INFORMS TutORials in Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Tabu and Scatter Search: Principles and Practice

Manuel Laguna

To cite this entry: Manuel Laguna. Tabu and Scatter Search: Principles and Practice. *In* INFORMS TutORials in Operations Research. Published online: 19 Oct 2018; 130–157.
https://doi.org/10.1287/educ.2018.0181

https://pubsonline.informs.org/series/educ/

TUTORIALS IN OPERATIONS RESEARCH
 | ISBN 978-0-9906153-2-3
https://doi.org/10.1287/educ.2018.0181

Tabu and Scatter Search: Principles and Practice

Manuel Laguna
Leeds School of Business, University of Colorado Boulder, Boulder, Colorado 80309
Contact: laguna@colorado.edu, https://orcid.org/0000-0002-8759-5523 (ML)

Abstract This tutorial focuses on the metaheuristics known as tabu search and scatter search. The goal is to explore the principles and connections associated with these methodologies and their applications in practice. Tabu search has dramatically changed our ability to solve a host of problems in applied science, business, and engineering. The adaptive memory designs of tabu search have provided useful alternatives and supplements to the types of memory embodied in other metaheuristic approaches. We also explore the evolutionary approach called scatter search, which originated from strategies for creating composite decision rules and surrogate constraints.

Supplemental material is available at https://doi.org/10.1287/educ.2018.0181.

Keywords tabu search • scatter search • metaheuristics • local search • evolutionary programming

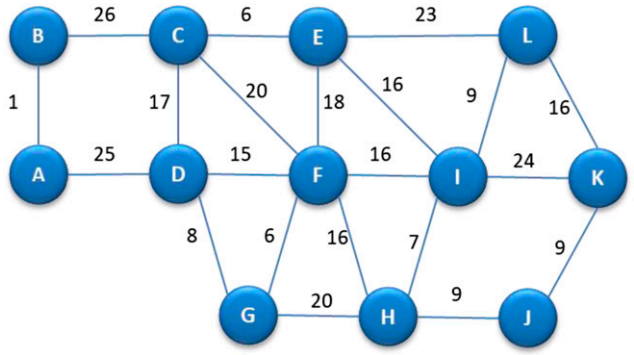
1. Introduction

The word "heuristic" derives from the Greek *heurisken*, which means "to find" or "to discover." In the context of optimization, heuristics refers to a class of procedures for finding solutions to decision problems. In colloquial language, the word "optimization" is often used as a synonym for "improving." However, in academic and scientific environments, optimization is the process of searching for the best possible solution to a given problem. Optimization problems are defined in such a way that they include a set (often very large or even infinite) of possible solutions and a criterion or a set of criteria to evaluate their merit. More precisely, these problems may be stated as finding the values for a set of decision variables for which one or more objective functions reach a minimum or a maximum value. Restrictions may be placed on the values of individual variables or combination of variables (Laguna and Martí [21]).

Optimization problems are found in many areas of business, engineering, and science. Some problem classes are relatively simple to solve. For instance, consider a simplified version of a problem that may arise in telecommunications in which it is desired to connect a number of customers in a network using the least amount of cable possible. Figure 1 shows a network with 12 customers (labeled A–L) and the costs of potential links. To connect all customers, it is only necessary to choose 11 links, and therefore the problem has 11 decision variables that consist of the identity of the links to be included in the solution.

The optimal solution (i.e., the connection that guarantees the minimum cable cost) can be found by a simple procedure that starts with the selection of the link with the smallest cost in the network. The remaining links are sequentially chosen to minimize the increase in total cost at each step, where the links considered meet exactly one customer from those that are endpoints of links previously chosen. The resulting solution is known as a *tree*, which is a set of links that contains no cycles—that is, that contains no paths that start and end at the same customer (without retracing any links). In the example of Figure 1, the first link to be added to the solution is A-B, with a cost of 1. The A-D link is chosen next to minimize the additional cost

Figure 1. (Color online) Customer network with 12 nodes.



of connecting a new customer. The link has a cost of 25, resulting in a total cost of 26 to connect A, B, and D. Now, customers C, F, or G can be connected. Cost minimization dictates that link D-G should be added, resulting in customer G to be connected to the partial solution with a total cost of $1 + 25 + 8 = 34$. The next candidates to be connected to the current tree are customers C, F, and H. The link with the minimum cost between the ends of the current tree and the candidate customers is G-F, with a cost of 6. After adding this link, the total cost is 40. Subsequent links are added in a similar fashion, resulting in the tree shown in Figure 2, which has a total connection cost of 112.

This problem is known in the operations research literature as the *minimum spanning tree problem*, and the procedure applied to the example in Figure 1 that resulted in the tree shown in Figure 2 has been shown to be optimal. The procedure avoids cycles because it is easy to verify that no optimal solution of the problem includes a cycle, because a cycle adds an unnecessary link and therefore additional cost (assuming that all costs are strictly positive). Note that an alternative optimal solution is available in which the F-H link is replaced with the F-I link. Because ties are arbitrarily broken, either solution is acceptable and optimal. The procedure is simple and can be applied to very large networks.

Unfortunately, the great majority of optimization problems are not as easy to solve optimally as the minimum spanning tree. In fact, in some cases, something that seems to be a simple change to a problem definition may turn the problem from easy to hard. For instance, if the problem of finding the minimum-cost connection of all customers is changed to finding the minimum-cost connection of a subset of customers, the problem becomes extremely difficult. The resulting problem is known as the *minimum k-tree*, which consists of finding a tree with k links so that the sum of the costs is minimized. Figure 3(a) shows the solution that is obtained from the application of the procedure described above for the case of $k = 4$.

Figure 2. (Color online) Minimum spanning tree.

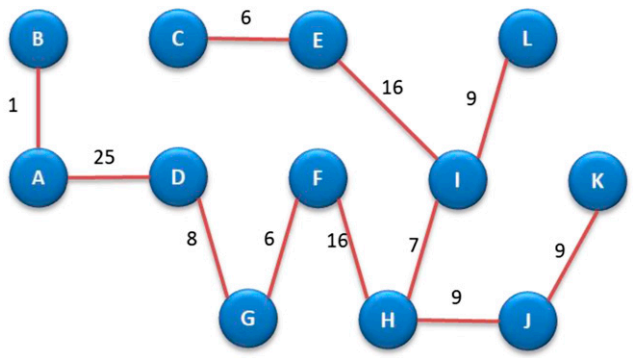
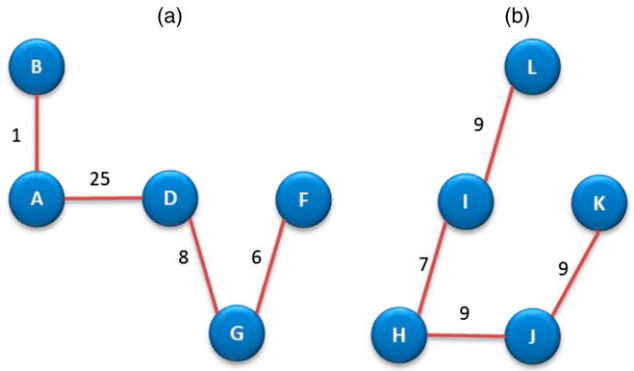


Figure 3. (Color online) Heuristic solution (a) vs. optimal solution (b).



However, this solution (with a cost of 40) is not optimal for the minimum k-tree problem. The optimal solution is shown in Figure 3(b) and has a total cost of 34. Hence a solution procedure that is exact (i.e., optimal) for the minimum spanning problem becomes a heuristic for the minimum k-tree problem.

Heuristic procedures are limited to providing a reasonably good solution to an optimization problem without guaranteeing that such a solution is optimal. The computational time of an exact method (one that guarantees finding an optimal solution) is typically orders of magnitude larger than the time employed by simple heuristics. Metaheuristic[1] methodologies provide a compromise between simple heuristics and exact methods. Optimization procedures that are based on metaheuristic methodologies are designed to search solution spaces and find multiple local optima in their quest for global optimality. Although metaheuristics do not guarantee that the best solution found is optimal, there is an abundance of empirical evidence in the literature that they are capable of finding solutions of high quality that in many cases are near optimal. One of the main advantages of metaheuristics is that the computational time can be controlled by the user. Therefore, they can be applied to problems where solutions are needed in real time as well as to problems that can be solved "off line," thus allowing for longer computational efforts.

Tabu search and scatter search are two well-known metaheuristic methodologies. Numerous studies have demonstrated the practical advantages of both tabu search and scatter search approaches for solving a diverse array of optimization problems from both academic and real-world settings. Scatter search contrasts with other evolutionary procedures, such as genetic algorithms, by providing unifying principles for joining solutions based on generalized path constructions and by utilizing strategic designs where other approaches resort to randomization. Additional advantages are provided by intensification and diversification mechanisms that exploit adaptive memory, drawing on foundations that link scatter search to tabu search. The main goal of this tutorial is to show the connections between tabu search and scatter search by demonstrating how they can be applied to many optimization problems found in practice.

We also discuss a search strategy called path relinking, which is relevant to both tabu and scatter search. Features that over time have been added to both tabu and scatter search by extension of their basic philosophy are captured in the path-relinking framework. From a spatial orientation, the process of generating linear combinations of a set of reference solutions (as typically done in scatter search) may be characterized as generating paths between and beyond these solutions, where solutions on such paths also serve as sources for generating additional paths. This leads to a broader conception of the meaning of creating combinations of solutions. By natural extension, such combinations may be conceived to arise by generating paths between and beyond selected solutions in a neighborhood space. Finally, we highlight key ideas and research issues associated with tabu search, scatter search, and path relinking that offer the promise of yielding future advances.

2. Tabu Search

Tabu search (TS) has its origins in Fred Glover's article "Heuristics for integer programming using surrogate constraints" (Glover [9]). This article introduced three key concepts: (1) scatter search, (2) oscillating assignment, and (3) strongly determined and consistent variables. Scatter search later developed into a separate metaheuristic methodology that is now well established in the literature. Oscillating assignment became the TS mechanism known as strategic oscillation. The concept of strongly determined and consistent variables became part of the tabu search long-term memory strategies, as described in Section 2.2. Without using the term "tabu," the knapsack heuristic in Glover [9] illustrates the use of short-term memory.

Recent applications of tabu search include the disjunctively constrained knapsack problem (Ben Salem et al. [4]); resource-constrained scheduling (Riley and Rego [31]); the edge dynamic graph-coloring problem (Hardy et al. [16]); Golomb rules (Polash et al. [27]); a heterogeneous surface-to-air missile defense battery location problem (Boardman et al. [5]); the multivehicle inventory routing problem (Archetti et al. [2]); and integrated production, inventory, and delivery problems (Li et al. [24]).

