Chapter 9
Local Search Learning

Local searches play an essential role in metaheuristics. Virtually, all efficient heuristic methods incorporate a local search. Moreover, metaheuristics are sometimes defined as a master process guiding a local search. In Chapter 5, we have already seen some basic neighborhood adaptation techniques, in particular its limitation by the list of candidate moves, granular search and its extension by filter-and-fan search, and ejection chains.

Most randomized methods reviewed in Chapter 7 are dedicated to local search extensions. They are not implementing a learning process. They only memorize the best solution found so far or statistics for self-calibrating the parameters. This allows taking the unit of measurement of the fitness function into account. The following step in the sophistication of metaheuristics is to learn to locally modify solutions to a problem. Among the popular techniques, taboo search (also written tabu search) offers many strategies and various local search learning mechanisms. This chapter reviews the basic mechanisms. Other strategies are proposed in the book of Glover and Laguna [5] and take a more natural place in other chapters of the present book.

9.1 Taboo Search

Proposed by Fred Glover in 1986, the key idea of taboo search is to explore the solution space with a local search beyond local optima [2–4]. This implies designing a mechanism to prevent the cycling phenomenon, the fact of entering a cycle where a limited subset of solutions is repeatedly visited. The simplest concept to imagine is to memorize all the solutions which have been successively encountered during a local search but preventing the latter from choosing a neighbor solution that has already been visited. The visited solutions thus become taboo.

This concept is simple but cumbersome to implement: imagine that a local search can require millions of iterations, which means memorizing the same number of

É. D. Taillard, *Design of Heuristic Algorithms for Hard Optimization*, Graduate Texts in Operations Research, https://doi.org/10.1007/978-3-031-13714-3_9

solutions. Each neighbor solution must be checked to ensure that it has not already been explored. Knowing that a neighborhood can contain thousands of solutions, we quickly realize the hopelessness of this way of doing things, either because of the memory space needed to store the visited solutions or because of the computational effort to compare neighbor solutions.

9.1.1 Hash Table Memory

A simple technique to implement an approximation of this principle of prohibiting previously visited solutions is to use a hash table. An integer value $h(s)$ is associated with each solution s of the problem. If we visit the solution s_i at iteration i of the search, we store the value i in the entry $h(s_i) \mod m$ of an array T of m integers. Thus, the value t_k of the kth entry of the table T indicates at which iteration a solution whose hash value k (modulo m) has been visited.

The h function is generally not bijective over the set of solutions to the problem, so various solutions can have the same hash value. Indeed, the size m of the array T must be limited due to the available memory. This technique is, therefore, an approximation of the concept of prohibiting solutions already visited. Indeed, not only the latter is prohibited but also all those that have the same hash value. Moreover, since the value of m is limited, we cannot forever forbid returning to a solution of a given hash value. After m iterations at the latest, all the solutions would be prohibited.

It is therefore necessary to implement a key feature of the learning process: oblivion. These considerations lead us to introduce the key parameter of a taboo search: the taboo duration, sometimes referred to as the *taboo list length*.

9.1.1.1 Hash Functions

The choice of a hash function to implement a taboo search is not very difficult. In some cases, the value of the fitness function is perfect, especially when the neighborhood includes many moves at zero cost (plateaus). Indeed, taboo search chooses the best move allowed at each iteration. Hence, neutral changes make learning difficult. Being on a plateau, the choice of one or the other neighbor is problematic and cycling can occur. In case the fitness function admits an extensive range of values, prohibiting during a number of iterations to return to a given fitness value allows, in many cases, to break the local optimum structure and discover another one.

A general hash function is as follows. Let us use the notation introduced in Chapter 4 devoted to constructive methods. A solution is composed of elements $e \in E$. Each of them is associated with an integer value z_e. These values are randomly generated at the beginning of the algorithm. The hash value of a solution s is provided by $h(s) = \sum_{e \in s} z_e$. A more sophisticated hash technique using multiple tables is discussed in [6]. It makes it possible to obtain the equivalent of a very large table, while limiting the memory space.

