2.5 TABU SEARCH

Tabu search algorithm was proposed by Glover [323]. In 1986, he pointed out the controlled randomization in SA to escape from local optima and proposed a deterministic algorithm [322]. In a parallel work, a similar approach named "steepest ascent/mildest descent" has been proposed by Hansen [364]. In the 1990s, the tabu search algorithm became very popular in solving optimization problems in an approximate manner. Nowadays, it is one of the most widespread S-metaheuristics. The use of memory, which stores information related to the search process, represents the particular feature of tabu search.

TS behaves like a steepest LS algorithm, but it accepts nonimproving solutions to escape from local optima when all neighbors are nonimproving solutions. Usually, the whole neighborhood is explored in a deterministic manner, whereas in SA a random neighbor is selected. As in local search, when a better neighbor is found, it replaces the current solution. When a local optima is reached, the search carries on by selecting a candidate worse than the current solution. The best solution in the neighborhood is selected as the new current solution even if it is not improving the current solution. Tabu search may be viewed as a dynamic transformation of the neighborhood. This policy may generate cycles; that is, previous visited solutions could be selected again.

To avoid cycles, TS discards the neighbors that have been previously visited. It memorizes the recent search trajectory. Tabu search manages a memory of the solutions or moves recently applied, which is called the *tabu list*. This tabu list constitutes the short-term memory. At each iteration of TS, the short-term memory is updated. Storing all visited solutions is time and space consuming. Indeed, we have to check at each iteration if a generated solution does not belong to the list of all visited solutions. The tabu list usually contains a constant number of tabu moves. Usually, the attributes of the moves are stored in the tabu list.

By introducing the concept of solution features or move features in the tabu list, one may lose some information about the search memory. We can reject solutions that have not yet been generated. If a move is "good," but it is tabu, do we still reject it? The tabu list may be too restrictive; a nongenerated solution may be forbidden. Yet for some conditions, called *aspiration criteria*, tabu solutions may be accepted. The admissible neighbor solutions are those that are nontabu or hold the aspiration criteria.

In addition to the common design issues for S-metaheuristics such as the definition of the neighborhood and the generation of the initial solution, the main design issues that are specific to a simple TS are

- **Tabu list:** The goal of using the short-term memory is to prevent the search from revisiting previously visited solutions. As mentioned, storing the list of all visited solutions is not practical for efficiency issues.
- **Aspiration criterion:** A commonly used aspiration criteria consists in selecting a tabu move if it generates a solution that is better than the best found solution. Another aspiration criterion may be a tabu move that yields a better solution among the set of solutions possessing a given attribute.

Some advanced mechanisms are commonly introduced in tabu search to deal with the intensification and the diversification of the search:

- **Intensification (medium-term memory):** The medium-term memory stores the elite (e.g., best) solutions found during the search. The idea is to give priority to attributes of the set of elite solutions, usually in weighted probability manner. The search is biased by these attributes.
- **Diversification (long-term memory):** The long-term memory stores information on the visited solutions along the search. It explores the unvisited areas of the solution space. For instance, it will discourage the attributes of elite solutions in the generated solutions to diversify the search to other areas of the search space.

Algorithm 2.9 describes the template of the TS algorithm.

Algorithm 2.9 Template of tabu search algorithm.

$s = s_0$; /* Initial solution */
Initialize the tabu list, medium-term and long-term memories ;
Repeat
  Find best admissible neighbor s' ; /* non tabu or aspiration criterion holds */
  $s = s'$;
  Update tabu list, aspiration conditions, medium and long term memories ;
  If intensification_criterion holds **Then** intensification ;
  If diversification_criterion holds **Then** diversification ;
Until Stopping criteria satisfied
Output: Best solution found.

Theoretical studies carried out on tabu search algorithms are weaker than those established for simulated annealing. A simulated annealing execution lies within the convex hull of a set of tabu search executions [28]. Therefore, tabu search may inherit some nice theoretical properties of SA.

TABLE 2.8 The Different Search Memories of Tabu Search

Search Memory	Role	Popular Representation
Tabu list	Prevent cycling	Visited solutions, moves attributes Solution attributes
Medium-term memory	Intensification	Recency memory
Long-term memory	Diversification	Frequency memory

In addition to the search components of local search (hill climbing), such as the representation, neighborhood, initial solution, we have to define the following concepts that compose the search memory of TS: the tabu list (short-term memory), the medium-term memory, and the long-term memory (Table 2.8).