In this tutorial, we focus on the main TS elements. For a complete description of basic and advanced TS components and strategies, we refer the reader to Glover and Laguna [12]. The main TS elements are as follows:

1. Attribute-based short-term memory
2. Frequency-based long-term memory
3. Strategic oscillation
4. Path relinking

All TS implementations include a short-term memory structure. There is no tabu search without this basic element. The other elements add effectiveness and robustness as well as complexity. Our goal is to introduce the four elements in designs with as few adjustable parameters as possible, because the number of parameters is a good indicator of the level of complexity of a search procedure. In general, the fewer parameters, the less complex a procedure is.

2.1. Short-Term Memory

Tabu search can be applied directly to verbal or symbolic statements of many kinds of decision problems, without the need to transform them into mathematical formulations. Nevertheless, it is useful to introduce mathematical notation to express a broad class of these problems. We characterize this class of problems as follows:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{subject to} & x \in X, \end{array}$$

where the set X summarizes constraints on the vector of decision variables x, including those that compel all or some components of x to receive discrete values. The problem of interest may not have an easy mathematical representation, but the verbal stipulations could be coded as rules.

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 $x \in X$ 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*. Hence, x' is said to be a neighbor of x.

In a minimization problem, *best improving* refers to the strategy of identifying $x^* \in N(x)$ such that $f(x^*) < f(x')$ for all $x' \in N(x)$. The move value is the change in the objective function value from the current solution to the solution after the move (i.e., $v = f(x^*) - f(x)$). Therefore, $v < 0$ for improving moves and $v \geq 0$ for nonimproving moves. In steepest descent, the procedure makes only improving moves, and it stops when no improving move is available

in the neighborhood of the current solution. That is, the best neighbor of a solution becomes the new current solution as long as the move reduces the objective function value. Algorithm 1 summarizes the steepest descent procedure.

Algorithm 1 (Steepest Descent)

1. $x \leftarrow$ initial solution
2. **do**
3. identify $x^* \in N(x)$
4. $v = f(x^*) - f(x)$
5. **if** $v < 0$ then $x \leftarrow x^*$
6. **while** $v < 0$

An initial solution is generated in line 1, typically by means of a constructive procedure that takes advantage of context information. The do-loop is repeated as long as the move value is strictly negative (see line 6). The solution x^* in line 3 is the best in the neighborhood of x. The search moves to x^* only if this solution is strictly better than the current solution x (see line 5). The procedure delivers x as the best solution found, which is a local optimal point in the neighborhood $N(x)$.

Despite its attractiveness, in certain settings the best-improving strategy is sometimes impractical because it may be computationally too expensive, such as when $N(x)$ is large or each element is costly to retrieve or evaluate. The relevance of choosing good solutions from current neighborhoods is magnified when the guidance mechanisms of tabu search are introduced to go beyond the locally optimal termination point of a descent method. Thus, an important first-level consideration for tabu search is to determine an appropriate *candidate list strategy* for narrowing the examination of elements of $N(x)$ in order to achieve an effective trade-off between the quality of x^* and the effort expended to find it. A popular candidate list strategy is known as the *first-improving strategy*. This strategy is such that neighbors are scanned in a predetermined order and the search moves to the first $x' \in N(x)$ for which $f(x') < f(x)$. Algorithm 2 summarizes the first-improving descent.

Algorithm 2 (First-Improving Descent)

1. $x \leftarrow x^* \leftarrow$ initial solution
2. **do**
3. **for all** $x' \in N(x)$
4. **if** $f(x') < f(x)$ then $x^* \leftarrow x'$ and break
5. $v = f(x^*) - f(x)$
6. **if** $v < 0$ then $x \leftarrow x^*$
7. **while** $v < 0$

In the first-improving descent, the best solution found in each iteration is x^*. This solution can be either x or an improved neighbor of x. If x^* is an improved neighbor of x, then the search moves to that solution (line 6). Otherwise, the search ends (line 7). Note that the exploration of the neighborhood of x stops when the first improved solution is found (see line 4). If no improved solution exists in $N(x)$, then x^* is equal to x, $v = 0$ (line 5), and the search terminates.

In contrast to a descent method, TS chooses the "best neighbor" of x even when $v \geq 0$. The guidance mechanisms of TS determine the definition of best. The core guidance mechanism in tabu search is the short-term memory (STM). STM functions give the search the opportunity to continue beyond local optima, by allowing the execution of nonimproving moves coupled with the modification of the neighborhood structure of subsequent solutions. The modified neighborhood, denoted by $N^*(x)$, is the result of maintaining a selective history of the states encountered during the search. Algorithm 3 outlines an STM tabu search based on first-improving descent.

An STM tabu search keeps track of three different solutions: (1) the current solution x, (2) the best solution in the neighborhood of x (i.e., x^*), and (3) the best solution found during the search (x^{best}). Initially, the current solution and the best solution found during the search are the same (line 1). The do-loop is repeated until a termination criterion is satisfied (lines 2–10). Typical termination criteria are total number of iterations, iterations without improvement, and total execution time. Each iteration starts with the search for the best neighbor of x. Initially, the objective function value of the best neighbor is assigned a large positive value (line 3). As the neighborhood is scanned (line 4), the best neighbor is updated as appropriate (line 5). If a neighbor is found that is better than the current solution, then the scanning of the neighborhood ends (line 6). If the best neighbor is better than the best solution found so far, then the best overall solution is updated (line 7). Regardless of the move direction (i.e., improving or nonimproving), the search moves to the best neighbor (line 8). The short-term memory structure is updated in line 9.

Algorithm 3 (STM First-Improving Tabu Search)

1. $x \leftarrow x^{\text{best}} \leftarrow$ initial solution
2. **do**
3. $\quad f(x^*) = \infty$
4. $\quad$ **for all** $x' \in N^*(x)$
5. $\quad\quad$ **if** $f(x') < f(x^*)$ then $x^* \leftarrow x'$
6. $\quad\quad$ **if** $f(x') < f(x)$ then break
7. $\quad$ **if** $f(x^*) < f(x^{\text{best}})$ then $x^{\text{best}} \leftarrow x^*$
8. $\quad x \leftarrow x^*$
9. $\quad$ update STM
10. **while** termination criterion is not satisfied

The STM structure serves to identify elements that are excluded from $N(x)$ to form $N^*(x)$. The exclusion of neighbor solutions is done through rules that classify certain moves as tabu. 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. A TS process based on STM 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. From a practical standpoint, the method will characteristically identify an optimal or near-optimal solution long before a substantial portion of X is examined.

The approach of storing complete solutions (*explicit* memory) generally consumes a massive amount of space and time when applied to each solution generated. Therefore, instead of recording full solutions, TS memory structures are based on recording attributes (*attributive* memory). In addition, STM is based on the most recent history of the search trajectory (*recency-based* memory). In recency-based attributive memory, selected attributes that occur in solutions recently visited are labeled *tabu active*, and moves that contain tabu-active elements, or particular combinations of these attributes, are those that are classified as 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.

The number of iterations (i.e., executed moves) that force an attribute to remain tabu active is known as the *tabu tenure*. Tabu tenure can be static (i.e., fixed throughout the search) or dynamic (i.e., vary either systematically or probabilistically). Aspiration criteria are introduced in tabu search to determine when tabu activation rules can be overridden, thus removing a tabu classification otherwise applied to a move. The most common aspiration criterion consists of removing a tabu classification from a trial move when the move yields a solution better than the best obtained so far.

The updating of the STM structure in line 9 of Algorithm 3 consists of recording the attribute or attributes associated with the move from x to x^*. The updated STM determines

what moves are available (i.e., are not classified tabu) in the next iteration. In other words, the STM modifies $N^*(x)$.

2.1.1. STM Example: Knapsack Problem. We illustrate the short-term memory structure of tabu search with the following instance of the classical 0-1 knapsack problem:

$$\begin{aligned} \text{Maximize} \quad & 10x_1 + 14x_2 + 9x_3 + 8x_4 + 7x_5 + 5x_6 + 9x_7 + 3x_8 \\ \text{subject to} \quad & 7x_1 + 12x_2 + 8x_3 + 9x_4 + 8x_5 + 6x_6 + 11x_7 + 5x_8 \leq 38, \\ & x_j = \{0, 1\} \quad \text{for } j = 1, \ldots, 8. \end{aligned}$$

An initial solution may be constructed in several different ways. For instance, a common greedy approach is to select items (i.e., setting variable values to 1), one by one, in decreasing order of their bang-for-buck ratio. The process ends when adding one more item exceeds the capacity of the knapsack. The bang for buck is the ratio of profit to weight, where profits are the objective function coefficients and weights are the constraint coefficients. For simplicity, the variables in the example are indexed by their bang-for-buck ratio. That is, x_1 is the variable with the highest ratio and x_8 is the variable with the lowest ratio. For illustration purposes, we are going to use the initial solution shown in Table 1. This solution will allow us to show some TS features as well as some design issues.

A typical neighborhood for binary problems consists of all the solutions that can be reached by changing the value of a single variable from either 0 to 1 or 1 to 0. This change is also known as a "flip" move. For constrained problems, such as the knapsack, this neighborhood may contain infeasible solutions. In fact, from the initial solution, all the flips from 0 to 1 lead to infeasible solutions, whereas all the moves from 1 to 0 lead to feasible but inferior solutions. This is illustrated in Table 2.

Because in this initial design we do not allow visiting infeasible solutions, there is no improving move in the neighborhood of the initial solution. Under this restriction, the best move (as measured by the largest profit) is the last in the table and consists of changing the value of x_8 from 1 to 0. We start with the following TS design:

- *Move*: Find the smallest non-tabu-active j such that $x_j = 0$ and set $x_j = 1$, as long as the knapsack constraint is not violated. If no such a move is available, then find the largest non-tabu-active j such that $x_j = 1$ and set $x_j = 0$.
- *Tabu attribute*: The index of the variable whose value was changed in the most recent move.
- *Tabu activation rule*: A tabu-active variable is not allowed to change values for *Tabu-Tenure* iterations.
- *Aspiration criterion*: Remove the tabu-active classification of a move that leads to the improvement of the best solution found so far.

Table 1. Initial solution.

Variable	Profit	Weight	Ratio	Value	Profit	Weight
1	10	7	1.43	1	10	7
2	14	12	1.17	0	0	0
3	9	8	1.13	0	0	0
4	8	9	0.89	1	8	9
5	7	8	0.88	1	7	8
6	5	6	0.83	1	5	6
7	9	11	0.82	0	0	0
8	3	5	0.60	1	3	5
Total	—	—	—	—	33	35

Table 2. Neighborhood of initial solution.