9.1.2 *Taboo Moves*

Prohibition based on a hash function is uncommon in taboo search implementations. Frequently, one prohibits some moves or solutions with certain features. To be concrete, consider the example of the symmetric TSP.

A 2-opt move can be characterized by the pair $[i, j]$ which consists in replacing the edges $[i, s_i]$ and $[j, s_j]$ of the current solution s by the edges $[i, j]$ and $[s_i, s_j]$. One assumes here that the solution is provided by the “successor” of each city and that the city j comes “after” the city i when traveling in the order given by s. If the move $[i, j]$ is carried out at an iteration, one can prohibit the reverse move $[i, s_i]$ during the following iterations. This is a direct prohibition based on the opposite of a move.

After performing the move $[i, j]$, another possibility is to indirectly prohibit the moves leading to a solution containing both edges $[i, s_i]$ and $[j, s_j]$.

By abuse of language, let m^{-1} denote the inverse of a move, or a feature of a solution that is forbidden after performing the move m of a neighborhood characterized by a set M of moves. Although $(s \oplus m) \oplus m^{-1} = s$, there may be various ways to define m^{-1}. Since the size of the neighborhood is limited, it is necessary to relax the taboo status of a move after relatively few iterations. Therefore, the taboo list is frequently presented as a *short-term memory*. The most basic taboo search framework is given by Algorithm 9.1.

9.1.2.1 Implementation of Taboo Status

If the neighborhood size is not too large, it can be stored, for each move, the iteration from which it can be used again. Let us immediately illustrate such an implementation for the following knapsack instance with nine variables.

$$\begin{aligned} \max r &= 12s_1 + 10s_2 + 9s_3 + 7s_4 + 4s_5 + 8s_6 + 11s_7 + 6s_8 + 13s_9 \\ \text{Subject } & 10s_1 + 12s_2 + 8s_3 + 7s_4 + 5s_5 + 13s_6 + 9s_7 + 6s_8 + 14s_9 \leqslant 45 \\ \text{to: } & s_i \in \{0, 1\} \quad (i = 1, \dots, 9) \end{aligned} \tag{9.1}$$

A solution s of this problem is a 0—1 vector, with $s_i = 1$ if the object i is chosen and $s_i = 0$ otherwise. Each object occupies a certain volume in the knapsack and the latter possesses a global volume of 45. An elementary neighborhood for this problem is to alter the value of a unique variable of s.

The taboo conditions can be stored as a vector t of integers with t_i giving the iteration number at which the variable s_i can revert to a previous value. Initially,

Algorithm 9.1: Elementary taboo search framework

```
Input: Solution $s$, set $M$ of moves, fitness function $f(\cdot)$ to minimize, parameters $I_{max}, d$.
Result: Improved solution $s^*$
1  $s^* \leftarrow s$
2  for $I_{max}$ iterations do
3      $best\_neighbor\_value \leftarrow \infty$
4      forall $m \in M$ (such that $m$ (or $s \oplus m$) is not marked as taboo) do
5          if $f(s \oplus m) < best\_neighbor\_value$ then
6              $best\_neighbor\_value \leftarrow f(s \oplus m)$
7              $m^* \leftarrow m$
8      if $best\_neighbor\_value < \infty$ then
9          Mark $(m^*)^{-1}$ (or $s$) as taboo for the next $d$ iterations
10         $s \leftarrow s \oplus m^*$
11         if $f(s) < f(s^*)$ then
12             $s^* \leftarrow s$
13     else
14         Error message: $d$ too large: no move allowed!
```

$t = 0$: at the first iteration, all variables can be modified. For this small instance, let us assume a taboo duration of $d = 3$. The initial solution can be set to $s = 0$, which represents the worst feasible solution to the problem. Table 9.1 gives the evolution of a taboo search for this small instance.

