a program parameter, and $w_0 > 0$ is a weight given to $f(x)$. This function encourages search near the border of feasibility and infeasibility. If a new solution x is better than the best feasible but is also infeasible, its objective-function improvement $f(x) - z$ is weighted less than its corresponding penalty $p(x)$ by using $\theta < 1$ and $w_0 < 1$. There is concern in setting the parameters. If w_0 is large, solutions with better objective values are readily found, but it may be difficult to obtain feasibility. On the other hand, if it is small, feasibility is generally maintained but the objective-function value may suffer. Therefore, N&I introduce an adaptive method to control w_0 to encourage feasible solutions within a prespecified range of lower and upper bounds, LB and UB, respectively. If the rate of infeasible solutions in the most recent 100 iterations is less than LB, w_0 is multiplied by σ. If this rate is greater than UB, then w_0 is divided by σ, where $\sigma > 1$ is a program parameter. From preliminary trials, they found that performance is not very sensitive to σ, however LB and UB are important, so their values must be carefully set. The LB and UB values can be set according to the difficulty of obtaining feasible solutions during the search, i.e., large values should be assigned if it is hard to find feasible solutions and small values if it is rather easy. In their paper, the parameter values are set as follows: $\theta = 0.5$, $\sigma = 3$, $w_0^{(0)} = 1$ (initial value of w_0), LB $= 0.4$, and UB $= 0.6$.

2.2. The Penalty Function of Gendreau, Hertz, and Laporte

Gendreau et al. (1994) (GHL) solve the VRP using TS with capacity and route-duration constraints. They develop a penalty approach that depends on recent constraint violations over a last predefined number of solutions. Using the notation of GHL, the objective is to find a set of shortest (least cost) vehicle routes.

$$\min F_1(S) = \sum_r \sum_{(v_i, v_j) \in R_r} c_{ij},$$

where S is a solution, r are the routes in the set, R_r is a specific route in the set, v_i and v_j are two consecutive vertices (i.e., two different cities), and c_{ij} is a nonnegative distance (cost) associated with the arc(v_i, v_j). With any solution S (feasible or not), they associate the objective in the following penalized form:

$$\min F_2(S) = F_1(S) + \beta_1(\max(\text{capacity violation}, 0)) + \beta_2(\max(\text{duration violation}, 0)).$$

If the solution is infeasible, $F_2(S)$ incorporates a penalty term for excess vehicle capacity and another for excess route duration. Positive weights β_1 and β_2 are initially set to one and then updated after each h iterations as follows. A weight for a constraint that was *always* violated over the past h iterations is doubled. A weight for a constraint that was *never* violated over the past h iterations is halved. Otherwise, weights remain unchanged. This has the property of inflating (deflating) the penalty imposed if the recent search history is entirely within the infeasible (feasible) region. They use a value of 10 for h although their experimental results show that the search is not sensitive to this value.

3. Proposed Adaptive Penalty Method

This paper builds on the approaches discussed above, especially those of N&I and GHL, by penalizing infeasible solutions according to the distance from the

feasible region and the search history, as remembered by the tabu list, in short-term memory, and by the best solutions found in long-term memory. The penalty is implicitly bounded and can be influenced through an amplification parameter for each constraint. This approach is demonstrated on three dissimilar combinatorial problems: the unequal-area, shape-constrained block layout problem, the series-parallel redundancy allocation problem, and the orienteering problem, a constrained version of the traveling salesman problem. In the first and last problem types, there is a single, discrete constraint while in the second problem type, there are multiple, continuous constraints. A variety of instances with different degrees of constraint are solved using the memory-based penalty TS for both classes of problems, and compared with the penalty approaches of N&I and GHL.

3.1. Basic Structure of the Penalty

The proposed penalty function uses the central idea of near feasibility threshold (NFT) as first defined by Smith and Tate (1993) and enhanced by Coit et al. (1996) in their work on penalty functions for GA. NFT marks the portion of the infeasible region where search should be encouraged. While solutions are penalized in relation to their distance from feasibility, within the NFT region (i.e., between feasibility and the NFT), infeasible solutions are penalized relatively lightly, while beyond the NFT region solutions are penalized relatively heavily.

The definition of NFT depends on both the structure of the problem and that of the constraints. While NFT can be as simple as a constant (Tate and Smith 1995), it is often effective to employ a dynamic or adaptive NFT (Coit and Smith 1996a) that reacts in response to the search history. An obvious dynamic formulation is to adjust NFT with the length of the search, so that NFT continually decreases, which increases the penalty imposed, all else being equal. A dynamic decreasing NFT will encourage search through the infeasible region early, but then increasingly encourage focus in the feasible region. This was done in Coit et al. (1996) as:

$$\mathrm{NFT} = \frac{\mathrm{NFT}_0}{1+\Lambda},$$

where NFT_0 is an initial value for NFT and Λ is a variable to adjust NFT. For example, in a GA Λ can be defined as a function of the number of GA generations, g, by letting $\Lambda = \lambda \times g$, where λ is a user-defined constant.

Hence, the general penalized objective function is in the following form (for a minimization problem) with η constraints.

$$F_p(\mathbf{x}) = F(\mathbf{x}) + (F_{feas} - F_{all}) \times \sum_{i=1}^{\eta} \left(\frac{d_i(\mathbf{x}, B)}{\mathrm{NFT}_i} \right)^{\kappa_i} \qquad (1)$$

where $F(\mathbf{x})$ and $F_p(\mathbf{x})$ are the unpenalized and penalized objective-function values, respectively, for solution $\mathbf{x}$. F_{all} represents the unpenalized objective-function value of the best solution found so far, and F_{feas} is the value of the best feasible solution found so far. Exponent κ_i is a user-defined parameter that amplifies the behavior of the ratio, and NFT_i is the near-feasibility threshold for constraint i. The penalty imposed above depends on both the distance of the solution from feasibility and the search history regarding the relative difference between the best feasible and infeasible solutions found.

The penalty function above has several nice properties. It is adaptive and automatically scales itself to the particular constraint through the ratio $d_i(\mathbf{x}, B)/\mathrm{NFT}_i$. It has been shown to be robust to κ_i, to problem instance, degree of problem constraint, and number and type of constraints when used in a GA (Coit et al. 1996). There are few user-set parameters. The central idea of this paper is to include these advantageous properties in a method explicitly designed for TS that leverages its memory properties.

3.2. Incorporating Memory into NFT

NFT represents the area in the infeasible region where search is encouraged. In the GA implementation (Coit et al. 1996, Tate and Smith 1995), NFT ranged from a constant to a variable depending on the number of generations. NFT can assume a more sophisticated role in TS and there is greater potential for improved efficiency in solving constrained optimization problems by exploiting search-history information.

In this paper, NFT adapts to the recent search history by using the short-term memory capability of the tabu list along with knowledge of the current move. If most of the recent moves have been feasible, NFT is increased, thereby decreasing the penalty and encouraging more exploration of the near-feasible region. If most of the recently accepted moves have been infeasible, NFT is decreased, increasing the penalty and moving the search towards the feasible region. Unlike the GA dynamic implementation where NFT monotonically decreases with search length, the TS NFT can increase or decrease depending on short-term memory. This allows the magnitude of the penalty imposed on infeasible solutions to shrink or grow according to the recent search results. An analogous technique in simulated annealing was used by Osman and Christofides (1994) where reheating was invoked according to the search results. This eliminated the monotonic decrease of temperature in SA just as the memory-based NFT described here eliminates the monotonic increase of the penalty. Using the memory-based NFT infeasible solutions are penalized according to (1). A long-term memory term, $F_{feas} - F_{all}$,

which is the difference between the best feasible solution and the unpenalized value of the best solution yet found in the search, is included in (1).

