23

Principles of Tabu Search

Fred Glover
University of Colorado

Manuel Laguna
University of Colorado

Rafael Martí
University of Valencia

23.1 Introduction

Tabu search (TS) is a metaheuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality. The term *tabu search* was coined in the same paper that introduced the term *metaheuristic* [1]. Tabu search is based on the premise that problem solving, to qualify as intelligent, must incorporate *adaptive memory* and *responsive exploration*. The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memoryless designs that heavily rely on semirandom processes that implement a form of sampling. The emphasis on responsive exploration (and hence purpose) in TS, whether in a deterministic or probabilistic implementation, derives from the supposition that a bad strategic choice can often yield more information than a good random choice. Over a wide range of problem settings, strategic use of memory can make dramatic differences in the ability to solve problems.

Tabu search can be directly applied to virtually any kind of optimization problem. We can state most of these problems in the following form, where "optimize" means to minimize or maximize:

$$\begin{aligned} &\text{Optimize} && f(x) \\ &\text{subject to} && x \in \mathrm{X} \end{aligned}$$

The function $f(x)$ may be linear, nonlinear, or even stochastic, and the set $\mathbf{X}$ summarizes constraints on the vector of decision variables x. The constraints may similarly include linear, nonlinear, or stochastic inequalities, and may compel all or some components of x to receive discrete values.

While this representation is useful for discussing a number of problem-solving considerations, we emphasize that in many applications of combinatorial optimization, the problem of interest may not be easily formulated as an objective function subject to a set of constraints. The requirement $x \in \mathbf{X}$, for example, may specify logical conditions or interconnections that would be cumbersome to formulate mathematically, but may be better left as verbal stipulations that can be then coded as rules.

The TS technique is rapidly becoming the method of choice for designing solution procedures for hard combinatorial optimization problems. A comprehensive examination of this methodology can be found in the book by Glover and Laguna [2]. Widespread successes in practical applications of optimization have spurred a rapid growth of the method as a means of identifying extremely high-quality solutions

TABLE 23.1 Illustrative Tabu Search Applications

Scheduling	**Telecommunications**
Flow-time cell manufacturing	Call routing
Heterogeneous processor scheduling	Bandwidth packing
Workforce planning	Hub facility location
Rostering	Path assignment
Machine scheduling	Network design for services
Flow shop scheduling	Customer discount planning
Job shop scheduling	Failure immune architecture
Sequencing and batching	Synchronous optical networks
Design	**Production, Inventory, and Investment**
Computer-aided design	Supply chain management
Fault tolerant networks	Flexible manufacturing
Transport network design	Just-in-time production
Architectural space planning	Capacitated MRP
Diagram coherency	Part selection
Fixed charge network design	Multiitem inventory planning
Irregular cutting problems	Volume discount acquisition
Layout planning	Project portfolio optimization
Logic and Artificial Intelligence	**Routing**
Maximum satisfiability	Vehicle routing
Probabilistic logic	Capacitated routing
Pattern recognition/classification	Time window routing
Data mining	Multimode routing
Clustering	Mixed fleet routing
Statistical discrimination	Traveling salesman
Neural network training	Traveling purchaser
Neural network design	Convoy scheduling
Location and Allocation	**Graph Optimization**
Multicommodity location/allocation	Graph partitioning
Quadratic assignment	Graph coloring
Quadratic semiassignment	Clique partitioning
Multilevel generalized assignment	Maximum clique problems
Large-scale GAP problems	Maximum planner graphs
Technology	**General Combinational Optimization**
Seismic inversion	Zero–one programming
Electrical power distribution	Fixed charge optimization
Engineering structural design	Nonconvex nonlinear programming
Minimum volume ellipsoids	All-or-none networks
Space station construction	Bilevel programming
Circuit cell placement	Multiobjective discrete optimization
OffShore oil exploration	General mixed integer optimization

efficiently. Tabu search methods have also been used to create hybrid procedures with other heuristic and algorithmic methods, to provide improved solutions to problems in production planning and scheduling, resource allocation, network design, routing, financial analysis, telecommunications, portfolio planning, supply chain management, agent-based modeling, business process design, forecasting, machine learning, data mining, biocomputation, molecular design, forest management and resource planning, and many other areas. Some of the diversity of TS applications is shown in Table 23.1.