Unsurprisingly, object 9 is put in the knapsack at the first iteration. Indeed, this object has the largest value. At the end of iteration 1, it is forbidden to set $s_9 = 0$ again up to the iteration $t_9 = 4 = 1 + 3$. As long as there is room in the knapsack, taboo search behaves like a greedy constructive algorithm. At iteration 4, it reaches the first local optimum $s = (1, 1, 0, 0, 0, 0, 1, 0, 1)$ of value $r = 46$.

Table 9.1 Evolution of an elementary taboo search for ten iterations for the knapsack instance 9.1. This search forbids changing again a given variable for $d = 3$ iterations

Iteration number	Variable modified	Modified solution	Fitness value	Volume used	Taboo status
1	s_9	(0, 0, 0, 0, 0, 0, 0, 0, 1)	13	14	(0, 0, 0, 0, 0, 0, 0, 0, 4)
2	s_7	(1, 0, 0, 0, 0, 0, 0, 0, 1)	25	24	(5, 0, 0, 0, 0, 0, 0, 0, 4)
3	s_7	(1, 0, 0, 0, 0, 0, 1, 0, 1)	36	33	(5, 0, 0, 0, 0, 0, 6, 0, 4)
4	s_2	(1, 1, 0, 0, 0, 0, 1, 0, 1)	46	45	(5, 7, 0, 0, 0, 0, 6, 0, 4)
5	s_9	(1, 1, 0, 0, 0, 0, 1, 0, 0)	33	31	(5, 7, 0, 0, 0, 0, 6, 0, 8)
6	s_3	(1, 1, 1, 0, 0, 0, 1, 0, 0)	42	39	(5, 7, 9, 0, 0, 0, 6, 0, 8)
7	s_8	(1, 1, 1, 0, 0, 0, 1, 1, 0)	48	45	(5, 7, 9, 0, 0, 0, 6, 10, 8)
8	s_2	(1, 0, 1, 0, 0, 0, 1, 1, 0)	38	33	(5, 11, 9, 0, 0, 0, 6, 10, 8)
9	s_4	(1, 0, 1, 1, 0, 0, 1, 1, 0)	45	40	(5, 11, 9, 12, 0, 0, 6, 10, 8)
10	s_5	(1, 0, 1, 1, 1, 0, 1, 1, 0)	49	45	(5, 11, 9, 12, 13, 0, 6, 10, 8)

At iteration 5, an object is removed, because the knapsack is dead full. Only object 9 can be removed due to taboo conditions. As a result, the fitness function decreases from $r = 46$ to $r = 33$, but space is freed up in the knapsack. At iteration 6, the best move would be to add object 9, but this move is taboo. It would correspond to return to the solution visited at iteration 4.

The best authorized move is therefore to add object 3. Then, at the subsequent iteration the object 8 is added, leading to a new local optimum $s = (1, 1, 1, 0, 0, 0, 1, 1, 0)$ of value $r = 48$. The knapsack is again completely full. At iteration 8, it is necessary to remove an object, setting $s_2 = 0$. The place thus released makes it possible to add the objects 4 and 5, discovering a solution $s = (1, 0, 1, 1, 1, 0, 1, 1, 0)$ even better than both local optima previously found.

For the TSP, the type of restrictions described above may be implemented using a matrix T whose entry t_{ij} provides the iteration from which we can again perform a move where the edge $[i, j]$ belongs to the tour. This principle extends to any combinatorial problem for which we search for an optimal permutation.

9.1.2.2 Taboo Duration

In the previous example, the taboo duration was set to three iterations. This value may seem arbitrary. If the taboo conditions are removed (duration set to zero), the search enters a cycle. Once a local optimum is reached, an object is removed and added again in the next iteration. The maximum taboo duration is clearly limited by the neighborhood size: indeed, the search performs all the moves of the neighborhood and then remains blocked. At that time, they are all prohibited.

These two extreme cases lead to inefficient searches—a zero taboo duration is equivalent to learning nothing; a very high duration implies poor learning. Consequently, we have to achieve a sensible compromise for the taboo duration. Therefore, this duration must be learned for the problem instance treated. Figure 9.1 illustrates this phenomenon for Euclidean TSP instances of size $n = 100$ randomly, uniformly distributed in a square. The taboo search performs $I_{max} = 1000$ iterations.