Specifically, the method is as follows. The tabu list size at any given iteration j is defined as T_j, and the number of feasible solutions on the current tabu list is defined as F_j. A feasibility ratio at iteration j, R_j, is defined as

$$R_j = \frac{F_j}{T_j}. \tag{2}$$

For constraint i, if the current move is to a feasible solution,

$$\mathrm{NFT}_{i,j+1} = \mathrm{NFT}_{i,j}\left(1 + \frac{R_j}{2}\right). \tag{3}$$

For constraint i, if the current move is to an infeasible solution,

$$\mathrm{NFT}_{i,j+1} = \mathrm{NFT}_{i,j}\,\frac{1 + R_j}{2}. \tag{4}$$

For a given constraint, NFT changes according to the count of the feasible versus infeasible solutions on the tabu list. The N&I method includes the idea of examining recent solutions to change the weight of the penalty. The GHL method uses the idea of altering the penalty according to the feasibility of solutions on the tabu list. In their case, it is a step function that changes when the tabu list has moved from wholly infeasible to wholly feasible, or vice versa. The method of (1) through (4) uses a continuous metric for the feasibility/infeasibility constituency of the tabu list, and additionally considers the feasibility of the current move. Long-term memory is also employed with the difference term between the best feasible solution yet found and the unpenalized value of the best solution yet found.

Consider the behavior of NFT for a single bounding constraint. If all moves on the tabu list are feasible and the current move is also feasible, NFT increases by a factor of 1.5. This has the property of encouraging search towards the infeasible region. If all moves on the tabu list are infeasible and the current move is also infeasible, NFT decreases by a factor of 0.5. This increases the penalty for an infeasible solution and moves the search towards the feasible region. This geometric change in NFT creates a lower bound on NFT of 0.

If all moves on the tabu list are feasible and the current move is infeasible, NFT remains unchanged. Similarly, if all moves on the tabu list are infeasible and the current move is feasible, NFT remains unchanged. In these cases, the value of NFT is appropriate as it has moved the search towards the recently unvisited region, either feasible or infeasible. In the next move, if the same region as the last move is chosen, NFT is slightly increased (in the case of recent feasible moves) or slightly decreased (in the case of recent infeasible moves).

An initial value of NFT needs to be set for each constraint, although the method is insensitive to this value since NFT will begin changing immediately. One simple way to do this is to take a percent of each constraint as its NFT_0. Note that this memory-based NFT does not correlate penalty with search iteration as does the version of Coit et al. (1996). Using the memory-based NFT results in a penalty that may increase, decrease, or stay the same as the search progresses, thus negating the need to consider reheating strategies or other nonmonotonic behavior-inducing devices. This formulation can easily handle dynamic tabu list sizes by using the current size, T_j, in (3) and (4). Multiple constraints are handled independently, and constraints that are discrete or continuous can be accommodated.

4. Demonstration Applications

The proposed penalty function and the other two general methods developed for TS, N&I, and GHL, are applied to three diverse NP-hard combinatorial problems: facility layout, system-reliability optimization, and orienteering. While the results for each problem class are only shown for the TS with each penalty approach, the best TS solutions equal or improve upon the best solutions found by other heuristics, as published in the literature. These are the GA of Coit et al. (1996) for the layout problems, the GA of Coit and Smith (1996b) for the redundancy-allocation problems, and the problem-specific improvement heuristic of Chao et al. (1996a) for the orienteering problems. The TS procedure and parameters used in the proposed penalty function will be described in the following subsection, while the parameter values used in the other penalty functions are shown in Table 1. All algorithms were coded in C and run using a Sun Ultra Enterprise-2 workstation with a 200 MHz Sparc dual processor and 128 MB of RAM.

4.1. Unequal-Area Block Layout

The unequal-area block-layout problem, from facilities design, was originally formalized by Armour and Buffa (1963). There is a rectangular region, R, with fixed dimensions H and W, and a collection of N departments, each of specified area a_j, the total

Table 1 Parameter Settings of Two Penalty Methods

N&I (1998)	
θ	0.5
σ	3.0
$w_0^{(0)}$	1.0
LB	0.4
UB	0.6
GHL (1994)	
Initial weights	1.0
h	10.0

area of which is $A = H \times W$. Each ordered pair of departments (j, k) is associated with a traffic flow $F(j, k)$. The objective is to partition the region into N subregions, of appropriate area, in order to minimize the sum

$$\sum_{j=1}^{N} \sum_{\substack{k=1 \\ (j \neq k)}}^{N} F(j,k) \times \delta(j,k,\Pi),$$

where $\delta(j, k, \Pi)$ is the distance (using a prespecified metric) between the center of department j and the center of department k in the partition Π. This formulation did not include a shape constraint and could result in unrealistically shaped departments through optimization of its centroid-to-centroid distance metric. Tate and Smith (1995) extend the problem by adding a maximum aspect ratio (MAR) for all departments to constrain shapes. Similarly, Coit et al. (1996) establish a minimum-side-length (MSL) constraint for each department.

4.1.1. The TS Algorithm. The TS approach in this paper uses the flexible-bay construct of Tate and Smith (1995) with a variable-length string encoding that concatenates a permutation of the department order (using a boustrophedon ordering) and the bay break position. Figure 1 illustrates this encoding. In the tabu list, an entire solution is kept. A dynamic-length tabu list is used that varies every 20 iterations according to a uniform random number, $U[8, 15]$. The termination criterion is a fixed number of moves without improvement in the objective function of the best feasible solution. The TS reported here is not sensitive to the exact tabu list size or the termination criterion.

The search proceeds as follows (using a six-department illustration):

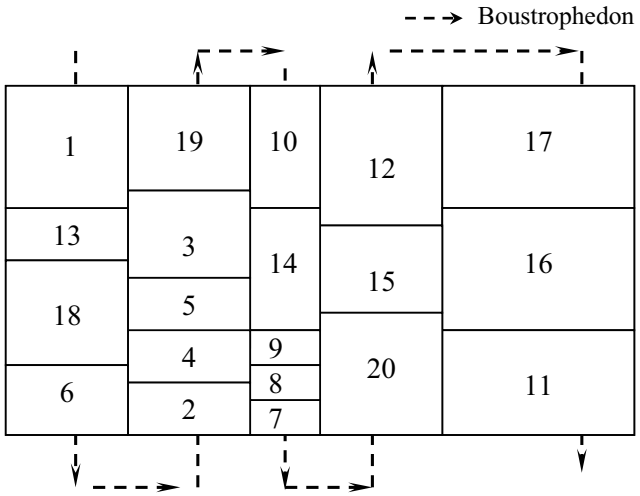

Figure 1 The Facility Layout Encoding

Step 0. Generate a random initial solution. Assign it to the BEST SO FAR, CURRENT CANDIDATE, and the BEST CANDIDATE. (At the beginning of the search, the initial values of the cost of the BEST FEASIBLE SO FAR and the BEST SO FAR are assigned the value of $2 \times H \times W \times (\sum_{j=1}^{N} \sum_{k=1,(j \neq k)}^{N} F(j,k))$, the maximum value of a solution.)