The TS emphasis on adaptive memory makes it possible to exploit the types of strategies that underlie the best of human problem-solving, instead of being confined to mimicking the processes found in lower orders of natural phenomena and behavior. The basic elements of TS have several important features, summarized in Table 23.2. Tabu search is concerned with finding new and more effective ways of taking advantage of the concepts embodied in Table 23.2, and with identifying associated principles that can expand the foundations of intelligent search.

TABLE 23.2 Principal Tabu Search Features

Adaptive Memory
Selectivity (including strategic forgetting)
Abstraction and decomposition (through explicit and attributive memory)
Timing
Recency of events
Frequency of events
Differentiation between short term and long term
Quality and impact
Relative attractiveness of alternative choices
Magnitude of changes in structure or constraining
Relationships
Context
Regional interdependence
Structural interdependence
Sequential interdependence
Responsive Exploration
Strategically imposed restraints and inducements (*tabu conditions* and *aspiration levels*)
Concentrated focus on good regions and good solution features (*intensification processes*)
Characterizing and exploring promising new regions (*diversification processes*)
Nonmonotonic search patterns (*strategic oscillation*)
Integrating and extending solutions (*path relinking*)

In this chapter we will describe some key aspects of this methodology, as the use of memory structures and search strategies, and illustrate them in an implementation to solve the linear ordering problem (LOP).

23.2 Memory Structures

Tabu search begins in the same way as ordinary local or neighborhood search, proceeding iteratively from one point (solution) to another until a chosen termination criterion is satisfied. Each solution x has an associated neighborhood $N(x) \subset X$, and each solution $x' \in N(x)$ is reached from x by an operation called a *move*.

We may contrast TS with a simple descent method where the goal is to minimize $f(x)$. Such a method only permits moves to neighbor solutions that improve the current objective function value and ends when no improving solutions can be found. The final x obtained by a descent method is called a local optimum, since it is at least as good as or better than all solutions in its neighborhood. The evident shortcoming of a descent method is that such a local optimum in most cases will not be a global optimum, that is, it usually will not minimize $f(x)$ over all $x \in \mathbf{X}$.

Tabu search permits moves that deteriorate the current objective function value but the moves are chosen from a modified neighborhood $N^*(x)$. Short- and long-term memory structures are responsible for the specific composition of $N^*(x)$. In other words, the modified neighborhood is the result of maintaining a selective history of the states encountered during the search. In the TS strategies based on short-term considerations, $N^*(x)$ characteristically is a subset of $N(x)$, and the tabu classification serves to identify elements of $N(x)$ excluded from $N^*(x)$. In TS strategies that include longer term considerations, $N^*(x)$ may also be expanded to include solutions not ordinarily found in $N(x)$, such as

TABLE 23.3 Examples of Recency-Based Memory

Context	Attributes	To Record the Last Time . . .
Binary problems	Variable index (i)	Variable i changed its value from 0 to 1 or 1 to 0 (depending on its current value)
Job sequencing	Job index (j)	Job j changed positions
	Job index (j) and position (p)	Job j occupied position p
	Pair of job indexes (i, j)	Job i exchange positions with job j
Graphs	Arc index (i)	Arc i was added to the current solution
		Arc i was dropped from the current solution

solutions found and evaluated in past search, or identified as high-quality neighbors of these past solutions. Characterized in this way, TS may be viewed as a dynamic neighborhood method. This means that the neighborhood of x is not a static set, but rather a set that can change according to the history of the search.

The structure of a neighborhood in TS differs from that used in local search in an additional manner, by embracing the types of moves used in constructive and destructive processes (where the foundations for such moves are accordingly called *constructive neighborhoods* and *destructive neighborhoods*). Such expanded uses of the neighborhood concept reinforce a fundamental perspective of TS, which is to define neighborhoods in dynamic ways that can include serial or simultaneous consideration of multiple types of moves.

Tabu search uses attributive memory for guiding purposes (i.e., to compute $N^*(x)$). Instead of recording full solutions, attributive memory structures are based on recording attributes. This type of memory records information about solution properties (attributes) that change in moving from one solution to another. The most common attributive memory approaches are recency- and frequency-based memory. Recency, as its name suggests, keeps track of solutions attributes that have changed during the recent past. Frequency typically consists of ratios about the number of iterations a certain attribute has changed or not (depending whether it is a transition or a residence frequency). Some examples of recency- and frequency-based memory are shown in Table 23.3 and Table 23.4 respectively.