Battiti and Tecchiolli [1] proposed a learning mechanism called *reactive taboo search*. All the solutions visited by the search are memorized. They can be stored in an approximate way, employing the hash technique presented in Section 9.1.1.1. The search starts with a restricted taboo duration. If the search visits a solution again, then the duration is increased. If the search does not revisit any of the solutions during a relatively significant number of iterations, then the taboo duration is diminished.

This last condition seems strange. Why should we force the search to return to previously explored solutions? The explanation is as follows: if the taboo duration is sufficient to avoid the cycling phenomenon, it also means we are forbidden to visit some good solutions because of the taboo status. We are therefore likely to ignore high-quality solutions.

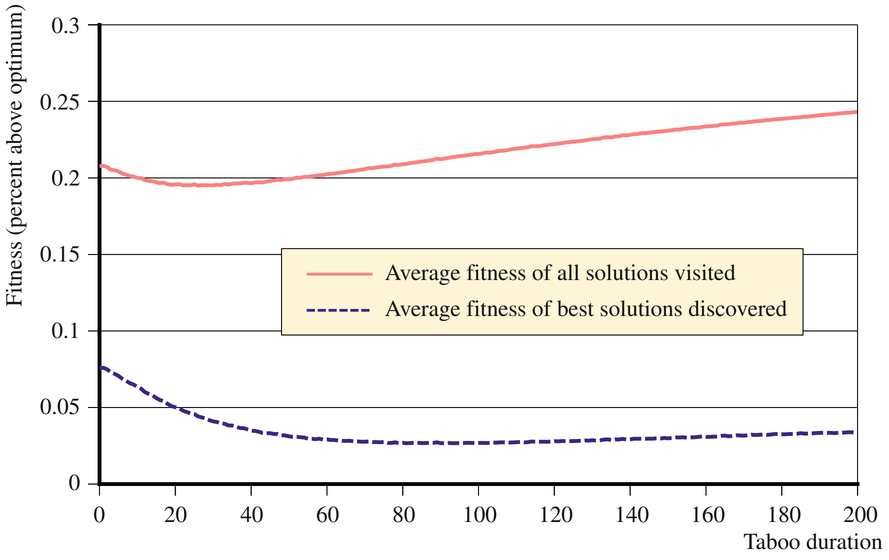

Fig. 9.1 Influence of the taboo duration for Euclidean TSP with 100 cities. A short duration allows visiting better quality solutions, on average. But the search cannot escape from local optima. Hence, the quality of the best solutions found is not excellent. Conversely, if the taboo duration is too high, the average solution quality decreases, as well as that of the best solutions discovered. In this case, a reasonable compromise seems to be a taboo duration around the instance size. More generally, the square root of the neighborhood size seems appropriate

It is therefore necessary to find a taboo duration long enough to avoid cycling but as short as possible so as not to prohibit good moves. This is precisely the purpose of reactive taboo search.

However, this learning technique only repels the problem. Indeed, the user must determine another parameter which is the number of iterations without revisiting a solution, triggering the taboo duration decrease. In addition, it requires the implementation of a storage mechanism for all visited solutions, which can be cumbersome to implement.

Another technique for choosing low taboo durations while strongly preventing the cycling phenomenon is to randomly set it at each iteration. A classic method is to select the taboo duration at random between a minimum duration d_{min} and a maximum value $d_{max} = d_{min} + \Delta$.

To create Fig. 9.2, 500 QAP instances of size $n = 12$ with known optimal solution have been generated. For each instance, we have performed a taboo search with a considerable number of iterations (for instances that small) with all possible parameters (d_{min}, Δ). The number of optimal solutions found for each couple was then counted. If the search succeeds in finding all the 500 optimal solutions, the average number of iterations needed to reach the optimum is recorded.