No.	Move	Neighbor	Profit	Weight	Feasible?
1	$x_1 = 0$	(4, 5, 6, 8)	23	28	Yes
2	$x_2 = 1$	(1, 2, 4, 5, 6, 8)	47	47	No
3	$x_3 = 1$	(1, 3, 4, 5, 6, 8)	42	43	No
4	$x_4 = 0$	(1, 5, 6, 8)	25	26	Yes
5	$x_5 = 0$	(1, 4, 6, 8)	26	27	Yes
6	$x_6 = 0$	(1, 4, 5, 8)	28	29	Yes
7	$x_7 = 1$	(1, 4, 5, 6, 7, 8)	42	46	No
8	$x_8 = 0$	(1, 4, 5, 6)	30	30	Yes

The moves that we define here are convenient as a way to illustrate how tabu search works. In an actual implementation, however, in iterations where a variable could be set to 1, this variable should be chosen to maximize the increase in profit. Using the design outlined above and a tabu tenure of 2 results in the moves shown in Table 3.

The "Current solution" column shows the indexes of the variables in the solution before the move in the current iteration is executed. The profit and the weight corresponding to the current solution are shown next. The "Tabu active" column shows the indexes of the variables that are tabu active. Each index appears for two iterations. It starts in the first position and moves one position in each iteration until it is removed from the list. The "Move" column indicates the variable that has been chosen to flip its value. The best solution is found in the third iteration and has a profit of 39 (and it is shown in bold in Table 3).

The search trajectory in Table 3 exhibits a *cycled* condition. It is said that a TS has cycled when the same sequence of moves repeats under the same tabu-active conditions and tabu tenure. Under this definition, the cycling starts in iteration 11. In this iteration, the same solution as iteration 5 is reached with the same tabu-active conditions. Note that although the solutions in iterations 3 and 9 are also the same, we cannot say that the search has cycled because the tabu-active conditions are not the same. In this example, the conditions are such that they lead to the same move in both iterations, making it seem as though the cycle starts in iteration 3.

Researchers have dealt with the issues of cycles in tabu search in several different ways. The most common include testing to determine an appropriate static tabu tenure, using a dynamic tabu tenure, and strategically varying the tenure during the search. Dynamic tabu tenures are

Table 3. Iterations of the STM TS with $TabuTenure = 2$.

Iteration	Current solution	Profit	Weight	Tabu active	Move
1	(1, 4, 5, 6, 8)	33	35		8
2	(1, 4, 5, 6)	30	30	8	3
3	(1, 3, 4, 5, 6)	**39**	38	3 8	6
4	(1, 3, 4, 5)	34	32	6 3	8
5	(1, 3, 4, 5, 8)	37	37	8 6	5
6	(1, 3, 4, 8)	30	29	5 8	6
7	(1, 3, 4, 6, 8)	35	35	6 5	8
8	(1, 3, 4, 6)	32	30	8 6	5
9	(1, 3, 4, 5, 6)	39	38	5 8	6
10	(1, 3, 4, 5)	34	32	6 5	8
11	(1, 3, 4, 5, 8)	37	37	8 6	5
12	(1, 3, 4, 8)	30	29	5 8	6
13	(1, 3, 4, 6, 8)	35	35	6 5	8
14	(1, 3, 4, 6)	32	30	8 6	5
15	(1, 3, 4, 5, 6)	39	38	5 8	6

Note. The best solution is found in the third iteration and has a profit of 39 (in bold).

typically implemented by randomly drawing the tenure from a predetermined range. That is, a tenure value is drawn from a uniform distribution, and it is assigned to the attribute that becomes tabu active after a move. Strategic changes to the tabu tenure are at the core of what is known as reactive tabu search (Battiti and Tecchiolli [3]). In our example, a tabu tenure of 3 is sufficient to break the cycle shown in Table 3. The new search trajectory is shown in Table 4.

The search in Table 4 identifies two solutions with a profit of 41 (in bold) in iterations 6 and 9. Figure 4 shows the trajectory of the objective function values of the current solutions in Tables 2 and 3. The effect of the increase in the tabu tenure can be noticed after the fourth iteration. In general, the range of the objective function values increases with the tabu tenure.

This example is based on very simple flip moves. The neighborhood defined by the move mechanism is a design choice of critical importance for the performance of the method. Often, the neighborhood is defined by several move types that may vary in complexity. The attributes to be used for the tabu classification of moves depend on those choices and they influence the performance of the search. For instance, in this example, a slightly more complex move would consist of flipping the values of two variables. Instead of n neighbors, where n is the number of variables, this move mechanism generates $m(n-m)$ neighbors, where m is the number of variables set to 1. The tabu attributes are the indices of the variables that change values. The tabu activation rule is the same as before; that is, a tabu-active variable is not allowed to change values for *TabuTenure* iterations. The tabu tenure could be set in such a way that the variable flipping from 1 to 0 is assigned a longer tenure than the variable flipping from 0 to 1. This tabu tenure setting recognizes that in a typical knapsack problem there will be more variables outside the knapsack (i.e., with their values set to 0) than those inside the knapsack (i.e., with their values set to 1). Therefore, if an item is taken from the knapsack (i.e., its corresponding variable value is switched from 1 to 0), the tabu tenure keeps this item outside longer than when an item enters the knapsack. This illustrates that the flexibility of the TS methodology allows the analyst to be creative and take full advantage of context information.

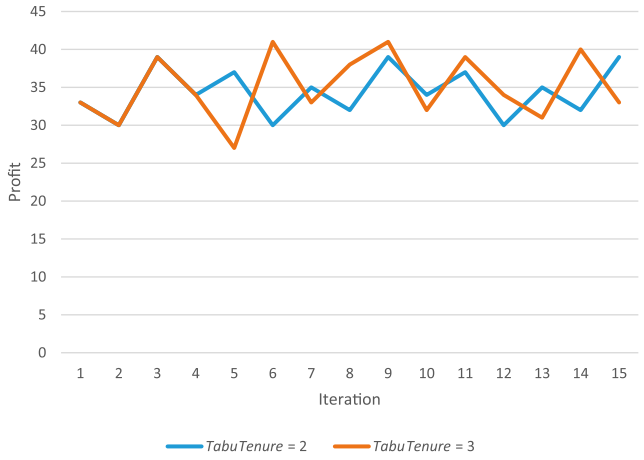
2.1.2. STM Example: Linear Ordering Problem. In this example, we show how an STM can be configured for moves that consist of changing the position of elements in an array. We also use the example to discuss the notion of employing efficient move evaluations in TS implementations. Given a matrix of weights, the *linear ordering problem* (LOP) consists of finding a (simultaneous) permutation p of the columns (and rows) to maximize the sum of the weights in the upper triangle. Consider the LOP instance in Table 5.

Table 4. Iterations of the STM TS with *TabuTenure* = 3.

Iteration	Current solution	Profit	Weight	Tabu active	Move
1	(1, 4, 5, 6, 8)	33	35		8
2	(1, 4, 5, 6)	30	30	8	3
3	(1, 3, 4, 5, 6)	39	38	3 8	6
4	(1, 3, 4, 5)	34	32	6 3 8	5
5	(1, 3, 4)	27	24	5 6 3	2
6	(1, 2, 3, 4)	**41**	36	2 5 6	4
7	(1, 2, 3)	33	27	4 2 5	6
8	(1, 2, 3, 6)	38	33	6 4 2	8
9	(1, 2, 3, 6, 8)	**41**	38	8 6 4	3
10	(1, 2, 6, 8)	32	30	3 8 6	5
11	(1, 2, 5, 6, 8)	39	38	5 3 8	6
12	(1, 2, 5, 8)	34	32	6 5 3	8
13	(1, 2, 5)	31	27	8 6 5	3
14	(1, 2, 3, 5)	40	35	3 8 6	5
15	(1, 2, 3)	33	27	5 3 8	4

Note. The search identifies two solutions with a profit of 41 (in bold) in iterations 6 and 9.

Figure 4. (Color online) Profit values for the STM TS with tabu tenure of two and three iterations.



The objective function value associated with the solution $p = (1, 2, 3, 4, 5, 6, 7)$ is $f(p) = 78$, corresponding to the sum of the 21 values in the upper triangle. The elements in the permutation correspond to columns (and rows) of the matrix. Suppose that a tabu search explores the solution space by moving an element $p(i)$ from its current position i to a position j, between elements $p(j-1)$ and $p(j)$. In permutation problems, these are commonly referred to as *insert* moves. For instance, inserting $p(6) = 6$ in position 2 results in $p' = (1, 6, 2, 3, 4, 5, 7)$ with $f(p') = 86$.

Because TS determines where to move via neighborhood search, it is important to be able to calculate move values with the fewest possible number of operations. One naïve way of calculating a move value consists of executing the proposed move and calculating, from scratch, the objective function of the resulting neighbor solution. Then, the value of the move is obtained as the difference between the objective function value of the current solution and the objective function value that results after executing the move. In many settings, this process is inefficient and unnecessary. Clever move value calculations can significantly improve the efficiency of the neighborhood exploration. In the case of the LOP, the move value involves only a portion of the entries in the column (and corresponding row) associated with the element that is changing positions. For instance, for the insertion of $p(6) = 6$ in position 2 to transform p into p', only the entries in italic in Table 6 are needed to calculate the move value v.

The move value in this case is $v = 1 - 4 + 6 - 0 + 2 - 6 + 13 - 4 = 8$. The objective function value is $f(p') = f(p) + v = 78 + 8 = 86$. A short-term memory structure for this problem may use the index of the element that was inserted in a new position as the tabu attribute. The tabu activation rule may consist of classifying as tabu any insertion of a tabu-active element.

Table 5. Instance of the linear ordering problem.

p	1	2	3	4	5	6	7
1	0	12	5	3	1	8	3
2	6	0	3	6	4	4	2
3	8	5	0	5	7	0	3
4	−2	7	2	0	−3	6	0
5	8	0	3	−1	0	4	1
6	9	1	6	2	13	0	4
7	2	9	4	−5	8	1	0

Table 6. Instance of the linear ordering problem.

p	1	2	3	4	5	6	7
1	0	12	5	3	1	8	3
2	6	0	3	6	4	*4*	2
3	8	5	0	5	7	*0*	3
4	−2	7	2	0	−3	*6*	0
5	8	0	3	−1	0	*4*	1
6	9	*1*	*6*	*2*	*13*	0	4
7	2	9	4	−5	8	1	0

Note. Entries in italic are needed to calculate the move value v.

Because of the nature of the moves and the tabu classification, elements during their tabu-active period may change positions. To illustrate this point, consider the insertion of $p(6) = 6$ in position 2 that results in $p' = (1, 6, 2, 3, 4, 5, 7)$. Element 6 becomes tabu active, and therefore it is not allowed to participate in insertions for the next five iterations. However, other non-tabu-active elements may be inserted in either position 2 or position 1, causing element 6 to shift to position 3.

2.2. Long-Term Memory

In many applications, the STM components are sufficient to produce very high-quality solutions. However, in general, TS becomes significantly stronger by including longer-term memory and its associated strategies. Long-term memory (LTM) plays an important role in creating the right balance between intensification and diversification of a tabu search. 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, on the other hand, encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before.