EXAMPLE: 2 6 3 4 1 5 | 3 5.

This layout has departments 2, 6, and 3 in the first bay, departments 4 and 1 in the second bay, and department 5 in the third bay.

Step 1. Search the neighborhood of all possible insertion moves for the department permutation and choose the first one with a better objective-function value. An insertion removes one department from the sequence and inserts it in another location. This is accomplished by generating a random permutation and considering the departments in order of that random permutation. If this candidate solution, the CURRENT CANDIDATE, is tabu and the corresponding solution is not better than the BEST SO FAR solution (i.e., the aspiration criterion is not satisfied), disallow it and repeat Step 1. If the solution is not tabu, this solution becomes the BEST CANDIDATE. Compare it with the BEST SO FAR and THE BEST FEASIBLE SO FAR and make the necessary updates. If no better department sequence can be found, maintain the current solution.

For example, to decide in which order departments will be inserted, a randomly generated permutation is used. However, insertions will be made on the sequence of the original solution. Insertions will be tried until an improvement is found. If the randomly generated permutation is

1 3 5 6 2 4,

then department 1 would be removed from its position and inserted as the first department, then after department 2, then after department 6, and so on. Next, department 3 would be removed and inserted as the first department, then after department 2, then after department 4, and so on. This would continue until an improvement is found. For example, if the first improving move for the previous solution was to insert 3 after 5 then the new solution would be

2 6 4 1 5 3 | 2 4.

Now, departments 2 and 6 are in the first bay, departments 4 and 1 are in the second bay, and departments 5 and 3 are in the third bay.

Step 2. For the departmental sequence of the BEST CANDIDATE found in Step 1, examine new bay breaks by exhaustively adding/subtracting one in all possible locations from the current bay breaks. If the CURRENT CANDIDATE is tabu and the corresponding

solution is not better than the BEST SO FAR solution (i.e., the aspiration criterion is not satisfied), disallow it and repeat Step 2. If the solution is not tabu, this solution is the BEST CANDIDATE. Compare it with the BEST SO FAR and the BEST FEASIBLE SO FAR and make the necessary updates.

For example, there are two bay breaks in the current solution, so all possible deletions of a bay break and all possible additions of a bay break will be tried and the best will be selected, for example

$$2\ 6\ 4\ 1\ 5\ 3\ |\ 2\ 4\ 5.$$

In this solution, departments 2 and 6 are in the first bay, departments 1 and 4 are in the second bay, department 5 is in the third bay, and department 3 is in the fourth bay. Two bay break additions, 1 2 4 5 and 2 3 4 5, would be examined and three single bay deletions, 4 5, 2 5, and 2 4, would be examined and the bay break configuration would be set to the best of these five options and the current bay break configuration.

Step 3. Enter the solution selected by Steps 1 and 2 on the tabu list, also deleting the oldest tabu list entry if the tabu list is full. Check the stopping criterion, and if it is not satisfied, return to Step 1.

4.1.2. Penalty Function. NFT is defined as in Coit et al. (1996), using a metric of the number of infeasible departments rather than the degree of infeasibility of any certain department. There is a single constraint so $\kappa_i = \kappa$. Hence, following (1), the objective function is:

$$F_p(\mathbf{x}) = F(\mathbf{x}) + (F_{feas} - F_{all})\left(\frac{n}{\mathrm{NFT}}\right)^{\kappa},$$

where n is the number of infeasible departments (violating MSL or MAR), κ is set to 2, NFT is calculated as described in §3.2, and NFT_0 is set to 1.

4.1.3. Test Problems and Results. Three problems from the literature were studied. The largest problem was originally defined by Armour and Buffa (1963) and modified by Coit et al. (1996) to set an MSL for each department, and these constraints are used in this study. Coit et al. (1996) used this set of constraints to generate 100,000 solutions randomly and found only 1.6% of them to be feasible. A second, more constrained version, of Armour and Buffa was devised that specified an MAR of 3 for each department. Smaller test problems (12 and 14 departments) are from Bazaraa (1975) and an MSL of 1 was used. The stopping criterion was defined as the maximum number of iterations (Steps 1 and 2) that could be conducted without finding an improvement in the best feasible solution. The number of moves without improvement to the best feasible solution was set at 1,000 (Armour and Buffa problem) and 500 (Bazaraa problems).

Table 2 Results for the Facility-Design Problem

	Armour and Buffa (1963)		Bazaraa-12 dept. (1975)	Bazaraa-14 dept. (1975)
Method	MAR = 3	MSL of Coit et al. (1996)	MSL = 1	MSL = 1
Memory-based				
Best	5,618.2	**4,716.2**	**8,630.0**	**5,190.1**
Mean	**5,852.3**	5,177.1	**8,910.6**	**5,365.0**
Worst	**6,081.6**	**5,533.0**	**9,651.1**	**5,687.3**
Average CPU (sec)	*249*	*205*	*60*	*36*
N&I (1998)				
Best	5,648.5	4,967.7	8,988.833	5,292.478
Mean	5,934.8	5,236.8	9,242.026	5,472.389
Worst	6,252.5	5,588.3	9,725.727	5,751.364
Average CPU (sec)	*200*	*207*	*63*	*25*
GHL (1994)				
Best	**5,607.2**	4,753.5	8,587.0	5,244.2
Mean	5,918.6	**5,049.3**	8,930.7	5,448.4
Worst	7,082.0	5,549.8	9,768.2	5,810.9
Average CPU (sec)	*268*	*219*	*80*	*51*

All three penalty methods were compared over ten random trials. As can be seen from Table 2, where the best result of each row is set in bold, the performance of the TS with a memory-based penalty function is promising. Clearly, the penalty approach of N&I was not as effective as the methods of GHL and this paper. In addition, although it is not shown in Table 2, a few trials did not find a feasible solution with the penalty approach of N&I. The GHL approach generally had greater variability to seed, so that the best results tended to be better; however, the mean and especially the worst-case performance suffered. Average CPU seconds of ten trials are also given in Table 2. The GHL approach and the proposed memory-based approach are not significantly different in terms of CPU seconds. In Figure 2, the behavior

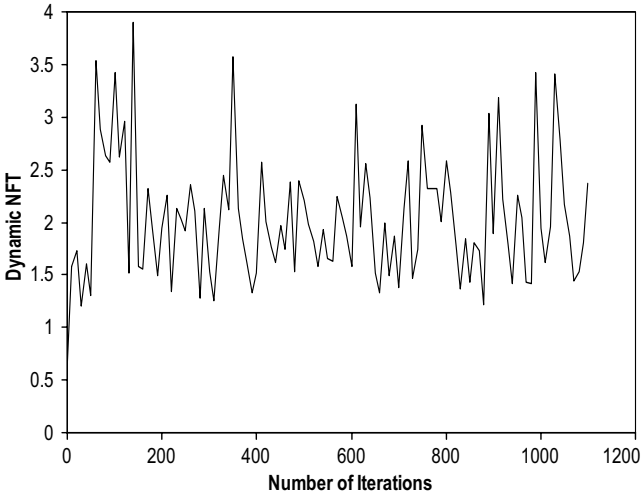

Figure 2 Typical NFT Behavior Over Search for the Armour and Buffa (1963) Block-Layout Problem

Table 3 Comparing the Most Constrained Versions of the Facility-Design Problem

	Armour and Buffa (1963)	
Method	MAR = 2	MAR = 1.70667
Memory-Based		
Best	**6,108.4**	**7,376.4**
Mean	**6,776.9**	—
Worst	**7,444.0**	—
Average CPU (sec)	*218*	*231*
GHL (1994)		
Best	6,342.2	**7,376.4**
Mean	7,197.8	—
Worst	8,496.4	—
Average CPU (sec)	*205*	*233*

Note. Dash denotes only one solution found for MAR = 1.70667.

of NFT over a typical search is shown. While the GA approach (Coit et al. 1996) used a static NFT of 1, it can be seen that the memory property of the TS finds NFTs around 2 to be more effective. NFT alters frequently, generally ranging from 1.5 to 3.5. This NFT activity is desirable in that it confirms that search is being focused near the boundary between feasibility and infeasibility, where the optimal solution is likely to be found.