With a deterministic taboo duration ($\Delta = 0$), it was never possible to achieve all the optimal solutions, even with a relatively large duration. Conversely, with

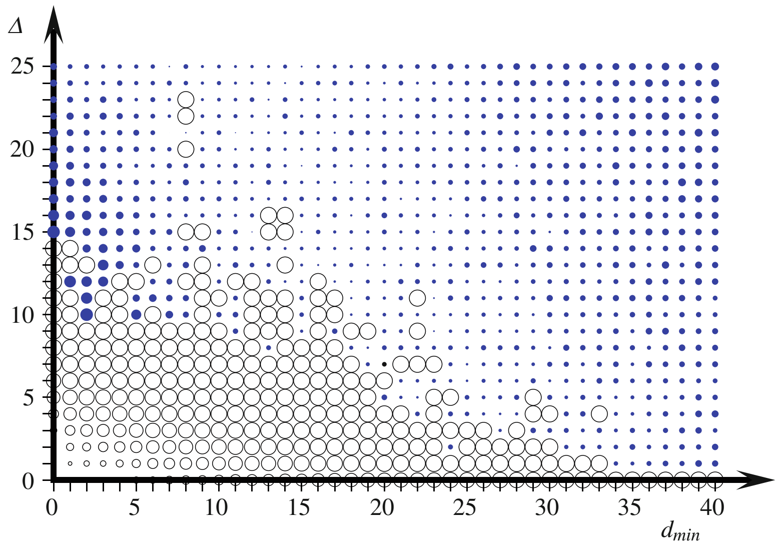

Fig. 9.2 Taboo duration randomly generated between d_{min} and $d_{min} + \Delta$. An empty circle indicates that taboo search has been unable to systematically find the optimum of 500 QAP instances. The circle size is proportional to the number of optimum found (the larger, the better). A filled disc indicates that the optimum has been systematically found. The disk size is proportional to the average number of iterations required for obtaining the optimum (the smaller, the better)

low minimum durations and a random variation equals to the size of the problem, the optimum is systematically obtained. Moreover, the optimum is reached with relatively few iterations.

Figure 9.3 reproduces a similar experiment for the TSP. It provides the solution quality obtained for a small instance for any couple (d_{min}, Δ). We observe similarities with Fig. 9.2. A random taboo duration proportional to half the instance size is a reasonable compromise.

9.1.2.3 Aspiration Criterion

The unconditional prohibition of a move can cause unwanted situations. For instance, one can skip an improvement of the best solution found. Thus, Line 4 of Algorithm 9.1 is modified, and if the move m allows achieving a solution better than s^*, it is retained. In the literature, this is referred to as an *aspiration criterion*. Other less trivial aspiration criteria can be imagined, in particular to implement a long-term memory.

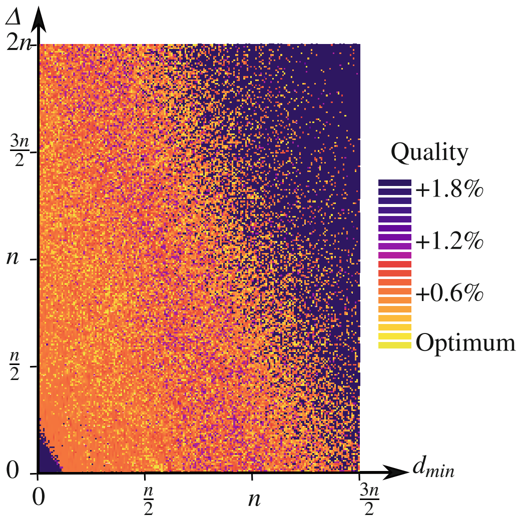

Fig. 9.3 Quality of the solutions obtained with a taboo search where the taboo duration is randomly chosen between d_{min} and $d_{min} + \Delta$ for a classical instance with $n = 127$ cities. The method performs $10n$ iterations starting from a deterministic greedy nearest neighbor tour. For all values of d_{min} between 0 and $1.5n$ and Δ between 0 and $2n$, we launched a search and represented the solution quality by a color (% above optimum)

9.2 Strategic Oscillations

Forbidding the inverse of moves recently performed implements a short-term memory. This mechanism can be very efficient for instances of moderate size. By cons, if we address more complex problems, this sole mechanism is not sufficient. A search strategy that has been proposed in the context of taboo search is to alternate *intensification and diversification* phases.