LTM is often implemented using a frequency-based approach. Frequency-based memory provides a type of information that complements the information provided by recency-based memory, broadening the foundation for selecting preferred moves. Similar to recency, frequency often is weighted or decomposed into subclasses by considering solution quality.

Frequencies consist of ratios, whose numerators represent either *transition* counts or *residence* counts. A transition count is the number of iterations where an attribute changes (enters or leaves) the solutions visited. A residence count is the number of iterations where an attribute belongs to solutions visited. 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 iterations), (2) the sum (or average) of the numerators, and (3) the maximum numerator value.

The use of longer-term memory does not require long solution runs before its benefits become visible. Often, its improvements begin to be manifest in a relatively modest length of time and can allow solution efforts to be terminated somewhat earlier than otherwise possible, as a result of finding very high-quality solutions within an economical time span. The chance of finding still better solutions as time grows—in the case where an optimal solution has not already been found—is enhanced by using longer-term TS memory in addition to short-term memory.

Residence frequencies and transition frequencies sometimes convey related information but in general carry different implications. For example, residence measures, in contrast to transition measures, are not concerned with whether an attribute changes in moving from inferior solutions to better solutions. A high residence frequency may indicate that an attribute is highly attractive if the domain consists of high-quality solutions, or it may indicate the opposite, if the domain consists of low-quality solutions.

Frequency-based memory is used to define penalty and incentive values to modify the evaluation of moves and therefore determine which moves are selected. In a minimization problem, a penalty function for the move value has the following mathematical form:

$$v' = \begin{cases} v & \text{if } v < 0, \\ v(1 + wq) & \text{if } v \geq 0. \end{cases}$$

Here, v is the original move value, w is the penalty factor, q is the frequency ratio, and v' is the modified move value. The modified move value is the same as the move value for improving moves. For nonimproving moves, the modified move value is increased by a factor of wq. Values of $w > 0$ are used to control the magnitude of the penalization for nonimproving moves that incorporate elements that have been part of solutions visited in the past. The penalty makes those moves less attractive. A value of $w < 0$ could be used to provide an incentive, because the resulting v' would make the move more attractive.

Long-term memory is linked to the notion of *strongly determined* and *consistent* variables. A strongly determined variable is one that cannot change its value in a given high-quality solution without seriously degrading quality or feasibility, whereas a consistent variable is one that frequently takes on a specific value (or a highly restricted range of values) in good solutions. The development of useful measures of "strength" and "consistency" is critical to exploiting these notions, particularly by accounting for trade-offs determined by context. However, straightforward uses of frequency-based memory for keeping track of consistency, sometimes weighted by elements of quality and influence, have produced methods with very good performance outcomes.

For instance, a penalty/incentive function based on frequency ratios may be used in a restarting procedure to generate initial solutions based on penalty/incentive functions. The function may be designed to assess a penalty on consistent variables in order to induce diversification.

2.2.1. LTM Example: Knapsack Problem. Suppose that we add a residency LTM to the STM TS for the knapsack problem described in Section 2.1.1. The memory accumulates the number of times each item is included in the knapsack. That is, the memory is a count of the number of times that a variable takes on the value of 1. This is a simple LTM that uses frequency information:

- *Frequency-based memory*: Record the number of times (f_j) that variable j has been set to 1 throughout the search. The score of variable j is defined as $j + f_j$.
- *Move*: Find the smallest non-tabu-active j such that $x_j = 0$ and set $x_j = 1$, as long as the knapsack constraint is not violated. If no such a move is available, find the non-tabu-active j with the largest score such that $x_j = 1$ and set $x_j = 0$.

The rationale behind this LTM structure is to penalize variables that stay in the knapsack for a large number of iterations (i.e., the strongly determined variables). The score of strongly determined variables increases, and they are eventually forced out of the solution. Variables that are added to the knapsack are not penalized, keeping the aggressive nature of the search. Alternatively, and for a number of diversification iterations, the variables added to the knapsack could be also chosen using their score. In such a variant, we first look for a variable to add that is currently set to 0 and that has the smallest score value. Table 7 shows the search trajectory of the version that uses the score for selecting variables to set to 0 and for selecting variables to set to 1.

The first six iterations in Table 7 are the same as the iterations in Table 4. The LTM changes the direction of the search in iteration 7. An improved solution with a profit of 42 (in bold) is found in iteration 8.

Frequency information accumulated in an LTM structure can also be used to restart the search. Restarting means that the short-term memory is erased and the long-term memory is

Table 7. Iterations of the TS with long-term memory.

Iteration	Current solution	Profit	Weight	Tabu active	Move
1	(1, 4, 5, 6, 8)	33	35		8
2	(1, 4, 5, 6)	30	30	8	3
3	(1, 3, 4, 5, 6)	39	38	3 8	6
4	(1, 3, 4, 5)	34	32	6 3 8	5
5	(1, 3, 4)	27	24	5 6 3	2
6	(1, 2, 3, 4)	41	36	2 5 6	4
7	(1, 2, 3)	33	27	4 2 5	7
8	(1, 2, 3, 7)	**42**	38	7 4 2	3
9	(1, 2, 7)	33	30	3 7 4	5
10	(1, 2, 5, 7)	40	38	5 3 7	1
11	(2, 5, 7)	30	31	1 5 3	6
12	(2, 5, 6, 7)	35	37	6 1 5	7
13	(2, 5, 6)	26	26	7 6 1	8
14	(2, 5, 6, 8)	29	31	8 7 6	1
15	(1, 2, 5, 6, 8)	39	38	1 8 7	5

Note. Bold entry in iteration 8 represents an improved solution with a profit of 42.

used to generate a new initial solution. For instance, an initial solution constructed with the bang-for-buck ratio (r) results in choosing the first four items for a total profit of 41 and a weight of 36. Let us define q_j as the percentage of the iterations in which variable j was set to 1. Then a penalized bang-for-buck ratio r' can be calculated as follows:

$$r'_j = r_j(1 - q_j).$$

The original bang-for-buck ratio is r, and q is the frequency ratio. In this penalty function, the implicit value of w is -1. The function penalizes items with high-frequency ratios (i.e., the consistent variables). In this example, long-term memory is being used to induce diversification by attempting to construct a solution that has not been visited so far. Table 8 shows the calculation of the modified bang-for-buck ratios for q values calculated from the iterations in Table 4.

The solution in Table 8 is generated by choosing the variables in the following order: x_7, x_4, x_3, and x_6. The penalized ratios give priority to x_7, which was never part of the solution. On the other hand, x_1 has the lowest priority given that it was set to 1 in all the solutions in Table 4. The solution in Table 8, which was not visited in the search shown in Table 4, becomes the restarting point for a new search.

2.3. Strategic Oscillation

Strategic oscillation is closely linked to the origins of tabu search, and it provides a means to achieve an effective interplay between intensification and diversification over the intermediate to long term. Strategic oscillation operates by orienting moves in relation to an oscillation boundary, which represents a point where a search would normally stop. In constrained search spaces, the oscillation is typically defined around the feasibility boundary (i.e., the one that separates feasible from infeasible solutions). Instead of stopping when this boundary is reached, however, the rules for selecting moves are modified to permit the region defined by the feasibility boundary 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.

The process of repeatedly approaching and crossing the feasibility boundary 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.

Table 8. Initial solution by penalized bang-for-buck ratio.

Variable	Profit	Weight	r	q	r'	Value	Profit	Weight
1	10	7	1.43	1.00	0.00	0	0	0
2	14	12	1.17	0.67	0.39	0	0	0
3	9	8	1.13	0.60	0.45	1	9	8
4	8	9	0.89	0.40	0.53	1	8	9
5	7	8	0.88	0.53	0.41	0	0	0
6	5	6	0.83	0.47	0.44	1	5	6
7	9	11	0.82	0.00	0.82	1	9	11
8	3	6	0.60	0.33	0.40	0	0	6
Total	—	—	—	—	—	—	31	34

The implementation of strategic oscillation entails the selection of a proximity measure and the amplitude of the oscillation. The proximity measure assigns a numerical value to moving toward or away from the feasibility boundary. This measure serves as guidance to create the oscillation pattern. The amplitude determines the point at which the search is directed to turn around once it has crossed the feasibility boundary in either direction.

Proximity measures are defined in reference to the problem constraints. For instance, for a combinatorial optimization problem consisting of selecting k out of a given number elements, a proximity measure t may be the difference between the number of elements that have been selected and k. Feasible solutions are those for which $t = 0$. Solutions with $t > 0$ have too many elements and solutions with $t < 0$ have too few. Moves can be designed to keep the search at $t = 0$ or to create an oscillation pattern around $t = 0$.

The oscillation amplitude is controlled by a number of moves m. If a move is made that causes the search to cross the feasibility boundary, then $m - 1$ is the number of additional moves that the search is allowed to make in the same direction (i.e., moving away from the feasibility boundary) before turning around.

In combinatorial optimization problems that require the selection of k elements, the rule to delete elements from the solution will typically be different in character from the one used for adding elements. In other words, one rule is not simply the inverse of the other. Rule differences are features of strategic oscillation that provide an enhanced heuristic vitality. The application of different rules may be accompanied by crossing a boundary to different depths on different sides. An option is to approach and retreat from the boundary while remaining on a single side, without crossing (i.e., electing a crossing of "zero depth").

In both one-sided and two-sided oscillation approaches, it is frequently important to spend additional search time in regions close to the feasibility boundary, and especially to spend time at the boundary itself. In problems for which feasibility is determined by a set of constraints, vector-valued functions 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 constraint set.

2.3.1. Strategic Oscillation Example: Knapsack Problem. The feasibility boundary in the knapsack example is defined by the capacity of the knapsack, which in the example introduced in Section 2.1.1 is a total weight of 38. To create a simple oscillation around the feasible boundary, we allow the search to cross to the infeasible region, and m is set to 1. The proximity measure t is defined as the difference between the total weight of the items in the knapsack and the knapsack capacity. Therefore, $t > 0$ indicates an infeasible solution and $t \leq 0$ corresponds to feasible solutions. The procedure operates with the following rules:

- When approaching the feasibility boundary from the feasible region, select the best non-tabu-feasible move. If no feasible move is available, then select the non-tabu move that improves profit the most.

• When approaching the feasibility boundary from the infeasible region, select the non-tabu move that removes the variable with the smallest bang-for-buck ratio.

The short-term memory structure is the same as the one defined in Section 2.1.1. Therefore, the move selections have to be done by considering the tabu classifications. Table 9 shows 15 iterations of TS with strategic oscillation. The initial solution is the one shown in Table 1.

The patterns in Figure 5 show the crossing of the feasibility boundary when strategic oscillation is used. The TS design that searches only in the feasible region produces solutions whose weight is below the dotted line. Strategic oscillation is particularly effective in problems such as the knapsack, where the best solutions lie on or very close to the feasibility boundary.