Although the Armour and Buffa problem with a MAR = 3 is a very constrained problem with 0.0015% feasible solutions per 1,000,000 randomly generated solutions, the performance of the methods on even more constrained problems needed to be gauged. Two smaller MARs, 2 and 1.70667, were used. (Tate and Smith 1995 found 1.70667 when the objective function was changed to search for minimum feasible aspect ratio.) For these values, 1,000,000 randomly generated solutions included no feasible solutions. Comparing GHL with the memory-based approach is shown in Table 3. While both took similar computational effort, the memory-based approach yielded superior results for MAR = 2 and identical results for the most constrained version. In fact, there may be only a single unique design that adheres to this most stringent level of constraint, evidenced by all searches returning the same feasible result.

4.2. Series-Parallel System Redundancy Allocation

The series-parallel system redundancy allocation problem (RAP), Figure 3, is described as follows: given overall restrictions on maximum system cost of C and system weight of W, determine the optimal design configuration to maximize system reliability, when there are multiple component choices available for each of several k-out-of-n:G subsystems. Formally:

Problem P1. Reliability maximization:

$$\max \quad R = \prod_{i=1}^{s} R_i(\mathbf{x}_i \mid k_i)$$

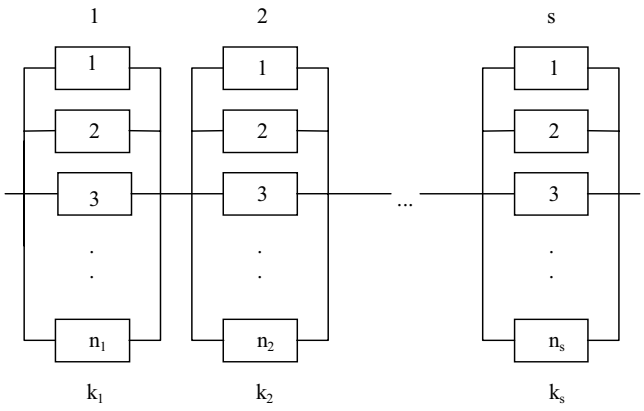

Figure 3 Series-Parallel System Configuration

$$\text{subject to} \quad \sum_{i=1}^{s} C_i(\mathbf{x}_i) \le C$$

$$\sum_{i=1}^{s} W_i(\mathbf{x}_i) \le W$$

$$k_i \le \sum_{j=1}^{m_i} x_{ij} \le n_{\max,\,i}$$

$$\forall i = 1, 2, \ldots, s,\ x_{ij} \in \{0, 1, 2, \ldots\},$$

where,

R: system reliability
C: cost constraint
W: weight constraint
s: number of subsystems
$\mathbf{x}_i$: $(x_{i1}, x_{i2}, \ldots, x_{i,\,m_i})$
x_{ij}: quantity of jth component in subsystem i
n_i: total number of components used in subsystem i, i.e., $\sum_{j=1}^{m_i} x_{ij}$
$n_{\max,\,i}$: maximum number of components in parallel used in subsystem i (user-assigned)
k_i: minimum number of components in parallel required for subsystem i to function
$R_i(\mathbf{x}_i \mid k_i)$: reliability of subsystem i, given k_i
$C_i(\mathbf{x}_i)$: total cost of subsystem i
$W_i(\mathbf{x}_i)$: total weight of subsystem i.

The following assumptions are made:

- The components and the system can be in one of two states: operating or failed.
- Failures of components are independent.
- The failure rates of components are independent of whether they are in use or not, i.e., there is active redundancy.

Chern (1992) shows that the series-parallel RAP is NP-hard. The problem has been widely studied with different approaches including dynamic programming (Bellman 1957; Bellman and Dreyfus 1958, 1962; Fyffe et al. 1968; Nakagawa and Miyazaki 1981) and integer programming (Ghare and Taylor 1969, Bulfin and Liu 1985, Misra and Sharma 1991). More

recently, heuristic methods such as GA (Painton and Campbell 1995; Coit and Smith 1996a, b; Coit et al. 1996) and the ant system algorithm (Liang and Smith 1999) have been applied to the problem.

4.2.1. The TS Algorithm. The encoding is the same as that used in Coit and Smith (1996b), which is a straightforward permutation encoding of size $\sum_{i=1}^{s} n_{\max,i}$ representing a concatenation of the components in each subsystem sorted from most reliable to least reliable including nonused components (i.e., blanks when $n_i < n_{\max,i}$) that have a reliability of 0. To obtain an initial solution, s integers between k_i and $n_{\max} - 3$ were uniformly randomly chosen to represent the number of parts in parallel (n_i) for a particular subsystem. Then, n_i components were randomly and uniformly selected from among the m_i different types.

Moves operate on subsystems only, and two kinds are used. The first changes the number of a particular component by adding or subtracting one, for all available component types. For example, if there are two type 1 components in a subsystem, change to a subsystem with either one type 1 component or three type 1 components. The second simultaneously adds one component to a component type of a certain subsystem and deletes one component from another component type in the same subsystem (enumerating all possibilities). For example, if a subsystem has three of component type 2 and one of component type 4, then one possibility deletes a component type 2 and adds a component type 4. A second possibility adds a component type 2 and deletes a component type 4. The two types of moves are performed independently on the current solution, and the best move among them is selected. An important advantage of these types of moves is that they do not require recalculating the entire system reliability. Each time only one subsystem is changed; therefore, only the reliability of that subsystem is recalculated and system reliability is updated accordingly. After considering all subsystems and all components within each subsystem, the best non-tabu candidate is selected as the move.

The structure of the subsystem that has been changed (in the accepted move) is stored in the tabu list. For example, if subsystem two has been changed from one with two type 1, three type 2, and one type 3 components ($\mathbf{x}_2 = 1\ 1\ 2\ 2\ 2\ 3\ 0\ 0$) to one with one type 1, three type 2, and one type 3 component ($\mathbf{x}_2 = 1\ 2\ 2\ 2\ 3\ 0\ 0\ 0$), then any move with two type 1, three type 2, and one type 3 component in subsystem two is now tabu. To know if an entry on the tabu list is feasible or infeasible, the system cost and weight are kept with the subsystem structure in the tabu list. The length of the tabu list changes every 20 iterations to an integer distributed uniformly on $[s, 3s]$. The stopping criterion was defined as 1,000 iterations without finding an improvement in the best feasible solution.