Characteristically, a TS process based strictly on short-term strategies may allow a solution x to be visited more than once, but it is likely that the corresponding reduced neighborhood $N^*(x)$ will be different each time. With the inclusion of longer term considerations, the likelihood of duplicating a previous neighborhood upon revisiting a solution, and more generally of making choices that repeatedly visit only a limited subset of X, is all but nonexistent.

Recency-based memory is the most common memory structure used in TS implementations. As its name suggests, this memory structure keeps track of solutions attributes that have changed during the recent past. To exploit this memory, selected attributes that occur in solutions recently visited are labeled

TABLE 23.4 Examples of Frequency-Based Memory

Context	Residence Measure	Transition Measure
Binary problems	Number of times variable i has been assigned the value of 1	Number of times variable i has changed values
Job sequencing	Number of times job j has occupied position p	Number of times job i has exchanged positions with job j
	Average objective function value when job j occupies position p	Number of times job j has been moved to an earlier position in the sequence
Graphs	Number of times arc i has been part of the current solution	Number of times arc i has been deleted from the current solution when arc j has been added
	Average objective function value when arc i is part of the solution	Number of times arc i has been added during improving moves

tabu-active, and solutions that contain tabu-active elements, or particular combinations of these attributes, are those that become tabu. This prevents certain solutions from the recent past from belonging to $N^*(x)$ and hence from being revisited. Other solutions that share such tabu-active attributes are also similarly prevented from being visited. Note that while the tabu classification strictly refers to solutions that are forbidden to be visited, by virtue of containing tabu-active attributes (or more generally by violating certain restriction based on these attributes), moves that lead to such solutions are also often referred to as being tabu.

Frequency-based memory provides a type of information that complements the information provided by recency-based memory, broadening the foundation for selecting preferred moves. Like recency, frequency often is weighted or decomposed into subclasses. Also, frequency can be integrated with recency to provide a composite structure for creating penalties and inducements that modify move evaluations.

Frequencies typically consist of ratios, whose numerators represent counts expressed in two different measures: a *transition measure*—the number of iterations where an attribute changes (enters or leaves) the solutions visited on a particular trajectory, and a *residence measure*—the number of iterations where an attribute belongs to solutions visited on a particular trajectory, or the number of instances where an attribute belongs to solutions from a particular subset. The denominators generally represent one of three types of quantities: (1) the total number of occurrences of all events represented by the numerators (such as the total number of associated iterations); (2) the sum (or average) of the numerators; and (3) the maximum numerator value. In cases where the numerators represent weighted counts, some of which may be negative, denominator (3) is expressed as an absolute value and denominator (2) is expressed as a sum of absolute values (possibly shifted by a small constant to avoid a zero denominator). The ratios produce *transition frequencies* that keep track of how often attributes change, and *residence frequencies* that keep track of how often attributes are members of solutions generated. In addition to referring to such frequencies, thresholds based on the numerators alone can be useful for indicating when phases of greater diversification are appropriate.

23.3 Search Strategies

The use of recency and frequency memory in TS generally fulfills the function of preventing searching processes from *cycling*, that is, from endlessly executing the same sequence of moves (or more generally, from endlessly and exclusively revisiting the same set of solutions). More broadly, however, the various manifestations of these types of memory are designed to impart additional robustness or vigor to the search.

A key element of the adaptive memory framework of TS is to create a balance between search intensification and diversification. Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Diversification strategies, however, seek to incorporate new attributes and attribute combinations that were not included within solutions generated previously. In one form, these strategies undertake to drive the search into regions dissimilar to those already examined. It is important to keep in mind that intensification and diversification are not mutually opposing, but are rather mutually reinforcing.

Most types of intensification strategies require a means for identifying a set of elite solutions as basis for incorporating good attributes into newly created solutions. Membership in the elite set is often determined by setting a threshold that is connected to the objective function value of the best solution found during the search. A simple instance of the intensification strategy is shown in Figure 23.1. Two simple variants for elite solution selection have proved quite successful. One introduces a diversification measure to assure the solutions recorded differ from each other by a desired degree, and then erases all short-term memory before resuming from the best of the recorded solutions. The other keeps a bounded length sequential list that adds a new solution at the end only if it is better than any previously seen, and the short-term memory that accompanied this solution is also saved.

```
Apply TS short-term memory
Apply an elite selection strategy.
do {
        Choose one of the elite solutions.
        Resume short-term memory TS from chosen solution.
        Add new solutions to elite list when applicable.
} while (iterations < limit and list not empty)
```

FIGURE 23.1 Simple TS intensification approach.