The *educ.2018.0181.sm2.xlsx* online supplement that accompanies this chapter may be used as an interactive learning tool for tabu search. It contains Tables 3, 4, 7, and 9, and Figures 4 and 9. The columns labeled "Move" should be cleared and the TS designs should be discussed with the students. Then the students can be asked for the next move to make according to each design. The spreadsheets are set up to update the current solution and the memory structures as variables names are entered in the Move cells.

2.3.2. Strategic Oscillation Example: Maximum Diversity Problem. The problem of maximizing diversity deals with selecting a subset of k elements from a given set of n elements in such a way that the diversity among the chosen elements is maximized. The best-known problem in this class deals with maximizing the sum of the diversities of the chosen elements. That is, if d_{ij} is the measure of diversity between elements i and j, then the problem consists of maximizing $\sum_{i,j\in K} d_{ij}$, where K is the subset of chosen elements. In this context, the proximity measure t may be defined as the difference between k and the number of elements that have been selected. For an amplitude of m, t ranges between $-m$ and m, with $t = 0$ representing a feasible solution.

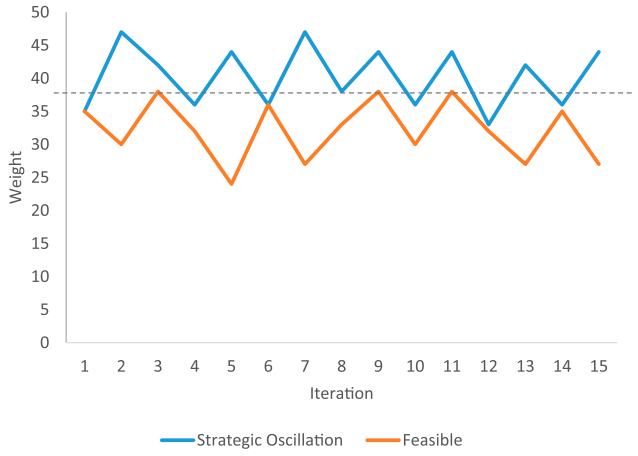
A swap move is defined as the exchange of an element currently in K with an element that is not currently in K. A swap neighborhood maintains feasibility as long as the starting solution is feasible (i.e., the solution includes k elements). By contrast, an add/delete neighborhood will not. A strategic oscillation for this problem may be defined as follows:

Table 9. Iterations of TS with strategic oscillation.

Iteration	Current solution	Profit	Weight	Feasible?	Tabu active	Move
1	(1, 4, 5, 6, 8)	33	35	Yes		2
2	(1, 2, 4, 5, 6, 8)	47	47	No	2	8
3	(1, 2, 4, 5, 6)	44	42	No	8 2	6
4	(1, 2, 4, 5)	39	36	Yes	6 8 2	3
5	(1, 2, 3, 4, 5)	48	44	No	3 6 8	5
6	(1, 2, 3, 4)	41	36	Yes	5 3 6	7
7	(1, 2, 3, 4, 7)	50	47	No	7 5 3	4
8	(1, 2, 3, 7)	42	38	Yes	4 7 5	6
9	(1, 2, 3, 6, 7)	47	44	No	6 4 7	3
10	(1, 2, 6, 7)	38	36	Yes	3 6 4	5
11	(1, 2, 5, 6, 7)	45	44	No	5 3 6	7
12	(1, 2, 5, 6)	36	33	Yes	7 5 3	4
13	(1, 2, 4, 5, 6)	44	42	No	4 7 5	6
14	(1, 2, 4, 5)	39	36	Yes	6 4 7	3
15	(1, 2, 3, 4, 5)	48	44	No	3 6 4	5

Notes. The oscillation can be observed in the "Weight" column. Every time that the weight exceeds 38, the search turns around, and moves are made to decrease the weight. A turnaround also occurs when the weight falls below 38. Figure 5 shows a graphical representation of the oscillation pattern.

Figure 5. (Color online) Weight values for the TS with and without strategic oscillation.



1. Construct a feasible solution K.
2. Add elements $j \notin K$ that increase diversity the most. Elements are added one at a time until m is reached.
3. Delete elements $j \in K$ that decrease the diversity the least. Elements are deleted one at a time until m is reached.
4. Repeat 2 and 3 until a termination criterion is satisfied.

Short-term memory tabu structures may be embedded into this simple design to avoid cycling. For instance, elements that have been added may be declared tabu active and are not allowed to be deleted during the next deletion cycle, and vice versa. Consider the following diversity values in Table 10 for a problem with $k = 5$ and $n = 10$.

Let us assume that the starting solution is given by (1, 2, 3, 4, 5) with an objective function value of 45, and let $m = 2$. Table 11 shows 16 iterations of the strategic oscillation process, where a solution is defined by a set of 10 binary variables x_j.

The first column ("Iter.") shows the iteration number, where 0 is the initial solution. The next 10 columns show the values of the binary variables, where a value of 1 indicates that the element has been chosen. The "Move" column indicates the index of the element that enters (plus sign) or leaves (minus sign) the solution. The proximity measure is given in the column labeled "t." The objective function value is shown in the last column. The solution at iteration 8 is the best feasible solution (shown in bold), with a maxSum value of 60 and $t = 0$. Note that an aspiration criterion is being used to override the tabu status. In particular, the tabu status of a move is overridden if it leads to a solution with an objective function value that is the best

Table 10. Diversity matrix for example problem.

	1	2	3	4	5	6	7	8	9	10
1		4	5	0	3	2	0	2	4	2
2			7	9	2	6	4	8	4	6
3				3	4	0	8	7	6	4
4					8	7	5	5	4	1
5						4	8	6	2	1
6							6	5	4	5
7								0	6	8
8									6	1
9										3
10										

Table 11. Search trajectory of a TS with strategic oscillation for the maximum diversity problem.

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	Move	t	MaxSum
0	1	1	1	1	1	0	0	0	0	0	+8	0	45
1	1	1	1	1	1	0	0	1	0	0	+9	1	73
2	1	1	1	1	1	0	0	1	1	0	−1	2	99
3	0	1	1	1	1	0	0	1	1	0	−5	1	81
4	0	1	1	1	0	0	0	1	1	0	−9	0	59
5	0	1	1	1	0	0	0	1	0	0	−3	−1	39
6	0	1	0	1	0	0	0	1	0	0	+6	−2	22
7	0	1	0	1	0	1	0	1	0	0	+5	−1	40
8	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**+7**	**0**	**60**
9	0	1	0	1	1	1	1	1	0	0	+3	1	83
10	0	1	1	1	1	1	1	1	0	0	−6	2	112
11	0	1	1	1	1	0	1	1	0	0	−8	1	84
12	0	1	1	1	1	0	1	0	0	0	−2	0	58
13	0	0	1	1	1	0	1	0	0	0	−4	−1	36
14	0	0	1	0	1	0	1	0	0	0	+9	−2	20
15	0	0	1	0	1	0	1	0	1	0	+10	−1	34
16	0	0	1	0	1	0	1	0	1	1		0	50

Note. The best feasible solution is found in iteration 8 (in bold).

visited so far for the corresponding t value. For instance, element 3 is added at iteration 9 even though this element is tabu (because it was deleted in the last deletion cycle). However, the addition of 3 results in the solution at iteration 10 that has the best objective function value of all those solutions with $t = 2$ that have been visited so far. The same logic is applied to allow the deletion of element 6 at iteration 10.

2.4. Path Relinking

A useful integration of intensification and diversification strategies occurs in the approach called *path relinking*. This approach generates new solutions by exploring trajectories that connect reference solutions—by starting from one of these solutions, called an *initiating solution*, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions.

Path relinking is 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 path-relinking approach subordinates all other considerations to the goal of choosing moves that introduce the attributes of the guiding solution, in order to create a "good attribute composition" in the current solution.

At each step, the procedure chooses the best move, according to the change in the objective function value, from the restricted set of moves that incorporate a maximum number of the attributes of the guiding solutions. As in other applications of TS, aspiration criteria can override this restriction to allow other moves of particularly high quality to be considered.

Membership in the reference set is determined by setting a threshold that is connected to the objective function value of the best solution found during the search. For example, reference solutions might be those whose objective function value is within 10% of the best objective function value known so far.

Path relinking can be used to generate intensification strategies by choosing reference solutions that lie in a common region or that share common features. Similarly, diversification strategies based on path relinking characteristically select reference solutions that come from different regions or that exhibit contrasting features. The initiating solution can be used to give

a beginning partial construction by specifying particular attributes as a basis for remaining constructive steps. Constructive neighborhoods can be viewed as a feasibility-restoring mechanism because a null or partially constructed solution does not satisfy all conditions to qualify as feasible. Similarly, path relinking can make use of destructive neighborhoods, where an initial solution is "overloaded" with attributes donated by the guiding solutions, and such attributes are progressively stripped away or modified until reaching a set with an appropriate composition. Destructive neighborhoods represent an instance of a feasibility-restoring function, as where an excess of elements may violate explicit problem constraints.

Path relinking consists of the following steps:

1. Identify the neighborhood structure and associated solution attributes for path relinking (possibly different from those of other TS strategies applied to the problem).
2. Select a collection of two or more reference solutions and identify which members will serve as the initiating solution and the guiding solution(s). (Reference solutions can be infeasible, such as "incomplete" or "overloaded" solution components treated by constructive or destructive neighborhoods.)
3. Move from the initiating solution toward the guiding solution(s), generating one or more intermediate solutions.

Recent applications of path relinking include facility location (Albareda-Sambola et al. [1]), vehicle routing (Shelbourne et al. [32]), and single-machine scheduling (González et al. [13]).

2.4.1. Path-Relinking Example: Knapsack Problem. Consider two solutions found during the normal operation of a tabu search. We use these solutions for a path-relinking phase with swap moves. The initiating solution is (1, 2, 4, 8) with a profit of 35 and a weight of 33. The guiding solution is (1, 3, 5, 7) with a profit of 35 and a weight of 34. We assume that these solutions were found with a TS that used simple moves consisting of flipping one variable at a time. Both of these solutions are feasible. The relinking process starts by identifying the values that are common to both solutions. These correspond to variables $x_1 = 1$ and $x_6 = 0$. Swap moves are used during the path-relinking process to keep the intermediate solutions close to the feasibility boundary. A swap in this context corresponds to adding one item to the knapsack while taking another item out of the knapsack. This means that a variable that is set to 1 in the initiating solution and to 0 in the guiding solution must be paired with a variable that is set to 0 in the initiating solution and to 1 in the guiding solution. Each time a swap is made, the process moves closer to the guiding solution by two variables. That is, two more variables in the solution after the swap match the values in the guiding solution. Table 12 shows the intermediate solutions that result from the path-relinking swaps.