4.2.2. Penalty Function. Because the series-parallel system RAP is formulated with two independent constraints (cost and weight), the penalty function is a linear summation as shown in (1) with $N = 2$.

$$R_p(\mathbf{x}) = R(\mathbf{x}) - (R_{all} - R_{feas})\left[\left(\frac{\Delta w}{\mathrm{NFT}_w}\right)^{\kappa_1} + \left(\frac{\Delta c}{\mathrm{NFT}_c}\right)^{\kappa_2}\right]$$

Δc and Δw represent the magnitude of the constraint violations, and NFT_{co} and NFT_{wo} are set to 1% of the constraint values, C and W. κ is set to 1 in each case, though results are not sensitive to this value.

4.2.3. Test Problems and Results. The 14 subsystems test problems considered were originally proposed by Fyffe et al. (1968) and modified by Nakagawa and Miyazaki (1981). Fyffe et al. (1968) specified 130 units of system cost, 170 units of system weight, and $k_i = 1$ (i.e., 1-out-of-n:G subsystems). Nakagawa and Miyazaki (1981) developed 33 variations of the original problem, where the cost constraint is maintained at 130 and the weight constraint, W, varies from 191 to 159. In both papers, the assumption was that only identical components could be placed in parallel. Coit and Smith (1996b), however, relaxed this assumption and assumed that different components could be placed in parallel. For all subsystems $n_{\max,i}$ is set to be 8.

The results of TS with the three different penalty methods are compared in Table 4 with the best solution for each row set in bold. Although differences in reliability are small, keep in mind that reliability is bounded by 1.0, and any increase in system reliability will result in significant savings over the life of the system. Results were similar to the layout problem—the N&I approach did not perform nearly as well as the others, and the GHL approach was more variable, which, in this problem class, caused worse mean and worst-case performance. In Table 4, the average CPU time in seconds over all instances for TS using memory-based penalty, N&I's penalty, and GHL's penalty are given. Two competitive methods, GHL's penalty and memory-based penalty of this paper, have similar CPU times. Figure 4 shows the NFT behavior of the weight constraint as it varies throughout the duration of the TS. As in Figure 3, NFT is actively altering to encourage a thorough search of the boundary area about feasibility and infeasibility. After more extreme adjustments early in the search, the memory property of the penalty function finds an NFT ranging from 1 to 6 to be most effective.

As in the previous problem class, more constrained versions needed to be tested. Therefore, another version of the RAP, which minimizes system cost given overall restrictions on maximum system weight of W and minimum system reliability R, was used.

Table 4 Comparison of TS Penalty Approaches to Redundancy Allocation

			System reliability (R) of TS over 10 runs								
			Memory-based			N&I (1998)			GHL (1994)		
No.	C	W	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min
1	130	191	**0.9868**	**0.9867**	**0.9865**	0.9742	0.9616	0.9354	**0.9868**	**0.9867**	**0.9865**
2	130	190	**0.9864**	**0.9863**	0.9861	0.9744	0.9648	0.9568	**0.9864**	**0.9863**	**0.9862**
3	130	189	**0.9859**	**0.9858**	0.9855	0.9849	0.9692	0.9570	**0.9859**	**0.9858**	**0.9857**
4	130	188	**0.9854**	0.9851	0.9850	0.9715	0.9524	0.9013	**0.9854**	**0.9853**	**0.9851**
5	130	187	**0.9847**	**0.9847**	**0.9847**	0.9670	0.9622	0.9575	**0.9847**	**0.9847**	**0.9847**
6	130	186	**0.9842**	**0.9842**	**0.9842**	0.9673	0.9469	0.8751	**0.9842**	**0.9842**	**0.9842**
7	130	185	**0.9835**	**0.9835**	**0.9834**	0.9704	0.9596	0.9451	**0.9835**	0.9834	0.9832
8	130	184	**0.9830**	**0.9830**	**0.9830**	0.9664	0.9577	0.9405	**0.9830**	0.9829	0.9827
9	130	183	**0.9823**	**0.9823**	**0.9822**	0.9666	0.9473	0.8837	**0.9823**	0.9822	0.9819
10	130	182	**0.9815**	**0.9815**	**0.9812**	0.9656	0.9614	0.9562	**0.9815**	**0.9815**	**0.9812**
11	130	181	**0.9810**	**0.9809**	**0.9805**	0.9653	0.9463	0.8956	**0.9810**	**0.9809**	**0.9805**
12	130	180	**0.9803**	**0.9803**	**0.9803**	0.9666	0.9372	0.8801	**0.9803**	0.9802	0.9800
13	130	179	**0.9795**	**0.9795**	**0.9793**	0.9678	0.9499	0.8959	**0.9795**	0.9794	**0.9793**
14	130	178	**0.9784**	**0.9784**	**0.9784**	0.9677	0.9438	0.8615	**0.9784**	**0.9784**	**0.9784**
15	130	177	**0.9776**	**0.9776**	**0.9776**	0.9671	0.9423	0.8757	**0.9776**	**0.9776**	0.9775
16	130	176	**0.9767**	**0.9767**	**0.9765**	0.9617	0.9524	0.9127	**0.9767**	0.9766	**0.9765**
17	130	175	**0.9757**	**0.9757**	**0.9757**	0.9607	0.9417	0.9085	**0.9757**	0.9756	0.9756
18	130	174	**0.9749**	**0.9749**	**0.9749**	0.9595	0.9463	0.9132	**0.9749**	**0.9749**	0.9748
19	130	173	**0.9738**	**0.9738**	**0.9738**	0.9604	0.9228	0.7879	**0.9738**	**0.9738**	**0.9738**
20	130	172	**0.9730**	**0.9730**	**0.9730**	0.9579	0.9266	0.8403	**0.9730**	**0.9730**	**0.9730**
21	130	171	**0.9719**	**0.9719**	**0.9719**	0.9573	0.9103	0.8313	**0.9719**	**0.9719**	**0.9719**
22	130	170	**0.9708**	**0.9708**	**0.9708**	0.9513	0.9079	0.8259	**0.9708**	**0.9708**	**0.9708**
23	130	169	**0.9693**	**0.9693**	**0.9693**	0.9538	0.9151	0.8205	**0.9693**	**0.9693**	**0.9693**
24	130	168	**0.9681**	**0.9681**	**0.9681**	0.9505	0.8942	0.8249	**0.9681**	0.9680	0.9673
25	130	167	**0.9663**	**0.9663**	**0.9663**	0.9478	0.9161	0.8901	**0.9663**	**0.9663**	**0.9663**
26	130	166	**0.9650**	**0.9650**	**0.9650**	0.9498	0.9151	0.8617	**0.9650**	**0.9650**	0.9646
27	130	165	**0.9637**	**0.9637**	**0.9637**	0.9498	0.9074	0.8420	**0.9637**	**0.9637**	0.9636
28	130	164	**0.9624**	**0.9624**	**0.9624**	0.9440	0.8971	0.8530	**0.9624**	0.9623	0.9617
29	130	163	**0.9606**	**0.9606**	**0.9605**	0.9384	0.8951	0.8362	**0.9606**	**0.9606**	0.9600
30	130	162	**0.9592**	**0.9592**	**0.9592**	0.9368	0.8976	0.8051	**0.9592**	0.9591	0.9589
31	130	161	**0.9580**	**0.9580**	**0.9580**	0.9445	0.9001	0.8571	**0.9580**	**0.9580**	**0.9580**
32	130	160	**0.9557**	**0.9557**	**0.9557**	0.9322	0.8931	0.8055	**0.9557**	**0.9557**	**0.9557**
33	130	159	**0.9546**	**0.9546**	**0.9546**	0.9347	0.9041	0.8594	**0.9546**	0.9544	0.9536
Average CPU (sec)				*2.41*			*1.38*			*3.08*	