Diversification is automatically created in TS (to some extent) by short-term memory functions, but is particularly reinforced by certain forms of longer term memory. TS diversification strategies are often based on modifying choice rules to bring attributes into the solution that are infrequently used. Alternatively, they may introduce such attributes by periodically applying methods that assemble subsets of these attributes into candidate solutions for continuing the search, or by partially or fully restarting the solution process. Diversification strategies are particularly helpful when better solutions can be reached only by crossing barriers or "humps" in the solution space topology.

The incorporation of modified choice rules can be moderated by using the following penalty function:

$$MoveValue' = MoveValue + d_* Penalty$$

This type of penalty approach is commonly used in TS, where the *Penalty* value is often a function of frequency measures such as those indicated in Table 23.2, and d is an adjustable diversification parameter. Larger d values correspond to a desire for more diversification.

23.4 Advanced Designs: Strategic Oscillation and Path Relinking

There are many forms in which a simple TS implementation can be improved by adding long-term elements. In this paper we restrict our attention to two of the most used methods, namely strategic oscillation and path relinking (PR), which constitute the core of many adaptive memory programming algorithms.

Strategic oscillation operates by orienting moves in relation to a critical level, as identified by a stage of construction or a chosen interval of functional values. Such a critical level or *oscillation boundary* often represents a point where the method would normally stop. Instead of stopping when this boundary is reached, however, the rules for selecting moves are modified, to permit the region defined by the critical level to be crossed. The approach then proceeds for a specified depth beyond the oscillation boundary, and turns around. The oscillation boundary again is approached and crossed, this time from the opposite direction, and the method proceeds to a new turning point (see Figure 23.2).

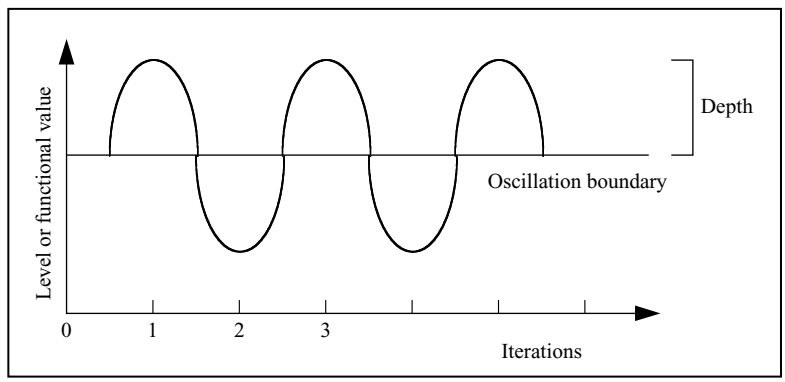

FIGURE 23.2 Strategic oscillation.

The process of repeatedly approaching and crossing the critical level from different directions creates an oscillatory behavior, which gives the method its name. Control over this behavior is established by generating modified evaluations and rules of movement, depending on the region navigated and the direction of search. The possibility of retracing a prior trajectory is avoided by standard TS mechanisms, like those established by the recency-based and frequency-based memory functions.

When the level or functional values in Figure 23.2 refer to degrees of feasibility and infeasibility, a vector-valued function associated with a set of problem constraints can be used to control the oscillation. In this case, controlling the search by bounding this function can be viewed as manipulating a parameterization of the selected constraint set. A preferred alternative is often to make the function a Lagrangean or surrogate constraint penalty function, avoiding vector-valued functions and allowing tradeoffs between degrees of violation of different component constraints.

Path relinking, as a strategy of creating trajectories of moves passing through high-quality solutions was first proposed in connection with TS by Glover [3]. The approach was then elaborated in greater detail as a means of integrating TS intensification and diversification strategies, and given the name PR by Glover and Laguna [4]. Path relinking generally operates by starting from an *initiating solution*, selected from a subset of high-quality solutions, and generating a path in the neighbourhood space that leads toward the other solutions in the subset, which are called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions.

Path relinking can be considered an extension of the combination method of scatter search [4,5]. Instead of directly producing a new solution when combining two or more original solutions, PR generates paths between and beyond the selected solutions in the neighborhood space. The character of such paths is easily specified by reference to solution attributes that are added, dropped, or otherwise modified by the moves executed. Examples of such attributes include edges and nodes of a graph, sequence positions in a schedule, vectors contained in linear programming basic solutions, and values of variables and functions of variables.