The best feasible swap from the initiating solution is (3, 4), and the search moves to an intermediate solution with profit of 36 and weight of 32. In the next step, the best swap is (7, 8), resulting in a solution with profit of 42 and weight 38. After the second move, there is only one swap left to reach the guiding solution—namely, (2, 5). In this case, the process produced only two intermediate solutions; however, in general, path relinking visits more solutions during the transformation of the initiating solution into the guiding solution. In addition, the exploration in the neighborhood of an intermediate solution may result in finding improved solutions even if the relinking process does not move in the direction of those solutions.

Table 12. Path relinking moves.

Solution	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Profit	Weight	Move
Initiating	1	1	0	1	0	0	0	1	35	33	(3, 4)
1	1	1	1	0	0	0	0	1	36	32	(7, 8)
2	1	1	1	0	0	0	1	0	42	38	(2, 5)
Guiding	1	0	1	0	1	0	1	0	35	34	—

3. Scatter Search

Scatter search (SS) (Laguna and Martí [19]) is an evolutionary method that has been successfully applied to hard optimization problems. Unlike genetic algorithms, it operates on a small set of solutions and employs diversification strategies of the form proposed in tabu search (Glover and Laguna [12]), which give precedence to strategic learning based on adaptive memory, with limited recourse to randomization. The fundamental concepts and principles were first proposed in the 1970s as an extension of formulations, dating back to the 1960s, for combining decision rules and problem constraints. (The constraint combination approaches, known as surrogate constraint methods, now independently provide an important class of relaxation strategies for global optimization.) The scatter search framework is flexible, allowing the development of alternative implementations with varying degrees of sophistication.

Recent applications of scatter search include single-machine scheduling (González et al. [13]), flow shop scheduling (Riahi et al. [29]), allocation of trainees to software projects (Gharote et al. [7]), optimization of incentive contract designs (Kerkhove and Vanhoucke [17]), and biochemical systems (Remli et al. [27]).

As described in Laguna [18], scatter search consists of five methods:

1. Diversification generation
2. Improvement
3. Reference set update
4. Subset generation
5. Solution combination

The *diversification generation method* is used to generate a set of diverse solutions that are the basis for initializing the search. The most effective diversification methods are those capable of creating a set of solutions that balances diversification and quality. It has been shown that SS produces better results when the diversification generation method is not purely random and constructs solutions by reference to both a diversification measure and the objective function.

The *improvement method* transforms solutions with the goal of improving quality (typically measured by the objective function value) or feasibility (typically measured by some degree of constraint violation). The input to the improvement method is a single solution that may or may not be feasible. The output is a solution that may or may not be better (in terms of quality or feasibility) than the original solution. The typical improvement method is a local search with the usual rule of stopping as soon as no improvement is detected in the neighborhood of the current solution. There is the possibility of basing the improvement method on procedures that use a neighborhood search but that are able to escape local optimality. Tabu search, simulated annealing, and variable neighborhood search qualify as candidates for such a design. This may seem as an attractive option as a general approach for an improvement method; however, these procedures do not have a natural stopping criterion. The end result is that choices need to be made to control the amount of computer time that is spent improving solutions (by running a metaheuristic-based procedure) versus the time spent outside the improvement method (e.g., combining solutions). In general, local search procedures seem to work well, and most SS implementations do not include mechanisms to escape local optimality within the process of improving a solution.

The *reference set update method* refers to the process of building and maintaining a reference set of solutions that are used in the main iterative loop of any scatter search implementation. While there are several implementation options, this element of scatter search is independent from the context of the problem. The first goal of the reference update method is to build the initial reference set of solutions from the population of solutions generated with the diversification method. Subsequent calls to the reference update method serve the purpose of maintaining the reference set. The typical design of this method builds the first reference set by

blending high-quality solutions and diverse solutions. While choosing diverse solution, reference needs to be made to a distance metric that typically depends on the solution representation. That is, if the problem context is such that continuous variables are used to represent solutions, then diversification may be measured with Euclidean distances. Other solution representations (e.g., binary variables or permutations) result in different ways of calculating distances and in turn diversification. The updating of the reference set during the scatter search iterations is customarily done based on solution quality.

The *subset generation method* produces subsets of reference solutions that become the input to the combination method. The typical implementation of this method consists of generating all possible pairs of solutions. The scatter search framework considers also the generation of larger subsets of reference solutions; however, most SS implementations have been limited to operate on pairs of solutions. Clearly, no context information is needed to implement the subset generation method.

The *solution combination method* uses the output from the subset generation method to create new solutions. New trial solutions are the results of combining, typically two but possibly more, reference solutions. The combination of reference solutions is usually designed to exploit problem context information and solution representation. Linear combinations of solutions represented by continuous variables have been used often since they were suggested by Glover [9] in connection with the solution of nonlinear programming problems. Several proposals for combining solutions represented by permutations have also been applied (Martí et al. [25]). Path relinking, originally proposed within the tabu search methodology (Glover and Laguna [12]), has also played a relevant role in designing combination methods for scatter search implementations.

The basic scatter search framework is outlined in Algorithm 4. The search starts with the application of the diversification and improvement methods (step 1 in Algorithm 4). The typical outcome consists of a set of about 100 solutions that is referred to as the population (denoted by P). In most implementations, the diversification generation method is applied first, followed by the improvement method. If the application of the improvement method results in the shrinking of the population (as a result of more than one solution converging to the same local optimum), then the diversification method is applied again until the total number of improved solutions reaches the desired target. Other implementations construct and improve solutions, one by one, until reaching the desired population size.

The most effective form of diversification generation is one in which the solutions are not only diverse but their collective quality is better than the outcome of a purely random process. Most diversification methods in SS implementations are based on the first phase of the greedy randomized adaptive search procedure (GRASP) (Feo and Resende [7]). The first phase of GRASP consists of a semigreedy construction of solutions, as described in Resende et al. [29].

Algorithm 4 (Scatter Search Framework)

1. Diversification generation and improvement methods
2. **do**
3. Reference set update method
4. **while** new reference solution
5. Subset generation method
6. Combination method
7. Improvement method
8. Reference set update method
9. Rebuild reference set
10. **while** termination criterion is not satisfied

The main scatter search loop is shown in lines 2–10 of Algorithm 4. The input to the first execution of the reference set update method (step 3) is the population of solutions generated in step 1, and the output is a set of solutions known as the reference set (or *RefSet*). Typically,

10 solutions are chosen from a population of 100. The first five solutions are chosen to be the best solutions (in terms of the objective function value) in the population. The other five are chosen to be the most diverse with respect to the solutions in the reference set. If the diverse solutions are chosen sequentially, then the sixth solution is the most diverse with respect to the 5 best solutions that were chosen first. The seventh solution is the most diverse with respect to the first six, and so on, until the 10th solution is added to the reference set.

Choosing solutions by quality is straightforward when quality is measured by the objective function value. To choose diverse solution, an appropriate measure of distance is needed. Euclidean distance is commonly used when solutions are represented by continuous variables. Hamming distance is appropriate for two strings of equal length. The distance is given by the number of positions for which the corresponding symbols are different. This distance has been typically used for problems whose solution representation is given by binary vectors (e.g., the max-cut and knapsack problems) and permutation vectors (e.g., the traveling salesman, quadratic assignment, and linear ordering problems). Distance measures are used to calculate the minimum distance of a solution in P and all the solutions in the *RefSet*. The next population solution to be added to the *RefSet* and deleted from P is the one with the largest minimum distance.

The inner while-loop (lines 4–8) is executed as long as at least one reference solution is new in the *RefSet*. A solution is considered new if it has not been subjected to the subset generation (step 5) and combination (step 6) methods. If the reference set contains at least one new solution, the subset generation method builds a list of all the reference solution subsets that will become the input to the combination method. The subset generation method creates new subsets only. A subset is new if it contains at least one new reference solution. This avoids the application of the combination method to the same subset more than once, which is particularly wasteful when the combination method is completely deterministic. Combination methods that contain random elements may be able to produce new trial solutions even when applied more than once to the same subset of reference solutions. However, this is generally discouraged in favor of introducing new solutions in the reference set by replacing some of the old ones in the rebuilding step (line 10). The most common subset generation consists of creating a list of all pairs (i.e., all 2-subsets) of reference solutions for which at least one of the solutions is new. If the reference set contains n new solutions and m old ones, the number of 2-subsets that the subset generation method produces is given by

$$nm + \frac{n^2 - n}{2}.$$

The combination method (step 6) is applied to the subsets of reference solutions generated in the previous step. Most combination methods are designed to produce more than one trial solution from the combination of the solutions in a subset. The implementation of this method depends on the solution representation. Problem context can also be exploited by this method; however, it is also possible to create context-independent combination mechanisms. When implementing a context-independent procedure, it is beneficial to employ more than one combination method and track their individual performance. Scatter search provides great flexibility in terms of generating new trial solutions. The methodology accepts fully deterministic combination methods or those containing random elements that are typically used in genetic algorithms (and labeled crossover or mutation operators). Path relinking has been used as a combination method within SS. Path relinking (PR) can be considered an extended form of the combination method. 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.

The trial solutions generated by the combination method are given to the improvement method (step 7), and the output forms a pool of improved trial solutions that will be considered for admission in the reference set (step 8). Most improvement methods within SS implementations consist of local search procedures of the form outlined in Algorithm 1. However, the SS framework admits a wide variety of improvement methods, including those based on variable neighborhood descent (Hansen et al. [15]) or tabu search.

The reference set update in line 8 typically consists of the selection of the best (according to the objective function value) solutions from the union of the reference set and the set of trial solutions generated by steps 5–7 in Algorithm 4. If no new solutions are added to the reference set after the execution of the reference set update method, then the process exits the inner while-loop. The rebuilding step in line 10 is optional. That is, it is possible to implement a scatter search procedure that terminates the first time that the reference set does not change. However, most implementations extend the search beyond this point by executing a *RefSet* rebuilding step. The rebuilding of the reference set entails the elimination of some current reference solutions and the addition of diverse solutions. In most implementations, all solutions except the best are replaced in this step. The diverse solutions to be added may be either population solutions that have not been used or new solutions constructed with the generation diversification method. Note that only 10 solutions out of 100 are used from the population to build the initial reference set, and therefore the remaining 90 could be used for rebuilding purposes. When generating new population solutions, frequency-based memory à la tabu search is used to modify the original semigreedy function.

The process (i.e., the main while-loop in lines 2–10) continues as long as the stopping criteria are not satisfied. Possible stopping criteria include the number of rebuilding steps or elapsed time. When scatter search is applied in the context of optimizing expensive black boxes, a limit on the number of calls to the objective function evaluator (i.e., the black box) may also be used as a criterion for stopping.

From this description of the SS methodology, we can identify the three major areas in which scatter search and tabu search overlap (see Table 13).