Problem P2. Cost minimization:

$$\min \quad C(\mathbf{x}) = \sum_{i=1}^{s} \sum_{j=1}^{m_i} c_{ij} x_{ij}$$

$$\text{subject to} \quad R(t_0; \mathbf{x}) \geq R$$

$$\sum_{i=1}^{s} \sum_{j=1}^{m_i} w_{ij} x_{ij} \leq W$$

$$k_i \leq \sum_{j=1}^{m_i} x_{ij} \leq n_{\max, i}$$

$$\forall i = 1, \ldots, s, \; x_{ij} \in \{0, 1, 2, \ldots\}.$$

This version carries more practical meaning because it is often used in actual engineering design problems; however, it is more difficult to find feasible solutions. For P1, it is always possible to find feasible solutions by reducing the number of components. However, in P2, the two constraints work against each other—It is necessary to add components to reach the minimum system reliability, which increases the system weight. Using $R = 0.9700$ and $W = 170$, no feasible solutions were found in 10,000,000 randomly generated solutions. Table 5 shows results between the memory-based approach and GHL. While both found the same best solution, of ten trials, the

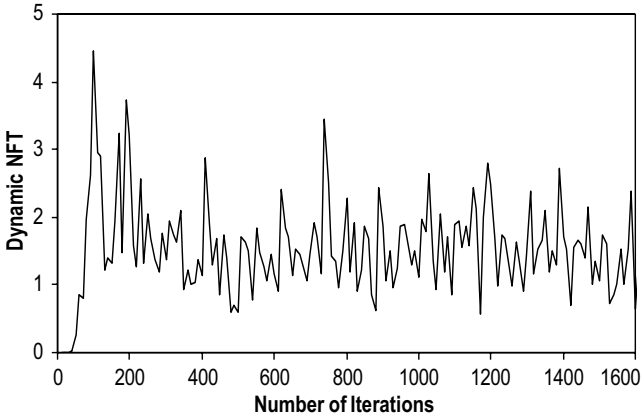

Figure 4 Typical NFT Behavior Over Search for the Redundancy-Allocation Problem (Weight Constraint)

Table 5 Results of the Very Constrained Version of the RAP

Method	$R = 0.9700$, $W = 170$
Memory-based	
Min C	**117**
Mean C	**118.5**
Max C	**120**
Average CPU (sec)	*14.20*
GHL (1994)	
Min C	**117**
Mean C	120
Max C	123
Average CPU (sec)	*15.30*

memory-based approach found feasible solutions four times while GHL found feasible solutions two times. Again, computational effort was roughly equal.

4.3. Orienteering Problem

Given a set of n control points with associated scores $S_i \geq 0$, along with start and end points (which have no score), the orienteering problem (OP) finds a path that maximizes the total score subject to a given time (or distance) budget, denoted by $T_{\max}$. A path between nodes i and j has a cost c_{ij} associated with it and each node can be visited at most once. Because of the constraint, tours will not include all points. The OP is equivalent to the traveling salesman problem (TSP) when the time is relaxed just enough to cover all points and start and end points are not specified. The OP is NP-hard and has applications in vehicle routing and production scheduling, as discussed in Golden et al. (1987) and Keller (1989). Generally, the mathematical model of the OP is formulated as follows, where $x_{ij} = 1$ if the path includes traveling from node i to node j:

$$\max \sum_{i=1}^{n}\sum_{j=1}^{n} S_i x_{ij}$$

$$\text{subject to } \sum_{j=2}^{n} x_{1j} = \sum_{i=1}^{n-1} x_{in} = 1$$

$$\sum_{i=2}^{n-1} x_{ik} = \sum_{j=2}^{n-1} x_{kj} \leq 1, \quad k = 2, \ldots, n-1$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} \leq T_{\max}$$

$$x_{ij} \in \{0, 1\}, \ i, j = 1, \ldots, n.$$

The OP has been studied with a number of heuristics, including Tsiligirides (1984), Golden et al. (1987, 1988), Keller (1989), Ramesh and Brown (1991), Kantor and Rosenwein (1992), Mittenthal and Noon (1992), Wang et al. (1995), and Chao et al. (1996a, b). These heuristics include neural networks, tree-based algorithms, center-of-gravity algorithms, and problem-specific improvement algorithms.

4.3.1. The TS Algorithm. The encoding is the order of cities visited. To generate a random initial solution, a simple heuristic is used. First, the number of cities to be visited is randomly chosen. Second, the total distance from each city to every other city is calculated. The ratio of the score of a city to its total distance is found, and this ratio correlates with the probability of including that city in the initial tour. Using these probabilities and uniform random numbers from 0 to 1, a (variable-length) permutation of visited cities is created.

Five move operators were used. The first inserts the city in the ith location after the jth location. The second adds a city, which is not in the tour, to the tour after the jth location. The third deletes the city in the ith location from the tour. The fourth simultaneously inserts a city and deletes a city. The fifth reverses the order of cities between two selected positions while keeping the origin and destination unchanged. The length of the tabu list changes every 20 iterations to an integer distributed uniformly on [$NC/2$, $2NC$] where NC is the total number of cities. The stopping criterion was 200 iterations without finding an improvement in the best feasible solution.

4.3.2. Penalty Function. Following (1), the penalized objective function is

$$S_p(\mathbf{x}) = S(\mathbf{x}) - (S_{all} - S_{feas})\left(\frac{\Delta d}{\mathrm{NFT}_d}\right)^{\kappa}.$$

Δd represents the magnitude of the time-constraint violation, NFT_{d0} is set to 10% of the constraint value, $T_{\max}$, and κ is set to 2.

4.3.3. Test Problems and Results. The test problems are those most studied in the literature (Chao et al. 1996a, Tsiligirides 1984). These are three sets of size 32, 21, and 33 nodes with 18, 11, and 20 instances, respectively, where each instance is generated by varying the $T_{\max}$ value. Chao et al. (1996a) found a mistake in the original data set of Tsiligirides (1984) in the size-32 problem, corrected the mistake, and created a new data set, named data set 4, which is different from the old set at node 30. The search spaces are of size 1.2×10^{17} for the 21-node problem, 2.7×10^{32} for the 32-node problem, and 8.2×10^{33} for the 33 node-problem. The N&I penalty approach performed poorly on these problems in all cases so exact results are not shown. Table 6 shows the remaining results compared to the best heuristic in the literature, that of Chao et al. (1996a). Both the proposed memory-based penalty and GHL's penalty performed well; the best of ten runs for each problem instance of both equaled or bettered the best results in the literature. For mean and worst over ten runs, results are fairly evenly mixed between the two methods over the problem instances, and the overall

conclusion is that either method is very effective for the OP problem. As shown in Figure 5, an NFT ranging from 0.5 to 3 is found to be most effective for this problem class.