2.5.1 Short-Term Memory

The role of the short-term memory is to store the recent history of the search to prevent cycling. The naive straightforward representation consists in recording all visited solutions during the search. This representation ensures the lack of cycles but is seldom used as it produces a high complexity of data storage and computational time. For instance, checking the presence of all neighbor solutions in the tabu list will be prohibitive. The first improvement to reduce the complexity of the algorithm is to limit the size of the tabu list. If the tabu list contains the last k visited solutions, tabu search prevents a cycle of size at most k. Using hash codes may also reduce the complexity of the algorithms manipulating the list of visited solutions.

In general, attributes of the solutions or moves are used. This representation induces a less important data storage and computational time but skips some information on the history of the search. For instance, the absence of cycles is not ensured.

The most popular way to represent the tabu list is to record the move attributes. The tabu list will be composed of the reverse moves that are forbidden. This scheme is directly related to the neighborhood structure being used to solve the problem. If the move m is applied to the solution s_i to generate the solution s_j ($s_j = s_i \oplus m$), then the move m (or its reverse m^{-1} such that $(s_i \oplus m) \oplus m^{-1} = s_i$) is stored in the list. This move is forbidden for a given number of iterations, named the *tabu tenure* of the move. If the tabu list contains the last k moves, tabu search will not guarantee to prevent a cycle of size at most k.

Example 2.26 Tabu list based on move attributes. Let us consider a permutation optimization problem where the neighborhood is based on exchanging two elements of the permutation. Given a permutation π, a move is represented by two indices (i, j). This move generates the neighbor solution π' such that

$$\pi'(k) = \begin{cases} \pi(k) & \text{for } k \neq i \text{ and } k \neq j \\ \pi(j) & \text{for } k = i \\ \pi(i) & \text{for } k = j \end{cases}$$

The inverse move (j, i) may be stored in the tabu list and is forbidden. A stronger tabu representation may be related to the indices i and j. This will disallow any permutation involving the indices i and j.

In tabu search, many different tabu lists may be used in conjunction. For instance, some ingredients of the visited solutions and/or the moves are stored in multiple tabu lists. A move m is not tabu if the conditions for *all* tabu lists are satisfied. In many optimization problems, it is more and more popular to use simultaneously various move operators, and hence different neighborhoods are defined.

Example 2.27 Multiple tabu lists. Let us consider the maximum independent set problem. Given a graph $G = (V, E)$, the problem consists in finding a subset of vertices X of V, maximizing the cardinality of the set X, and satisfying the constraint that there is no edge $(i, j) \in E$ having its adjacent vertices in X. Tabu search may be applied to determine an independent set of size k [285]. The objective function consists in minimizing the number of edges having their adjacent vertices in X. The neighborhood is defined by the exchange of a vertex $v \in X$ with a vertex $w \in E - X$. The size of the neighborhood is $k(n - k) = O(nk)$, where n represents the number of vertices. Three different tabu lists may be used:

- The first tabu list T_1 stores the last generated solutions. A hashing technique may be used to speed up the test of tabu conditions [825].
- the second tabu list T_2 stores the last vertices introduced into X.
- the third tabu list T_3 stores the last vertices removed from X.

The size of the tabu list is a critical parameter that has a great impact on the performances of the tabu search algorithm. At each iteration, the last move is added in the tabu list, whereas the oldest move is removed from the list. The smaller is the value of the tabu list, the more significant is the probability of cycling. Larger values of the tabu list will provide many restrictions and encourage the diversification of the search as many moves are forbidden. A compromise must be found that depends on the landscape structure of the problem and its associated instances.

The tabu list size may take different forms:

- **Static:** In general, a static value is associated with the tabu list. It may depend on the size of the problem instance and particularly the size of the neighborhood. There is no optimal value of the tabu list for all problems or even all instances of a given problem. Moreover, the optimal value may vary during the search progress. To overcome this problem, a variable size of the tabu list may be used.
- **Dynamic:** The size of the tabu list may change during the search without using any information on the search memory.

Example 2.28 Robust tabu search. In this tabu search algorithm, a uniform random change of the tabu list size in a given interval $[T_{\min}, T_{\max}]$ is used [738].

- **Adaptive:** In the adaptive scheme, the size of the tabu list is updated according to the search memory. For instance, the size is updated upon the performance of the search in the last iterations [573].

Example 2.29 Reactive tabu search. In the reactive tabu search, the main idea is to increase the size of the tabu list when the short-term memory indicates that the search is revisiting solutions, that is, the presence of a cycle in the search [59]. Hence, the short-term memory stores the visited solutions and not the attributes of the solutions or moves. Hashing techniques are used to reduce the complexity of manipulating such short-term memories.