The goal of intensification is to thoroughly examine a limited portion of the search space, maintaining solutions that possess a globally similar structure. Once all the attractive solutions of this portion are supposedly discovered, the search has to go elsewhere. Put differently, the search is diversified by altering the structure of the solution. The search intensification can be implemented with a short duration taboo list.

9.2.1 Long-Term Memory

Implementing a diversification mechanism supposes to include a long-term memory. Several techniques have been proposed to achieve that.

9.2.1.1 Forced Moves

The most certain and convenient way to break the structure of a solution is to perform moves that have never been selected during many iterations. With a basic taboo search memorizing the iteration from which each move can again be performed, the implementation of this form of long-term memory is virtually free. Indeed, if the iteration number stored for a move is considerably smaller than the current iteration, then this move has not been selected for a long time.

It is thus possible to force the use of this modification, regardless of the quality of the solution to which it leads. This mechanism requires a new parameter, K, representing the number of iterations from which a never chosen move is forced. Naturally, this parameter must be larger than the size of the neighborhood; otherwise the search degenerates, performing only forced moves. If several moves are to be forced at a given iteration, one is chosen arbitrarily. The others will be forced in subsequent iterations. This type of long-term memory represents a kind of aspiration criterion, introduced in the previous section.

9.2.1.2 Penalized Moves

A weakness of taboo search with very short-term memory is that it only makes small changes. To illustrate this on the TSP, such a search will “knit” a small knot on a tour that was locally optimal, then another elsewhere and so on until the taboo condition drops. At that point, the search unknits the first knot. This situation is illustrated in Fig. 9.4.

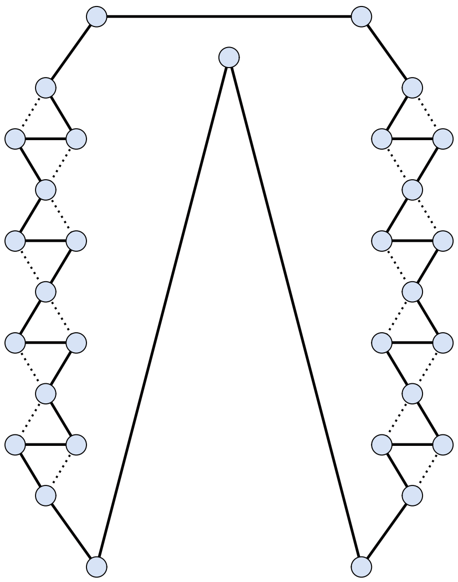

Fig. 9.4 A basic taboo search with a short-term memory can enter cycling with this 2-optimal tour. Indeed, this solution belongs to a plateau. There are eight moves not changing the tour length (dotted lines). With a taboo duration shorter than eight, the search repeatedly chooses one of these moves

To avoid this behavior, an idea is to store the number of times f_m a move m was chosen and limit its use. During the move evaluation, a penalty $F \cdot f_m$ is added. The proportionality factor F is a new parameter of the technique that must be tuned. Naturally, an aspiration criterion must be used in conjunction with this mechanism. Indeed, the search should nevertheless be allowed choosing a heavily penalized move leading to an improvement of the best solution known.

Code 9.1 provides a taboo search implementation for the TSP, based on the 2-opt neighborhood. Two types of memories are employed: a short-term conventional memory that prevents moves from reintroducing both edges that have been recently removed and a long-term memory that counts the number of times each edge has been inserted in the solution.

A move is penalized proportionally to the number of times the concerned edges have been introduced in the solution. A move is forbidden if both edges have recently been removed from the solution (eventually at different iterations). Ultimately, a move is aspired if it improves on the best solution achieved so far.