It is difficult to make a definitive computational comparison since CPU seconds may vary according to hardware, software, and coding. Chao et al.'s (1996a) heuristic was coded in FORTRAN and executed on a SUN 4/370 workstation, whereas the TS algorithms were coded in C and run using a SUN Ultra 2 Model 2170 workstation with 168 MHz dual processors and 128 MB of RAM. In order to conduct a fair comparison, all CPU times were converted to their equivalent time on a SUN Ultra 2 machine (assuming that the machine of Chao et al. (1996a) was roughly nine times slower than the TS machine). In Table 6, converted average CPU times in seconds over all instances for Chao et al. (1996a), GHL, and the memory-based approach are given. All three have similar CPU times.

Table 6 Comparison of Results on the Orienteering Problem

	Score						
	Chao et al. (1996a)	Memory-based			GHL (1994)		
T_{max}	Max	Max	Mean	Min	Max	Mean	Min
Problem Set 1—32 nodes							
5	10	10	10	10	10	10	10
10	15	15	15	15	15	15	15
15	45	45	43.5	30	45	45	45
20	65	65	60	45	65	63	55
25	90	90	88.5	75	90	90	90
30	110	110	110	110	110	110	110
35	135	135	130	100	135	132.5	130
40	155	155	151.5	130	155	152.5	130
46	175	175	170	150	175	175	175
50	190	190	185	165	190	188	180
55	205	205	200.5	180	205	200	190
60	225	225	221	215	225	222.5	220
65	240	240	238	220	240	240	240
70	260	260	257.5	235	260	260	260
73	265	265	265	265	265	265	265
75	270	270	270	270	270	270	270
80	280	280	280	280	280	278.5	265
85	285	285	285	285	285	285	285
Problem Set 2—21 nodes							
15	120	120	120	120	120	120	120
20	200	200	200	200	200	200	200
23	210	210	210	210	210	210	210
25	230	230	230	230	230	230	230
27	230	230	221.5	220	230	225	220
30	265	265	264.5	260	265	265	265
32	300	300	293	265	300	297	270
35	320	320	311	310	320	312.5	305
38	360	360	355.5	350	360	354	350
40	395	395	395	395	395	395	395
45	450	450	450	450	450	450	450
Problem Set 3—33 nodes							
15	170	170	170	170	170	170	170
20	200	200	200	200	200	198	180
25	260	260	256	240	260	258	250
30	320	320	320	320	320	320	320
35	390	390	369	260	390	374	350
40	430	430	401	300	430	422	410
45	470	470	460	450	470	468	460
50	520	520	508	440	520	512	440
55	550	550	549	540	550	527	480
60	580	580	576	560	580	576	560
65	610	610	608	600	610	604	560
70	640	640	632	580	640	633	600
75	670	670	669	660	670	666	640
80	710	710	705	700	710	700	660
85	740	740	731	690	740	739	730
90	770	770	766	730	770	767	740
95	790	790	790	790	790	787	760
100	800	800	800	800	800	799	790
105	800	800	800	800	800	800	800
110	800	800	800	800	800	800	800

Table 6 Continued

	Score						
	Chao et al. (1996a)	Memory-based			GHL (1994)		
T_{max}	Max	Max	Mean	Min	Max	Mean	Min
Problem Set 4—32 nodes (corrected)							
5	10	10	10	10	10	10	10
10	15	15	15	15	15	15	15
15	45	45	43.5	30	45	45	45
20	65	65	61	45	65	63	55
25	90	90	88.5	75	90	90	90
30	110	110	110	110	110	110	110
35	135	135	128	100	135	132.5	125
40	155	155	149	120	155	153	135
46	175	175	170	150	175	172.5	160
50	190	190	185	165	190	184.5	165
55	205	205	200.5	180	205	201.5	195
60	220	225	222	220	225	221.5	215
65	240	240	239	230	240	240	240
70	260	260	257.5	245	260	257	245
73	265	265	261.5	240	265	263.5	250
75	275	275	273	255	275	269.5	255
80	280	280	278.5	270	280	279	270
85	285	285	285	285	285	285	285
Average CPU (sec)	*1.58*		*1.56*			*2.02*	

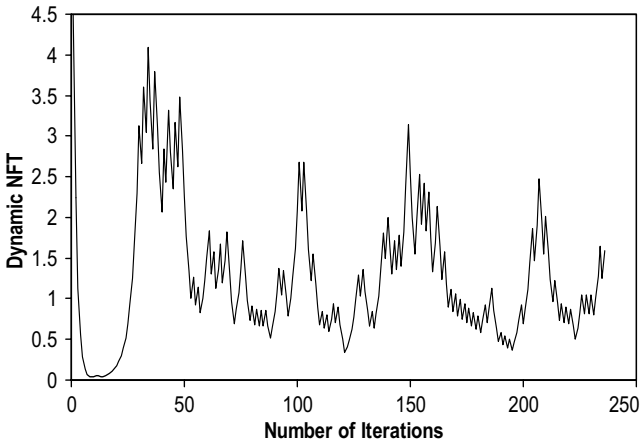

Figure 5 Typical NFT Behavior Over Search for the Orienteering Problem

5. Conclusions

This paper has put forth a general method for effectively handling constraints when using a tabu-search metaheuristic. The memory property of TS is distinct and has proven useful for many difficult combinatorial problems. Most of these problems are constrained, and discarding or repairing infeasible solutions has been observed to be inefficient. The memory-based penalty function of this paper encourages search within both the feasible region and promising areas of the infeasible region. It self-scales to the magnitude of each constraint and requires setting an initial near-feasibility threshold and an amplification exponent for each constraint, though these can be easily established and results are robust to their values. The penalty adapts to both the long-term memory of the search (through comparison of the best solution and best feasible solution found) and the short-term memory of the search (through feasibility/infeasibility characterization of the current move relative to recently accepted moves).

This approach can be used on a variety of constrained problems. In computational comparisons on three distinct combinatorial problems, the memory-based approach is superior to the N&I approach with many fewer parameters to set a priori. Comparisons with the GHL approach are mixed. The number of parameters to set in each method is essentially the same (κ and NFT_0 in this method, and initial weights and h in the GHL method). The methods both change the penalty applied according to recent search history; the memory-based approach explicitly uses the tabu list, and the GHL approach uses a window size, h. Both handle multiple constraints independently and use the distance from feasibility as the main penalty term. The memory-based approach adds a self-scaling of constraints through the NFT and feedback from the long-term memory using the best solutions (feasible and overall) found. It seems that for some problems such as the OP, these are not advantageous additions; however, for other problems, especially those with multiple constraints, such as the RAP, or those that are more tightly constrained, they may well be. Computational evidence herein indicates that the GHL approach is more variable to seed and is less competitive with the memory-based approach in the most highly constrained instances.

Finally, the dynamic behavior of NFT over the search clearly illustrates that a penalty that adjusts to search history is more effective than a static penalty or one that operates monotonically. Furthermore, the dynamic behavior eliminates the need to choose the initial value of NFT wisely as it will automatically adjust upwards or downwards as needed. The method of this paper is simple, general, and effective, and is a step forward in using memory to advantage in TS.

Acknowledgments

Part of this work has been supported by the National Science Foundation Grants DMI 9908322, DMI 9874716, and DMI 9502134.

References

Armour, G. C., E. S. Buffa. 1963. A heuristic algorithm and simulation approach to relative location of facilities. *Management Sci.* **9** 294–309.

