Chapter 3
Tabu Search

Eric Taillard

3.1 Introduction

Tabu search was first proposed by Fred Glover in an article published in 1986 [3], although it borrowed many ideas suggested before during the 1960s. The two articles entitled simply "Tabu search" [4, 5] proposed most of tabu search principles which are currently used. Some of these principles did not gain prominence among the scientific community for a long time. Indeed, in the first half of the 1990s, the majority of the research publications on tabu search used a very restricted range of the principles of the technique. They were often limited to a *tabu list* and an elementary *aspiration condition*.

The popularity of tabu search is certainly due to the contribution of de Werra's team at the Federal Polytechnic School of Lausanne during the late 1980s. In fact, the articles by Glover, the founder of the method, were not well understood at the time, when there was not yet a "culture" of metaheuristics-based algorithms. Hence the credit for the popularization of the basic technique must go to [8, 10], which surely played a significant role in the dissemination of the algorithm.

At the same time, competition developed between simulated annealing (which then had an established convergence theorem as its theoretical advantage) and tabu search. For many applications, tabu-search-based heuristics definitely showed more effective results [12–15], which increased the interest in the method among some researchers.

At the beginning of the 1990s, the technique was extensively explored in Canada, particularly in the Center for Research on Transportation in Montreal, where several postdoctoral researchers from de Werra's team worked in this field. This created another focus in the field of tabu search. The technique was then quickly disseminated

E. Taillard (✉)
HEIG-VD, 1401 Route de Cheseaux 1, Cp, Yverdon-les-bains, Switzerland
e-mail: eric.taillard@heig-vd.ch

© Springer International Publishing Switzerland 2016

P. Siarry (ed.), *Metaheuristics*, DOI 10.1007/978-3-319-45403-0_3

among several research communities, and this culminated in the publishing of the first book which was solely dedicated to tabu search [7].

In this chapter, we shall not deal with all of the principles of tabu search presented in the book by Fred Glover and Manuel Laguna [6], but instead we shall focus on the most significant and most general principles.

What unquestionably distinguishes it from the local search technique presented in the preceding chapter is that tabu search incorporates intelligence. Indeed, there is a huge temptation to guide an iterative search in a good, promissing, direction, so that it is not guided merely by chance and the value of an objective function to be optimized. Implementing a tabu search gives rise to a couple of challenges: first, as in any iterative search, it is necessary that the search engine, i.e., the mechanism for evaluating neighboring solutions, is effective; and second, pieces of knowledge regarding the problem under consideration should be transmitted to the search procedure so that it will not get trapped in bad regions of the solution space. On the contrary, it should be guided intelligently in the solution space, if such a term is permitted to be used.

Glover proposed a number of learning techniques that can be embedded in a local search. One of the guiding principles is to construct a history of the iterative search or, equivalently, to equip the search with memory.

- *Short-term memory*. The name "tabu search" is a reference to the use of a short-term memory that is embedded in a local search. The idea is to memorize in a data structure T the elements that the local search is prohibited from using. This structure is called a *tabu list*. In its simplest form, a tabu search scans the whole set of neighboring solutions in each iteration and chooses the best that is not forbidden, even if it is worse than the current solution. To prevent the search being blocked or being forced to visit only solutions of bad quality owing to tabu conditions, the number of elements in T is limited. Since the number of prohibitions contained in T is limited—this number is frequently called the *tabu list size*—this mechanism implements a short-term memory.
- *Long-term memory*. A tabu list does not necessarily prevent a cycling phenomenon, that is, visiting a subset of solutions cyclically. If the tabu duration is long enough to avoid a cycling phenomenon, the search may be forced to visit only bad solutions. To avoid both of these complementary phenomena, another kind of memory, operating over a longer term, must be used.
- *Diversification*. A technique for avoiding cycling, which is the basis of variable neighborhood search, is to perform jumps in the solution space. But, in contrast to variable neighborhood search, which performs random jumps, tabu search uses a long-term memory for these jumps, for instance by forcing the use of solution modifications that have not been tried for a large number of iterations. Another diversification technique is to change the modeling of the problem, for instance by accepting nonfeasible solutions but assigning them a penalty.
- *Intensification*. When an interesting solution is identified, an idea is to tentatively examine more deeply the solution space in its neighborhood. Many intensification techniques are used. The simplest one is to come back to the best solution found so far and to change the search parameters, for instance by limiting the tabu duration,

by using a larger neighborhood, or by using a more constrainted model of the problem.

- *Strategic oscillations*. For tackling particularly difficult problems, it is convenient to alternate phases of diversification and intensification. So, the search oscillates between phases where the solution structure is greatly modified and phases where better solutions are built again. This strategy is the origin of other metaheuristics that were proposed later, such as variable neighborhood search, large neighborhood search, and iterated local search.

Some of these principles of tabu search will be illustrated with the help of a particular problem, namely the quadratic assignment problem, so that these principles does not stay in the clouds. We chose this problem for several reasons. First of all, it has applications in multiple fields. For example, the problem of placing electronic modules, which we discussed in Chap. 1 devoted to simulated annealing, is a quadratic assignment problem. In this case, its formulation is very simple, because it deals with finding a permutation. Here, it should be noted that many combinatorial optimization problems can be expressed in the form of searching for a permutation.