Code 9.1 tsp_TS.py Taboo search implementation for the TSP

```
from random_generators import *                                 # Listing 12.1

######### Taboo Search for the TSP, based on 2-opt moves
def tsp_TS(d,                                                   # Distance matrix
           tour,                                           # Intital tour provided
           length,                                        # Length of initial tour
           iterations,                              # Number of tabu search iterations
           min_tabu_duration,                              # Minimal tabu duration
           max_tabu_duration,
           F):                 # Factor for penalizing moves repeatedly performed

    n = len(tour)
    tabu = [[0] * n for _ in range(n)]                                  # Tabu list
    count = [[0] * n for _ in range(n)]                                # Move count
    best_tour = tour[:]
    best_length = length
    for iteration in range(0, iterations):
        delta_penalty = float('inf')
        ir = jr = -1                     # Cities retained for performing a move

        # Find best move allowed or aspired
        for i in range(n - 2):
            j = i + 2
            while j < n and (i > 0 or j < n - 1):
                delta =  d[tour[i]][tour[j]]  + d[tour[i+1]][tour[(j+1) % n]]\
                    -d[tour[i]][tour[i + 1]]  - d[tour[j]][tour[(j + 1) % n]]
                penality =  F * (count[tour[i]][tour[j]]
                                 + count[tour[i + 1]][tour[(j + 1) % n]])
                # Conditions for accepting a candidate move
                better = delta + penality < delta_penalty
                allowed = tabu[tour[i]][tour[j]] <= iteration \
                          or  tabu[tour[i + 1]][tour[(j + 1) % n]] <= iteration
                aspirated = length + delta < best_length

                if better and (allowed or aspirated):
                    delta_penalty = delta + penality
                    ir, jr = i, j
                j += 1                                              # Next neighbor

        # Perform retained move
        if delta_penalty < float('inf'):
            tabu[tour[ir]][tour[ir + 1]] = tabu[tour[jr]][tour[(jr + 1) % n]] \
            = tabu[tour[ir + 1]][tour[ir]] = tabu[tour[(jr+1) % n]][tour[jr]] \
            = unif(min_tabu_duration, max_tabu_duration) + iteration

            count[tour[ir]][tour[ir + 1]] += 1
            count[tour[jr]][tour[(jr + 1) % n]] += 1
            count[tour[ir + 1]][tour[ir]] += 1
            count[tour[(jr + 1) % n]][tour[jr]] += 1

            length += d[tour[ir]][tour[jr]] + d[tour[ir+1]][tour[(jr+1) % n]]\
                     -d[tour[ir]][tour[ir+1]] - d[tour[jr]][tour[(jr+1) % n]]

            for k in range((jr - ir) // 2):
                tour[k + ir + 1], tour[jr - k] = tour[jr - k], tour[k + ir + 1]
        else:
            print('All moves are forbidden tabu list too long')
        if best_length > length:                       # is there an improvement?
            best_length = length
            best_tour = tour[:]
            print('TS {:d} {:d}'.format(iteration+1, best_length))
    return best_tour, best_length
```

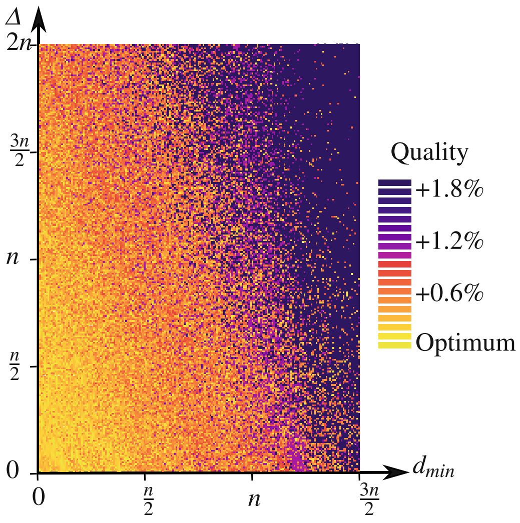

Fig. 9.5 Same diagram as Fig. 9.3 but with a taboo search managing a long-term memory. Frequently performed moves are penalized. The value of the penalty is $F \cdot n_e$, where n_e is the number of times the edge e has been included in or removed from the tour and the value of F is the average length of an edge divided by the instance size