The approach may be viewed as an extreme (highly focused) instance of a strategy that seeks to incorporate attributes of high-quality solutions, by creating inducements to favor these attributes in the moves selected. However, instead of using an inducement that merely encourages the inclusion of such attributes, the PR approach subordinates other considerations to the goal of choosing moves that introduce the attributes of the guiding solutions, to create a "good attribute composition" in the current solution. The composition at each step is determined by choosing the best move, using customary choice criteria, from a restricted set—the set of those moves currently available that incorporate a maximum number (or a maximum-weighted value) of the attributes of the guiding solutions. (Exceptions are provided by aspiration criteria, as subsequently noted.) The approach is called PR either by virtue of generating a new path between solutions previously linked by a series of moves executed during a search, or by generating a path between solutions previously linked to other solutions but not to each other.

To generate the desired paths, it is only necessary to select moves that perform the following role: upon starting from an *initiating solution,* the moves must progressively introduce attributes contributed by a *guiding solution* (or reduce the distance between attributes of the initiating and guiding solutions). The roles of the initiating and guiding solutions are interchangeable; each solution can also be induced to move simultaneously toward the other as a way of generating combinations. First, consider the creation of paths that join two selected solutions x' and x'', restricting attention to the part of the path that lies "between" the solutions, producing a solution sequence $x' = x(1), x(2), \ldots, x(r) = x''$. To reduce the number of options to be considered, the solution $x(i+1)$ may be created from $x(i)$ at each step by choosing a move that minimizes the number of moves remaining to reach x''. The relinked path may encounter solutions that may not be better than the initiating or guiding solution, but that provide fertile "points of access" for reaching other, somewhat better, solutions. For this reason it is valuable to examine neighboring solutions along a relinked path, and keep track of those of high quality that may provide a starting point for launching additional searches.

As described by Martí et al. [6], we can apply different PR elements to perform more elaborated designs. Some examples are simultaneous relinking, tunneling strategy, extrapolated relinking, multiple guiding solutions, constructive neighborhoods, or vocabulary building.

23.5 The Linear-Ordering Problem

Given a matrix of weights $E = \{e_{ij}\}_{m \times m}$, the LOP consists of finding a permutation p of the columns (and rows) to maximize the sum of the weights in the upper triangle. In mathematical terms, we seek to maximize

$$C_E(p) = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} e_{p_i p_j}$$

where p_i is the index of the column (and row) in position i in the permutation. Note that in the LOP, the permutation p provides the ordering of both the columns and the rows. Solution methods for this NP-hard problem have been proposed since 1958, when Chenery and Watanabe outlined some ideas on how to obtain solutions for this problem [7]. In this section we describe a TS implementation by Laguna et al. [8] for the LOP.

The LOP has a wide range of applications in several fields. Perhaps, the best known application occurs in the field of economics. In this application, the economy (regional or national) is first subdivided into sectors. Then, an input/output matrix is created, in which the entry (i, j) represents the flow of money from sector i to sector j. Economists are often interested in ordering the sectors so that suppliers tend to come first followed by consumers. This is achieved by permuting the rows and columns of the matrix so that the sum of entries above the diagonal is maximized, which is the objective of the LOP.

Insertions are used as the primary mechanism to move from one solution to another in Laguna et al.'s method for the LOP. *INSERT_MOVE*(p_j, i) consist of deleting p_j from its current position j to be inserted in position i (i.e., between the current sectors p_{i-1} and p_i). This operation results in the ordering p', as follows:

$$p' = \begin{cases} (p_1, \ldots, p_{i-1}, p_j, p_i, \ldots, p_{j-1}, p_{j+1}, \ldots, p_m) & \text{for } i < j \\ (p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_{i-1}, p_j, p_i, \ldots, p_m) & \text{for } i > j \end{cases}$$

The neighborhood N consists of all permutations resulting from executing general insertion moves as

$$N = \{p' : \textit{INSERT\_MOVE}(p_j, i), \textit{for } j = 1, \ldots, m \textit{ and } i = 1, 2, \ldots, j-1, j+1, \ldots, m\},$$

and N is partitioned into m neighborhoods, N^j, associated with each sector p_j, for $j = 1, \ldots, m$:

$$N^j = \{p' : \textit{INSERT\_MOVE}(p_j, i), i = 1, 2, \ldots, j-1, j+1, \ldots, m\}$$