A STM tabu search can be employed as the improvement method within scatter search. A limited number of iterations can be set as the stopping criterion for the tabu search to create the right balance between the computational time spent generating solutions through the combination method and the time spent improving them. PR is an effective way of combining solutions because the process naturally generates several trial solutions in the relinking path. A frequency-based LTM structure is an effective form of diversification when repopulating P to extend a scatter search beyond the first execution of the main loop.

3.1. Scatter Search Example: Knapsack Problem

We use the knapsack problem instance introduced in Section 2.1.1 to illustrate the design of a scatter search procedure. We start with a diversification method for binary problems original suggested by Glover [9]. This generator operates as follows. Choose a value for a parameter $h_{\max} \leq n-1$, where n is the number of variables in the problem. Let us choose $h_{\max} = 7$. For each value of h, $h = 1, \ldots, h_{\max}$, two solutions are generated. Type 1 solutions are denoted by x'

Table 13. Relationship between scatter search and tabu search.

SS method	TS element
Improvement	Short-term memory
Combination	Path relinking
Diversification	Long-term memory

and are initialized by a chosen seed x. In this case, we choose $x = (0, 0, 0, 0, 1, 0, 0, 1)$ as the seed. Then some of the individual elements are modified as follows:

$$x'_1 = 1 - x_1,$$
$$x'_{1+hk} = 1 - x_{1+hk} \quad \text{for } k = 1, \ldots, \lfloor (n-1)/h \rfloor.$$

This means that when $h = 1$, all variable values from the seed solution are "flipped"; that is, they are changed from 0 to 1 or 1 to 0. When $h = 2$, then the first, third, fifth, and seventh variable values are flipped. This continues until h reaches $h_{\max}$—that is, until $h = 7$. Type 2 solutions are denoted by x'' and are obtained as the complement of type 1 solutions (i.e., $x'' = 1 - x'$). Table 14 shows the 14 solutions generated with this method using the parameters given above.

The improvement method is then applied to the solutions in Table 14. Because the solutions generated with the diversification generation method are not guaranteed to be feasible, the improvement method must be capable of handling trial solutions that are either feasible or infeasible. To illustrate, we consider a simple procedure that operates as follows:

- If a trial solution is *infeasible*, then the method attempts to improve it by first finding a feasible solution. A feasible solution is found by changing variable values from 1 to 0 until the constraint is no longer violated. The variables are considered in increasing order of their bang-for-buck ratio, starting with the one with the smallest ratio. Once the solution becomes feasible, the following procedure is applied.
- If a trial solution is *feasible*, then the method attempts to improve it by changing variable values from 0 to 1. This is done in a greedy fashion, so that the first variable to be considered is the one with the largest bang-for-buck ratio. The procedure stops when no more variables can be given a value of one without violating the constraint.

Because the variables in our example are indexed by bang-for-buck ratio, the first part of the improvement procedure translates to setting variable values from 1 to 0 in decreasing index value, whereas the second part consists of setting variable values from 0 to 1 in increasing index value.

Table 15 shows the solutions obtained by the application of the improvement method to the trial solutions in Table 14.

Note that the trial solutions 4 and 10 in Table 15 became the same improved solution with an objective value of 41. In these cases, which are not atypical, the diversification generation method can be applied until a desired number of different improved solutions are found. Trial solutions 1, 3, 11, 12, and 14 are infeasible. Their corresponding improved solutions are feasible but the profit drops, except for solution 12, where the improved solution is feasible with a better profit value. The "Distance" column in Table 15 shows the average Hamming distance of each solution to all other solutions in the set. By the nature of the diversification method, the distance value for all of the original trial solutions is 4.3. The effect of the improvement method is that the total diversification in the set is reduced. Note that most distance values for the

Table 14. Set of diverse solutions.

h	x'	x''
1	(1, 1, 1, 1, 0, 1, 1, 0)	(0, 0, 0, 0, 1, 0, 0, 1)
2	(1, 0, 1, 0, 0, 0, 1, 1)	(0, 1, 0, 1, 1, 1, 0, 0)
3	(1, 0, 0, 1, 1, 0, 1, 1)	(0, 1, 1, 0, 0, 1, 0, 0)
4	(1, 0, 0, 0, 0, 0, 0, 1)	(0, 1, 1, 1, 1, 1, 1, 0)
5	(1, 0, 0, 0, 1, 1, 0, 1)	(0, 1, 1, 1, 0, 0, 1, 0)
6	(1, 0, 0, 0, 1, 0, 1, 1)	(0, 1, 1, 1, 0, 1, 0, 0)
7	(1, 0, 0, 0, 1, 0, 0, 0)	(0, 1, 1, 1, 0, 1, 1, 1)

Table 15. Improved set of diverse solutions.

Solution	Trial solution	Profit	Improved solution	Profit	Distance
1	(1, 1, 1, 1, 0, 1, 1, 0)	55[a]	(1, 1, 1, 1, 0, 0, 0, 0)	41	3.4
2	(1, 0, 1, 0, 0, 0, 1, 1)	31	(1, 0, 1, 0, 0, 1, 1, 1)	36	4.5
3	(1, 0, 0, 1, 1, 0, 1, 1)	37[a]	(1, 0, 0, 1, 1, 0, 1, 0)	34	4.9
4	(1, 0, 0, 0, 0, 0, 0, 1)	13	(1, 1, 1, 0, 0, 1, 0, 1)	41	3.2
5	(1, 0, 0, 0, 1, 1, 0, 1)	25	(1, 1, 0, 0, 1, 1, 0, 1)	39	3.5
6	(1, 0, 0, 0, 1, 0, 1, 1)	29	(1, 0, 0, 0, 1, 1, 1, 1)	34	4.8
7	(1, 0, 0, 0, 1, 0, 0, 0)	17	(1, 1, 1, 0, 1, 0, 0, 0)	40	3.4
8	(0, 0, 0, 0, 1, 0, 0, 1)	10	(1, 1, 0, 0, 1, 1, 0, 1)	39	3.5
9	(0, 1, 0, 1, 1, 1, 0, 0)	34	(0, 1, 0, 1, 1, 1, 0, 0)	34	3.8
10	(0, 1, 1, 0, 0, 1, 0, 0)	28	(1, 1, 1, 0, 0, 1, 0, 1)	41	3.2
11	(0, 1, 1, 1, 1, 1, 1, 0)	52[a]	(0, 1, 1, 1, 1, 0, 0, 0)	38	3.8
12	(0, 1, 1, 1, 0, 0, 1, 0)	40[a]	(1, 1, 1, 1, 0, 0, 0, 0)	41	3.4
13	(0, 1, 1, 1, 0, 1, 0, 0)	36	(0, 1, 1, 1, 0, 1, 0, 0)	36	3.5
14	(0, 1, 1, 1, 0, 1, 1, 1)	48[a]	(0, 1, 1, 1, 0, 1, 0, 0)	36	3.5

[a]Infeasible solution.

improved solutions are under 4.3. Only improved solutions 2, 3, and 6 experienced an increase in their distance values. We now apply the reference set update method.

This method is used to create and maintain a set of reference solutions. Table 16 shows the reference set of size 4 that results from selecting the population solutions of the highest quality.

An alternative design is to select some solutions by quality and some solutions by diversity. Table 17 shows the reference set that is the result of selecting the two highest-quality solutions and then the solutions with the largest minimum distance to the solutions already in the reference set. The last column of Table 17 contains the minimum distance between the solution in the row and the other three solutions in the reference set.

After selecting the best two improved solutions in Table 15, the third solution in Table 17 is selected to maximize the minimum distance to the two reference solutions already in the set. Likewise, once the third solution is included in the reference set, the fourth solution is selected to maximize the minimum distance between itself and the three solutions already in the reference set.

Next, we apply the subset generation method. We focus on subsets of size 2. Therefore, a reference set of size 4 generates six pairs in the first iteration because all solutions are "new" in the reference set. Each pair is then combined to generate new trial solutions.

Combination methods are typically problem-specific, and they are directly related to the solution representation. Depending on the specific form of the solution combination method, each subset can create one or more new solutions. For illustration purposes, we use a combination method that creates only one solution from each subset. Specifically, our solution combination method calculates a score for each variable, based on the solutions in the subset and their corresponding objective values. Let x^i be the ith solution in the subset. The score for variable j that corresponds to the solutions in the subset is calculated with the following formula:

$$score(j) = \frac{\sum_i f(x^i) x^i_j}{\sum_i f(x^i)},$$

Table 16. Reference set with high-quality solutions.

No.	Reference solution	Profit	Weight
1	(1, 1, 1, 1, 0, 0, 0, 0)	41	36
2	(1, 1, 1, 0, 0, 1, 0, 1)	41	38
3	(1, 1, 1, 0, 1, 0, 0, 1)	40	35
4	(1, 1, 0, 0, 1, 1, 0, 1)	39	38

Table 17. Reference set with high-quality and diverse solutions.

No.	Reference solution	Profit	Weight	Minimum distance
1	(1, 1, 1, 1, 0, 0, 0, 0)	41	36	
2	(1, 1, 1, 0, 0, 1, 0, 1)	41	38	
3	(1, 0, 0, 1, 1, 0, 1, 0)	34	35	4
4	(0, 1, 0, 1, 1, 1, 0, 0)	34	35	4

where $f(x)$ is the objective function value (profit) for solution x. The deterministic version of this scoring function can be used for subsets with more than two solutions, given that each solution has a relative weight that corresponds to its objective function value. The value of x_j in the new trial solution is 1 if $score(j) > 0.5$ and 0 otherwise. In a probabilistic version, the score may be interpreted as the probability that variable takes on the value of 1. The probabilistic version is used for subsets of size 2, because the deterministic version most of the time results in a copy of the reference solution with the better objective function value of the two being combined.

This combination mechanism may construct infeasible solutions. This does not represent a problem because the improvement method is applied to each trial solution created after the execution of the combination method. Recall that the improvement method is designed to deal with either feasible or infeasible solutions.

The new solutions generated and improved in these steps of the procedure are considered for membership in the reference set. A solution may become a member if its objective value is better than the objective value of any of the solutions in the reference set, when the reference set is being updated by solution quality only. Otherwise, a new solution that improves the diversity of the reference set could replace one that is currently in the diverse set. If the reference set is modified, then the subset generation method, the combination method, and the improvement method are applied in sequence. The application of the methods continues until the reference set converges (i.e., no elements in the set are replaced). At this point, the diversification method can be applied again from a different seed, and the search continues until the termination criteria are satisfied.

The *educ.2018.0181.sm1.xlsm* online supplement illustrates the main elements of scatter search. It contains the improvement method after the diversification generation (Table 15), the reference set update method (Table 16), and the combination method described on pages 153 and 154. It also shows how the improvement method is applied after the combination method (tab labeled "Improvement 2"). This workbook uses macros to perform the operations associated with each method. The Clear button should be clicked first, followed by the buttons that trigger the corresponding scatter search method. Table 15 includes a Diversity button that calculates the distance metric to show how diversity decreases after the improvement method is performed.