Figure 9.5 illustrates the quality of this taboo search performing $10n$ iterations on a TSP instance with $n = 127$ cities. The search implements the penalty mechanism based on the frequency of moves. Compared to a taboo search not employing this mechanism (Fig. 9.3), the taboo duration can be reduced and the search achieves good solutions more frequently. This mechanism could even be operated alone, without a taboo list. Indeed, an excellent solution is obtained with a minimum and maximum taboo duration of 0. A taboo list can be implemented by means of a matrix whose entry (i, j) gives the iteration number from which one can again use the edge $[i, j]$ in a move. Counting the frequency of moves is implemented in a similar way.

9.2.1.3 Restarts

A frequently used technique to intensify a taboo search is to restart with the best solution achieved so far. This is done if the search seems to stagnate, for instance, if there has been no improvement in the best solution during a relatively significant number of iterations. When restarting, the information collected during the previous iterations by the taboo list is kept, as well as other statistics, if any. Hence, the work achieved during these iterations is exploited.

Thus, the data structures guiding the search being in an altered state after restarting, the trajectory followed by the search, will also be. This mechanism can be identified as the opposite of the one presented above where we force the use of neglected attributes for many iterations. Its purpose is to achieve search intensification, not diversification. Naturally, the implementation of this mechanism implies the introduction of new parameters that must be adjusted, like the number of iterations to be carried out before a restart and a possible adaptation of the value of other parameters (taboo duration, frequency penalty) to guide the search toward diversified trajectories.

Problems

9.1 Taboo Search for an Explicit Function

An integer function of integer variables $f(x, y)$ is explicitly given in Table 5.1. We seek the minimum of this function by applying a taboo search. The neighborhood consists in modifying by a unit the value of one variable. The taboo conditions consist in forbidding to increment (respectively: to decrement) a variable that has been decremented (respectively: incremented). First, consider a taboo duration of $d = 3$ and $(-7, -6)$ as the starting solution. Next, start from $(-7, 7)$ and use $d = 1$. The search stops if there is no more move allowed or if 25 iterations have been performed.

9.2 Taboo Search for the VRP

For the VRP, the neighborhood consists in either moving a customer from one route to another, or swapping two customers from different routes. Suggest taboo criteria for this neighborhood.

9.3 Taboo Search for the QAP

Consider the QAP instance given by the flow F and distance D matrices:

$$F = \begin{bmatrix} 0 & 5 & 2 & 4 & 1 \\ 5 & 0 & 3 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 5 \\ 1 & 2 & 0 & 5 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 1 & 1 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 2 & 1 & 0 \end{bmatrix}$$

Starting with the solution $p = (1, 2, 3, 4, 5)$, perform six iterations of a taboo search. The moves are defined by pairs (i, j) that swap the elements p_i and p_j. If the move (i, j) is performed, then, it is forbidden for $d = 5$ iterations to place the element p_i in position i and, simultaneously, the element p_j in position j. For each iteration, provide the solution, its value, that of all the moves and their taboo status.

References

1. Battiti, R., Tecchiolli, G.: The reactive tabu search. ORSA J. Comput. **6**, 126–140 (1994). https://doi.org/10.1287/ijoc.6.2.126
2. Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. **13**(5), 533–549 (1986). https://doi.org/10.1016/0305-0548(86)90048-1

3. Glover, F.: Tabu search—part I. ORSA J. Comput. **1**(3), 190–206 (1989). https://doi.org/10.1287/ijoc.1.3.190
4. Glover, F.: Tabu search—part II. ORSA J. Comput. **2**(1), 4–32 (1990). https://doi.org/10.1287/ijoc.2.1.4
5. Glover, F., Laguna, M.: Tabu Search. Kluwer, Dordrecht (1997)
6. Taillard, É.D.: Comparison of iterative searches for the quadratic assignment problem. Location Science **3**(2), 87–105 (1995). https://doi.org/10.1016/0966-8349(95)00008-6