Avriel, M. 1976. *Nonlinear Programming: Analysis and Methods*. Prentice Hall, Englewood Cliffs, NJ.

Bazaraa, M. S. 1975. Computerized layout design: A branch and bound approach. *AIIE Trans.* **7** 432–438.

Bellman, R. E. 1957. *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Bellman, R. E., S. E. Dreyfus. 1958. Dynamic programming and reliability of multicomponent devices. *Oper. Res.* **6** 200–206.

Bellman, R. E., S. E. Dreyfus. 1962. *Applied Dynamic Programming*. Princeton University Press, Princeton, NJ.

Bulfin, R. L., C. Y. Liu. 1985. Optimal allocation of redundant components for large systems. *IEEE Trans. Reliability* **R-34** 241–247.

Chao, I-M., B. L. Golden, E. A. Wasil. 1996a. A fast and effective heuristic for the orienteering problem. *Eur. J. Oper. Res.* **88** 475–489.

Chao, I-M., B. L. Golden, E. A. Wasil. 1996b. The orienteering problem. *Eur. J. Oper. Res.* **88** 464–474.

Chern, M. S. 1992. On the computational complexity of reliability redundancy allocation in a series system. *Oper. Res. Lett.* **11** 309–315.

Coit, D. W., A. E. Smith. 1996a. Penalty guided genetic search for reliability design optimization. *Comput. Indust. Engrg.* **30** 895–904.

Coit, D. W., A. E. Smith. 1996b. Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Trans. Reliability* **45** 254–260.

Coit, D. W., A. E. Smith, D. M. Tate. 1996. Adaptive penalty methods for genetic optimization of constrained combinatorial problems. *INFORMS J. Comput.* **8** 173–182.

Committee on the Next Decade of Operations Research (Condor). 1988. Operations research: The next decade. *Oper. Res.* **36** 619–637.

Costa, D. 1990. A tabu search algorithm for computing an operational time table. Working paper, Département de Mathématiques, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

Dammeyer, F., P. Forst, S. Voss. 1991. On the cancellation sequence method of tabu search. *ORSA J. Comput.* **3** 262–265.

Dammeyer, F., S. Voss. 1993. Dynamic tabu list management using the reverse elimination method. *Ann. Oper. Res.* **41** 31–46.

Fisher, M. L. 1981. The Lagrangian relaxation method for solving integer programming problems. *Management Sci.* **27** 1–18.

Fyffe, D. E., W. W. Hines, N. K. Lee. 1968. System reliability allocations and a computational algorithm. *IEEE Trans. Reliability* **R-17** 74–79.

Gendreau, M., A. Hertz, G. Laporte. 1994. A tabu search heuristic for the vehicle routing problem. *Management Sci.* **40** 1276–1290.

Ghare, M., R. E. Taylor. 1969. Optimal redundancy for reliability in series system. *Oper. Res.* **17** 838–847.

Glover, F. 1986. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **13** 533–549.

Glover, F. 1989. Tabu search—Part I. *ORSA J. Comput.* **1** 190–206.

Glover, F. 1990. Tabu search—Part II. *ORSA J. Comput.* **2** 4–32.

Glover, F., M. Laguna. 1997. *Tabu Search*. Kluwer Academic Publishers, London, U.K.

Glover, F., E. Taillard, D. de Werra. 1993. A user's guide to tabu search. *Ann. Oper. Res.* **41** 3–28.

Golden, B. L., L. Levy, R. Vohra. 1987. The orienteering problem. *Naval Res. Logist.* **34** 307–318.

Golden, B. L., Q. Wang, L. Liu. 1988. A multifacet heuristic for the orienteering problem. *Naval Res. Logist.* **35** 359–366.

Hertz, A. 1992. Finding a feasible course schedule using tabu search. *Discrete Appl. Math.* **35** 255–270.

Hertz, A., D. de Werra. 1989. Informatique et horaires scolaires. *Output* **12** 53–56.

Joines, J. A., C. R. Houck. 1994. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's. *Proc. First IEEE Conf. Evolutionary Comput.* IEEE, Piscataway, NJ, 569–573.

Kantor, M. G., M. B. Rosenwein. 1992. The orienteering problem with time windows. *J. Oper. Res. Soc.* **43** 629–635.

Keller, C. P. 1989. Algorithms to solve the orienteering problem: A comparison. *Eur. J. Oper. Res.* **41** 224–231.

Liang, Y-C., A. E. Smith. 1999. An ant system approach to redundancy allocation. *Proc. 1999 Congress Evolutionary Comput.* IEEE, Piscataway, NJ, 1478–1484.

Misra, K. B., U. Sharma. 1991. An efficient algorithm to solve integer programming problems arising in system reliability design. *IEEE Trans. Reliability* **40** 81–91.

Mittenthal, J., C. E. Noon. 1992. An insert/delete heuristic for the travelling salesman subset-tour problem with one additional constraint. *J. Oper. Res. Soc.* **43** 277–283.

Nakagawa, Y., S. Miyazaki. 1981. Surrogate constraints algorithm for reliability optimization problems with two constraints. *IEEE Trans. Reliability* **R-30** 175–180.

Nonobe, K., T. Ibaraki. 1998. A tabu search approach to the constraint satisfaction problem as a general problem solver. *Eur. J. Oper. Res.* **106** 599–623.

Osman, I. H., N. Christofides. 1994. Capacitated clustering problems by hybrid simulated annealing and tabu search. *Internat. Trans. Oper. Res.* **1** 317–336.

Painton, L., J. Campbell. 1995. Genetic algorithms in optimization of system reliability. *IEEE Trans. Reliability* **44** 172–179.

Ramesh, R., K. M. Brown. 1991. An efficient four-phase heuristic for the generalized orienteering problem. *Comput. Oper. Res.* **18** 151–165.

Reeves, C. R. 1993. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley and Sons, New York.

Scholl, A., S. Voss. 1996. Simple assembly line balancing—Heuristic approaches. *J. Heuristics* **2** 217–244.

Schwefel, H-P. 1995. *Evolution and Optimum Seeking*. John Wiley and Sons, New York.

Smith, A. E., D. W. Coit. 1995. Penalty functions. T. Baeck, D. Fogel, Z. Michalewicz, eds. *Handbook of Evolutionary Computation*, Oxford University Press and Institute of Physics Publishing, Bristol, U.K., Chap. C5.2.

Smith, A. E., D. M. Tate. 1993. Genetic optimization using a penalty function. *Proc. Fifth Internat Conf. Genetic Algorithms*. Morgan Kaufmann, San Francisco, CA, 499–505.

Taillard, E., P. Badeau, M. Gendreau, F. Guertin, J-Y. Potvin. 1997. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Sci.* **31** 170–186.

Tate, D. M., A. E. Smith. 1995. Unequal area facility layout using genetic search. *IIE Trans.* **27** 465–472.

Thomas, P. R., S. Salhi. 1998. A tabu search approach for the resource constrained project scheduling problem. *J. Heuristics* **4** 123–139.

Tsiligirides, T. 1984. Heuristic methods applied to orienteering. *J. Oper. Res. Soc.* **35** 797–809.

Wang, Q., X. Sun, B. L. Golden, J. Jia. 1995. Using artificial neural networks to solve the orienteering problem. *Ann. Oper. Res.* **61** 111–120.