3.2. Implementing Scatter Search

Laguna and Martí [19] describe the scatter search methodology and effective ways to implement it. We summarize some of the most relevant recommendations:

1. Effective diversification generation methods employ controlled randomization and frequency-based memory to generate a set of diverse solutions. The use of frequency-based memory is typical in implementations of tabu search.

2. The diversification generation method is used at the beginning of the search to generate a set of diverse solutions. In most scatter search applications, the size of this population of solutions is set to $\max(100, 5b)$, where b is the size of the reference set.

3. Whereas some solution generation is done without considering the objective function—in other words, some diversification generation methods focus on diversification and not on the

quality of the resulting solutions—it is generally more effective to design a procedure that balances diversification and solution quality (such as those based on GRASP constructions).

4. Improvement methods must be capable of handling starting solutions that are either feasible or infeasible. When encountering an infeasible solution, an improvement method should search for a feasible solution first and then launch a search for improvement.

5. Whenever possible, the improvement method should be a known local search procedure. For example, when using scatter search for nonlinear optimization, the improvement method could be the well-known Nelder–Mead simplex procedure (Nelder and Mead [26]).

6. The reference set update method to try first is the so-called *static update.* Trial solutions that are constructed as a combination of reference solutions are placed in a solution *pool.* After the application of both the combination method and the improvement method, the pool is full, and the reference set is updated. The new reference set consists of the best b solutions (where b is the size of the reference set) from the solutions in the current reference set and the solutions in the pool; that is, the updated reference set contains the best b solutions in the union of the reference set and the pool.

7. The subset generation method should be limited to generating all solution pairs first before including subsets of higher dimensions. Even when the combination method includes stochastic elements, it is generally more effective to create only solution subsets that include at least one new reference solution (i.e., a solution that has been added to the reference set in the previous iteration).

8. The number of solutions created from the combination of two or more reference solutions should depend on the quality of the solutions being combined. For instance, the maximum number of solutions generated by the combination method should happen when the two best solutions in the reference set are being combined. Likewise, only one solution should be the result of combining the two reference solutions with the worst objective function value.

9. If scatter search is implemented to exploit problem context, the combination method should take full advantage of the information associated with the problem being solved. For instance, a combination method for the linear ordering problem should take into consideration that the largest contribution to the objective function value comes from the items that are placed in the first positions of the permutation.

10. Employing multiple combination methods has been shown to be an effective strategy. The combination methods are applied probabilistically, starting with an equal probability of selecting any of the available methods. The probability changes, with the success of each method, where success is typically defined as creating trial solutions that are admitted to the reference set because of their quality. As the search progresses, the more effective (i.e., successful) methods are chosen more often. This strategy is particularly effective when scatter search is used as a black-box optimizer, where no context information is used to create combination methods that are known to work well on certain classes of problems. Scatter search has been implemented as a black-box-optimizer for optimization problems with continuous variables (Laguna and Martí [20], Laguna et al. [22]), integer variables (Laguna et al. [23]), binary variables (Gortazar et al. [14]), and permutations (Campos et al. [6]).

4. Conclusions

This tutorial started with description of the essential elements of tabu search—namely, short-term memory, long-term memory, strategic oscillation, and path relinking. Attention was given to these core components of the tabu search framework for the following reasons:

- Such a focus may help to uncover a better form for the strategies associated with these core components.
- Weaknesses and strengths of these core components, when studied in isolation from other ideas, may stand out more clearly, thus yielding insights into the features that a complete approach may require to produce better methods.

• For methods that are susceptible to modular implementations, as typically occurs for tabu search, simpler designs can readily be made a part of more complex designs.

Whereas TS implementations using a combination of these elements are likely to produce decent outcomes, readers are encouraged to pursue more complete approaches by examining strategies outlined in Glover and Laguna [12].

A great deal remains to be learned about tabu search. Evidently, very little is known about how human beings use memory in problem solving. It is conceivable that discoveries about effective uses of memory within search methods will provide clues about strategies that humans skillfully employ. The potential links between the areas of heuristic search and psychology have barely been examined. The numerous successes of tabu search implementations provide encouragement that such relationships might be profitable to investigate more fully.

The goal with respect to scatter search was to introduce the framework at a level that would make it possible for the reader to implement a basic but robust procedure. A number of extensions are possible, and some of them have already been explored and reported in the literature. It is not possible within the limited scope of this tutorial to detail completely many of the aspects of scatter search that warrant further investigation. Additional implementation considerations, including those associated with intensification and diversification processes, and the design of accompanying methods to improve solutions produced by combination strategies are found in several publications, particularly in Laguna and Martí [19].

Endnote

[1] A term coined by Fred Glover in 1986 (see reference [10]).

References

[1] M. Albareda-Sambola, E. Fernández, and F. Saldanha-da-Gama. Heuristic solutions to the facility location problem with general Bernoulli demands. *INFORMS Journal on Computing* 29(4): 737–753, 2017.

[2] C. Archetti, N. Boland, and M. G. Speranza. A matheuristic for the multivehicle inventory routing problem. *INFORMS Journal on Computing* 29(3):377–387, 2017.

[3] R. Battiti and G. Tecchiolli. The reactive tabu search. *INFORMS Journal on Computing* 6(2): 126–140, 1994.

[4] M. Ben Salem, S. Hanafi, R. Taktak, and H. Ben Abdallah. Probabilistic tabu search with multiple neighborhoods for the disjunctively constrained knapsack problem. *RAIRO Operations Research* 51(3):627–637, 2017.

[5] N. T. Boardman, B. J. Lunday, and M. J. Robbins. Heterogeneous surface-to-air missile defense battery location: A game theoretic approach. *Journal of Heuristics* 23(6):417–447, 2017.

[6] V. Campos, M. Laguna, and R. Martí. Context-independent scatter and tabu search for permutation problems. *INFORMS Journal on Computing* 17(1):111–122, 2005.

[7] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6(2):109–133, 1995.

[8] M. Gharote, R. Patil, and S. Lodha. Scatter search for trainees to software project requirements stable allocation. *Journal of Heuristics* 23(4):257–283, 2017.

[9] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences* 8(1): 156–166, 1977.

[10] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research* 13(5):533–549, 1986.

[11] F. Glover. A template for scatter search and path relinking. J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers, eds. *Artificial Evolution*, *Lecture Notes in Computer Science*, Vol. 1363. Springer, Berlin, 1–51, 1998.

[12] F. Glover and M. Laguna. *Tabu Search*. Springer, New York, 1997.

[13] M. A. González, J. J. Palacios, C. R. Vela, and A. Hernández-Arauzo. Scatter search for minimizing weighted tardiness in a single machine scheduling with setups. *Journal of Heuristics* 23(2–3): 81–110, 2017.

[14] F. Gortazar, A. Duarte, M. Laguna, and R. Martí. Context-independent scatter search for binary problems. *Computers & Operations Research* 37(11):1977–1986, 2010.

[15] P. Hansen, N. Mladenović, J. Brimberg, and J. A. Moreno Pérez. Variable neighborhood search. M. Gendreau and J.-Y. Potvin, eds. *Handbook of Metaheuristics*. Springer, New York, 61–86, 2010.

[16] B. Hardy, R. Lewis, and J. Thompson. Tackling the edge dynamic graph colouring problem with and without future adjacency information. *Journal of Heuristics* 24(3):321–343, 2018.

[17] L.-P. Kerkhove and M. Vanhoucke. A parallel multi-objective scatter search for optimising incentive contract design in projects. *European Journal of Operational Research* 261(3):1066–1084, 2017.

[18] M. Laguna. Scatter search. E. K. Burke and G. Kendall, eds. *Search Methodologies: Introductory Tutorials and Decision Support Techniques*. Springer, New York, 119–141, 2014.

[19] M. Laguna and R. Martí. *Scatter Search Methodology and Implementations in C*. Springer, New York, 2003.

[20] M. Laguna and R. Martí. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization* 33(2):235–255, 2005.

[21] M. Laguna and R. Martí. Heuristics. S. Gass and M. Fu, eds. *Encyclopedia of Operations Research and Management Science*, 3rd ed., Springer, Boston, 2013, 695–703.

[22] M. Laguna, J. Molina, F. Pérez, R. Caballero, and A. Hernández-Díaz. The challenge of optimizing expensive black boxes: A scatter search/rough set theory approach. *Journal of the Operational Research Society* 61(1):53–67, 2010.

[23] M. Laguna, R. Martí, F. Gortázar, M. Gallego, and A. Duarte. A black-box scatter search for optimization problems with integer variables. *Journal of Global Optimization* 58(3):497–516, 2014.

[24] F. Li and Z-L, Tang, L. Chen. Integrated production, inventory and delivery problems: Complexity and algorithms. *INFORMS Journal on Computing* 29(2):232–250, 2017.

[25] R. Martí, M. Laguna, and V. Campos. Scatter search vs. genetic algorithms: An experimental evaluation with permutation problems. C. Rego and B. Alidaee, eds. *Metaheuristic Optimization via Adaptive Memory and Evolution: Tabu Search and Scatter Search*. Springer, New York, 263–282, 2005.

[26] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal* 7(4):308–313, 1965.

[27] M. M. A. Polash, M. A. H. Newton, and A. Sattar. Constraint-based search for optimal Golomb rulers. *Journal of Heuristics* 23(6):501–523, 2017.

[28] M. A. Remli, S Deris, M. S. Mohamad, S. Omatu, and J. M. Corchado. An enhanced scatter search with combined opposition-based learning for parameter estimation in large-scale kinetic models of biochemical systems. *Engineering Applications of Artificial Intelligence* 62(June):164–180, 2017.

[29] M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures. F. Glover and G. Kochenberger, eds. *Handbook of Metaheuristics*, International Series in Operations Research and Management Science, Vol. 57. Springer, Boston, 219–250, 2003.

[30] V. Riahi, M. Khorramizadeh, M. A. H. Newton, and A. Sattar. Scatter search for mixed blocking flowshop scheduling. *Expert Systems with Applications* 79(August):20–32, 2017.

[31] R. C. L. Riley and C. Rego. Intensification, diversification, and learning via relaxation adaptive memory programming: A case study on resource constrained project scheduling. *Journal of Heuristics*, ePub ahead of print March 15, https://doi.org/10.1007/s10732-018-9368-y, 2018.

[32] B. C. Shelbourne, M. Battarra, and C. N. Potts. The vehicle routing problem with release and due dates. *INFORMS Journal on Computing* 29(4):705–723, 2017.

CORRECTION

In this article, "Tabu and Scatter Search: Principles and Practice" by Manuel Laguna (*Tutorials in Operations Research*, 2018, ch. 7, pp. 130–157), supplemental material has been provided by the author to allow the reader to follow certain algorithms and explore what-if scenarios. The online version has been corrected to add citation to and description of these online supplemental materials.