2.5.2 Medium-Term Memory

The role of the intensification is to exploit the information of the best found solutions (elite solutions) to guide the search in promising regions of the search space. This information is stored in a medium-term memory. The idea consists in extracting the (common) features of the elite solutions and then intensifying the search around solutions sharing those features. A popular approach consists in restarting the search with the best solution obtained and then fixing in this solution the most promising components extracted from the elite solutions.

The main representation used for the medium-term memory is the *recency memory*. First, the components associated with a solution have to be defined; this is a problem-specific task. The recency memory will memorize for each component the number of successive iterations the component is present in the visited solutions. The most commonly used event to start the intensification process is a given period or after a certain number of iterations without improvement.

Example 2.30 Recency memory in solving the GAP. Suppose one has to solve the generalized assignment problem where a solution is encoded by a discrete vector and the neighborhood is defined by the substitute operator. The recency memory may be represented by a symmetric matrix R where an element r_{ij} represents the number of successive iterations the object i has been assigned to the site j. The intensification will consist in starting the search from the best found solution s^*. The largest values r_{ij} are extracted from the recency matrix R. Then the associated elements i in the solution s^* are fixed to the sites j. The search will focus only on the other decision variables of the solution s^*.

The intensification of the search in a given region of the search space is not always useful. The effectiveness of the intensification depends on the landscape structure of the target optimization problem. For instance, if the landscape is composed of many basins of attraction and a simple TS without intensification component is effective to search in each basin of attraction, intensifying the search in each basin is useless.

2.5.3 Long-Term Memory

As mentioned many times, S-metaheuristics are more powerful search methods in terms of intensification. Long-term memory has been introduced in tabu search to encourage the diversification of the search. The role of the long-term memory is to force the search in nonexplored regions of the search space.

The main representation used for the long-term memory is the *frequency memory*. As in the recency memory, the components associated with a solution have first to be defined. The frequency memory will memorize for each component the number of times the component is present in all visited solutions.

Example 2.31 Frequency memory in solving the GAP. The frequency memory may be represented by a symmetric matrix F where an element f_{ij} represents the number of times the object i has been assigned to the site j from the beginning of the tabu search. The diversification may consist in starting the search from a new solution s^*. The elements i of the initial solution s^* are assigned to the sites j that are associated with smaller values of f_{ij}.

The diversification process can be applied periodically or after a given number of iterations without improvement. Three popular diversification strategies may be applied [306]:

- **Restart diversification:** This strategy consists in introducing in the current or best solution the least visited components. Then a new search is restarted from this new solution.
- **Continuous diversification:** This strategy introduces during a search a bias to encourage diversification. For example, the objective function can integrate the frequency occurrence of solution components in the evaluation of current solutions. Frequently applied moves or visited solutions are penalized.
- **Strategic oscillation:** Introduced by Glover in 1989, strategic oscillation allows to consider (and penalize) intermediate solutions that are infeasible. This strategy will guide the search toward infeasible solutions and then come back to a feasible solution.

Example 2.32 Tabu search for the TSP problem. Designing a TS for the TSP problem needs the definition of the tabu list (short-term memory), the medium- and long-term memories, and the aspiration criterion:

- The short-term memory maintains a list of t edges and prevents them from being selected for consideration of moves for a number of iterations l. After the given number of iterations (tabu tenure), these edges are released. One can use a 2D Boolean matrix to decide if an edge is tabu or not.
- The medium- and long-term memory maintains a list of t edges that have been considered in the last k best (worst) solutions. The process encourages (or discourages) their selection in future solutions using their frequency of

appearance in the set of elite solutions and the quality of solutions they have appeared.

- The usual aspiration criterion that accepts the tabu moves that generate better solutions than the best found one.

As for the intensification, the diversification of the search is not always useful. It depends on the landscape structure of the target optimization problem. For instance, if the landscape is a "massif central" where all good solutions are localized in the same region of the search space within a small distance, diversifying the search to other regions of the search space is useless. The search time assigned to the diversification and the intensification components of TS must be carefully tuned depending on the characteristics of the landscape structure associated with the problem.

TS has been successfully applied to many optimization problems. Compared to local search and simulated annealing, various search components of TS are problem specific and must be defined. The search space for TS design is much larger than for local search and simulated annealing. The degree of freedom in designing the different ingredients of TS is important. The representation associated with the tabu list, the medium-term memory, and the long-term memory must be designed according to the characteristics of the optimization problem at hand. This is not a straightforward task for some optimization problems. Moreover, TS may be very sensitive to some parameters such as the size of the tabu list.