Starting from a randomly generated permutation p, the basic TS procedure alternates between an intensification and a diversification phase. An iteration of the *intensification phase* begins by randomly selecting a sector. The probability of selecting sector j is proportional to its weight w_j according to

$$w_j = \sum_{i \neq j} \left(e_{ij} + e_{ji}\right)$$

The move INSERT_MOVE$(p_j, i) \in N^j$ with the largest move value is selected. (Note that this rule may result in the selection of a nonimproving move.) The move is executed even when the move value is not positive, resulting in a deterioration of the current objective function value. The moved sector becomes tabu-active for *TabuTenure* iterations, and therefore it cannot be selected for insertions during this time.

The number of times that sector j has been chosen to be moved is accumulated in the value *freq*(j). This frequency information is used for diversification purposes. The intensification phase terminates after *MaxInt* consecutive iterations without improvement. Before abandoning this phase, a local search procedure based in the same neighborhood is applied to the best solution found (during the current intensification). We denote this solution as $p^{\#}$, in contrast to p^* (the best solution found over the entire search). By applying this greedy procedure (without tabu restrictions), a local optimum is guaranteed as the output of the intensification phase.

The *diversification phase* is performed for *MaxDiv* iterations. At each iteration, a sector is randomly selected, where the probability of selecting sector j is inversely proportional to the frequency count $freq(j)$. The chosen sector is placed in the best position, as determined by the move values associated with the insert moves in N^j. The procedure stops when *MaxGlo* global iterations are performed without improving $C_E(p^*)$. A global iteration is an application of the intensification phase followed by the application of the diversification phase.

An additional intensification is introduced by implementing a long-term PR phase. Specifically, the best solution found at the end of an intensification phase $p^{\#}$ (which does not necessarily represent p^*, the best solution overall) is subjected to a relinking process. The process consists of making moves starting from $p^{\#}$ (the initiating solution) in the direction of a set of elite solutions (also referred to as guiding solutions). The set of elite solutions consists of the *EltSol* best solutions found during the entire search. The insertions used to move the initiating solution closer to the guiding solutions can be described as follows. For each sector p_j in the current solution:

(1) Find the position i for which the absolute value of $(j\text{-}i)$ is minimized, where i is the position that p_j occupies in at least one of the guiding solutions.
(2) Perform *INSERT_MOVE*(p_j, i).

A long-term diversification phase is also implemented to complement the diversification phase in the basic procedure. The long-term diversification is applied after *MaxLong* global iterations have elapsed without improving $C_E(p^*)$. For each sector p_j, a rounded average position $\alpha(p_j)$ is calculated using the positions occupied by this sector in the set of elite solutions and the solutions visited during the last intensification phase. Then, m diversification steps are performed which insert each sector p_j in its complementary position $m\text{-}\alpha(p_j)$, that is, *INSERT_MOVE*$(p_j, m\text{-}\alpha(p_j))$ is executed for $j = 1, \ldots, m$.

After preliminary experimentation, the search parameters are set to $MaxGlo = 100$, $MaxLong = 50$, $EltSol = 4$, $TabuTenure = 2\sqrt{m}$, $MaxInt = m$, and $MaxDiv = 0.5m$ and $EltSol = 4$. In the 49 instances of the public domain library of linear ordering problems (LOLIB [15]), the method obtains the optimal solution within 1 s of computer time run on a Pentium IV at 3 GHz. The method is also compared with a previous procedure due to Chanas and Kobylanski [9] and a greedy procedure based on the N local search. The methods were run in a way that the best solution found was reported every 0.5 s. These data points were used to generate the performance graph in Figure 23.3. The superior performance of TS_LOP is made evident by Figure 23.3.

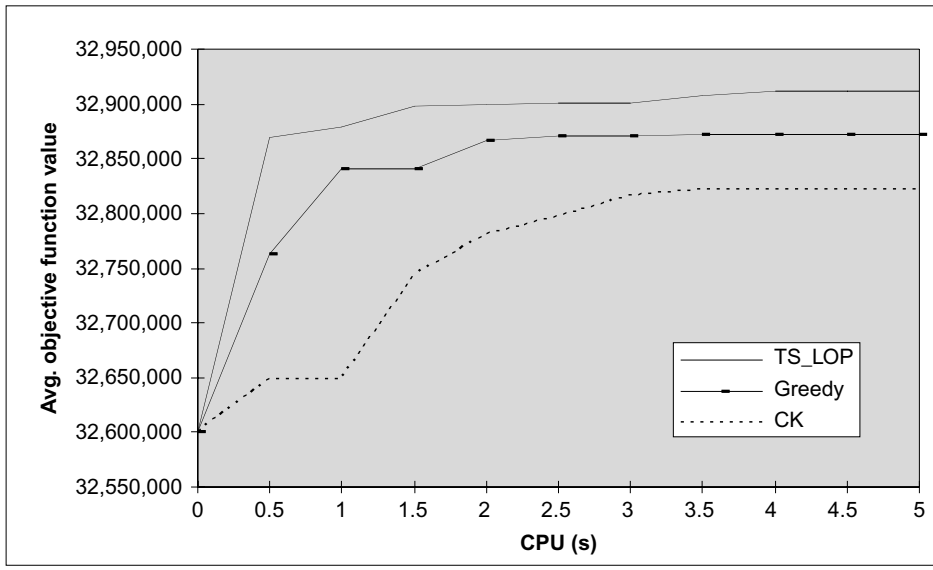

FIGURE 23.3 Performance graph.

23.6 The Tabu Cycle and Conditional Probability Methods

In this section, we describe the implementation and testing of the tabu cycle method and two variants of the conditional probability method [10]. These methods were originally described by Glover [11] and again in a book by Glover and Laguna [2], but have been largely ignored in the TS literature. The *tabu cycle method* is a short-term memory mechanism that is based on partitioning the elements (i.e., move attributes) of a tabu list. The methodology is general and capable of accommodating multiattribute TS memory, as described by Glover and Laguna [2]. In its most basic form, the tabu cycle method divides the short-term memory list into *TabuGroups* groups, where group k consists of elements that were added to the list between a specified range of iterations ago. While in some variants of TS (e.g., probabilistic TS) it is common to progressively relax the tabu status of elements as they become older, the tabu cycle method, by contrast, allows the elements of some groups to fully escape their tabu status according to certain frequencies that increase with the age of the groups. The method is based on the use of iteration intervals called *tabu cycles*, which are made smaller for older groups than for younger groups (with the exception of a small buffer group). Specifically, if group k has a tabu cycle of $TC(k)$ iterations, then at each occurrence of this number of iterations, on average, the elements of group k escape their tabu status and are free to be chosen. In other words, on average, group k is designated as FREE every $TC(k)$ iterations. Mechanisms and data structures that are useful for achieving this are described in Ref. [10].

The *conditional probability method* is a variant of the tabu cycle method that chooses elements by establishing the probability that a group will be FREE on a given iteration. The probability assigned to group k may be viewed conceptually as the inverse of the tabu cycle value. That is, $P(k) = 1/TC(k)$. Analogous to the tabu cycle method, group k is FREE if all older groups likewise are FREE. The method employs a *conditional probability, CP*(k), as a means of determining whether a particular group k can be designated as FREE. The conditional probability values are fixed and the status of a group is determined by a probabilistic process that is not affected by previous choices. Consequently, the approach ignores the possibility that actual tabu cycle values may be far from their targets for some groups. This may happen, for example, when for a number of iterations no elements are chosen from a particular set of FREE groups. The conditional probability method also makes use of a buffer group, for which no element is allowed to escape its tabu status.

A variant of the conditional probability method uses substitute probability values to keep the expected number of elements per iteration chosen from groups no older than any given group k close to $P(k)$. The substitute probabilities replace the original $P(k)$ values in the determination of the conditional probabilities. These substitute probabilities make use of cycle counts, which are also used in connection with the tabu cycle method.

Laguna [10] uses a single machine scheduling problem to test the merit of implementations of the tabu cycle method and both variants of the conditional probability method. The problem consists of minimizing the sum of the setup costs and linear delay penalties when n jobs, arriving at time zero, are to be scheduled for sequential processing on a continuously available machine. Several variants of TS for this problem have been reported in the literature [12–14]. Experiments with more than 300 problem instances with up to 200 jobs were performed to compare a simple static and dynamic short-term memory schemes with a tabu cycle implementation (Cycle), a conditional probability implementation (C-Prob) and an implementation of the conditional probability method with substitute probabilities (S-Prob). The static short-term memory assigns a constant to the tabu tenure to all attributes during the search. The dynamic short-term memory randomly selects from a specified range a tabu tenure. Therefore, the tabu tenure assigned to an attribute in a given iteration may not be the same as the tabu tenure assigned to another attribute in a different iteration. Table 23.5 shows the number of best solutions found by each method in each set of 100 problems.

The results in Table 23.5 show the merit of the tabu cycle and the conditional probability variants as the problem size increases. In addition to these results, the S-Prob is able to find 17 new best solutions to 20 problems used for experimentation by Glover and Laguna [13]. For problems with up to 60 jobs,

TABLE 23.5 Number of Best Solutions (Out of 100) Found by Each Method

Problem Set	Static	Dynamic	Cycle	C-Prob	S-Prob
$n = 50$	2	50	9	31	65
$n = 100$	0	10	28	17	47
$n = 200$	0	8	37	26	29

for which a lower bound can be computed, S-Prob produces a maximum gap of 3.56% in relation to this optimistic bound.

These results confirm that a TS procedure based solely on a static tabu list is not a robust method, because it is incapable of maintaining an acceptable level of diversity during the search. The dynamic short-term memory continues to be an appealing alternative, because it is easy to implement and provides a good balance between diversification and intensification. The results also show that improved outcomes are possible with the additional effort required to implement the tabu cycle or conditional probability methods.

Additional strategies identified by Glover and Laguna [2] can be valuable for exploiting other aspects of intensification and diversification, but this example demonstrates the importance of handling short-term memory in a strategic way, especially when faced with larger and more difficult problems.

23.7 Conclusions

The focus and emphasis of TS have a number of implications for the goal of designing improved optimization procedures. These research opportunities carry with them an emphasis on producing systematic and strategically designed rules, rather than following the policy of relegating decisions to random choices, as often is fashionable in evolutionary methods. The highly attractive results provided by the adaptive memory structures underlying TS are producing an evident impact on the design of metaheuristic methods in general, and are motivating the emergence of new hybrids of TS with other procedures.

Acknowledgments

This research is partially supported by the Spanish Government under code TIN2006-02696.

References

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

[2] Glover, F. and Laguna, M., *Tabu Search,* Kluwer Academic Publishers, Dordrecht, 1997.

[3] Glover, F., Tabu search, Part I, *ORSA J. Comput.*, 1, 190, 1989.

[4] Glover, F. and Laguna, M., Tabu search, in *Modern Heuristic Techniques for Combinatorial Problems,* C. Reeves, Ed., Blackwell Scientific Publishing, Oxford, 1993, p. 70.

[5] Laguna, M. and Martí, R., *Scatter Search—Methodology and Implementations in C,* Kluwer Academic Publishers, Dordrecht, 2003.

[6] Martí, R., Laguna, M., and Glover, F., Principles of scatter search, *Eur. J. Oper. Res.*, 169, 359–372, 2004.

[7] Reinelt, G., The linear ordering problem: algorithms and applications, in *Research and Exposition in Mathematics*, Vol. 8, Hofmann, H. H., and Wille, R., Eds., Heldermann Verlag, Berlin, 1985.

[8] Laguna, M., Martí, R., and Campos, V., Intensification and diversification with elite tabu search solutions for the linear ordering problem, *Comput. Oper. Res.*, 26, 1217, 1999.

[9] Chanas, S. and Kobylanski, P., A new heuristic algorithm solving the linear ordering problem, *Comput. Optimization Appl.*, 6, 191, 1996.
[10] Laguna, M., Implementing and testing the tabu cycle and conditional probability methods, http://leeds-faculty.colorado.edu/laguna/articles/tabucycle.html, 2005.
[11] Glover, F., Tabu search, Part II, *ORSA J. Comput.*, 2, 4, 1990.
[12] Laguna, M., Barnes, J. W., and Glover, F., Intelligent scheduling with tabu search: an application to jobs with linear delay penalties and sequence dependent setup costs and times, *J. Appl. Intell.*, 3, 159, 1993.
[13] Glover, F. and Laguna, M., Target analysis to improve a tabu search method for machine scheduling, *The Arabian J. Sci. Eng.*, 16, 239, 1991.
[14] Laguna, M. and Glover, F., Integrating target analysis and tabu search for improved scheduling systems, *Expert Syst. Appl.*, 6, 287, 1993.
[15] LOLIB, http://www.iwr.uni-heildelberg.de/iwr/comopt/soft/LOLIB/LOLIB.html, 1997.