3.2 The Quadratic Assignment Problem

Given n objects and a set of flows f_{ij} between objects i and j ($i, j = 1, \ldots, n$), and given n locations with distance d_{rs} between the locations r and s ($r, s = 1, \ldots, n$), the problem deals with placing the n objects on the n locations so as to minimize the sum of the products flows $\times$ distance. Mathematically, this is equivalent to finding a permutation $\mathbf{p}$, whose ith component p_i denotes the position of the object i, which minimizes $\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{p_i p_j}$.

This problem has multiple practical applications; among them, the most popular ones are the assignment of offices or services in a building (e.g., a university campus or hospital), the assignment of departure gates to aircraft at an airport, the placement of logical modules in FPGA (field-programmable gate array) circuits, the distribution of files in a database, and the placement of the keys on typewriter keyboards. In these examples, the flow matrix represents the frequency with which people may move from one office to another, the number of people who may transit from one aircraft to another, the number of electrical connections to be made between two modules, the probability of requesting the access to a second file if one is accessing the first one, and the frequency with which two particular characters appear consecutively in a given language, respectively. The distance matrix has an obvious meaning in the first three examples; in the fourth, it represents the transmission time between databases and, in the fifth, it represents the time separating the striking of two keys.

The quadratic assignment problem is NP-hard. One can easily show this by noting that the traveling salesman problem can be formulated as a quadratic assignment problem. Unless P = NP, there is no polynomial approximation scheme for this problem. This can be shown simply by considering two problem instances which

Table 3.1 Number of connections between modules in the SCR12 problem

Module	1	2	3	4	5	6	7	8	9	10	11	12
1	—	180	120	—	—	—	—	—	—	104	112	—
2	180	—	96	2445	78	—	1395	—	120	135	—	—
3	120	96	—	—	—	221	—	—	315	390	—	—
4	—	2445	—	—	108	570	750	—	234	—	—	140
5	—	78	—	108	—	—	225	135	—	156	—	—
6	—	—	221	570	—	—	615	—	—	—	—	45
7	—	1395	—	750	225	615	—	2400	—	187	—	—
8	—	—	—	—	135	—	2400	—	—	—	—	—
9	—	120	315	234	—	—	—	—	—	—	—	—
10	104	135	390	—	156	—	187	—	—	—	36	1200
11	112	—	—	—	—	—	—	—	—	36	—	225
12	—	—	—	140	—	45	—	—	—	1200	225	—

differ only in the flow matrix. If an appropriate constant is removed from all the flow components of the first problem to obtain the second, the last have an optimum solution value of zero. Consequently, all ϵ-approximations to the second problem has an optimum solution, which is possible to implement in polynomial time only if P = NP. However, problems generated at random (with flows and distances drawn uniformly) satisfy the following property: as $n \to \infty$, the value of any solution (even the worst one) tends towards the value of an optimal solution [1].

3.2.1 Example

Let us consider the placement of 12 electronic modules $(1, \ldots, 12)$ on 12 sites $(a, b, \ldots, l)$. The number of wires connecting any pair of modules is known, and is given in Table 3.1. This problem instance is referred to as SCR12 in the literature.

The sites are distributed on a 3×4 rectangle. Connections can be implemented only horizontally or vertically, implying wiring lengths measured with Manhattan distances. In the solution of the problem represented in Fig. 3.1, which is optimal, module 6 was placed on site a, module 4 on site b, etc.

3.3 Basic Tabu Search

From here onwards and without being restrictive, we can make the assumption that the problem to be solved can be formulated in the following manner:

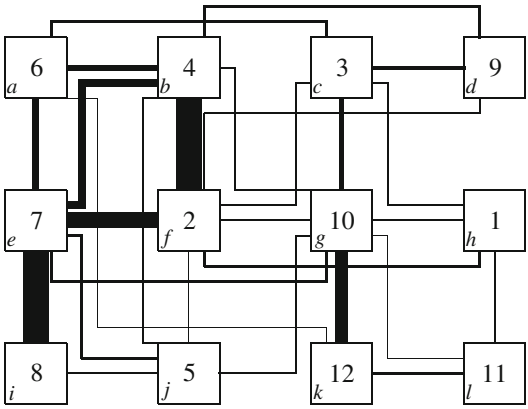

Fig. 3.1 Optimal solution of a problem of connection between electronic modules. The thickness of the lines is proportional to the number of connections

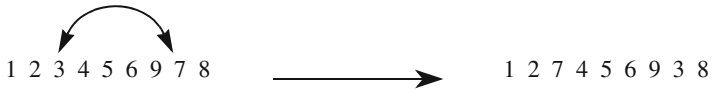

Fig. 3.2 A possibility for creating a neighboring solution in a permutation problem

$$\min_{s \in S} f(s)$$

In this formulation, f denotes the objective function, s a feasible solution of the problem, and S the entire collection of feasible solutions.

3.3.1 Neighborhood

Tabu search is primarily centered on a nontrivial exploration of all the solutions by using the concept of a neighborhood. Formally, one can define, for any solution s of S, a set $N(s) \subset S$ that is a collection of the neighboring solutions of s. For instance, for the quadratic assignment problem, s can be a permutation of n objects and the set $N(s)$ can be the possible solutions obtained by exchanging two objects in a permutation. Figure 3.2 illustrates one of the moves of the set $N(s)$, where objects 3 and 7 are exchanged.

Local search methods are almost as old as the world itself. As a matter of fact, how does a human being behave when they are seeking a solution to a problem for which they cannot find a solution or if they do not have enough patience to find an optimal solution? The person may try to slightly modify the proposed solution and may check whether it is possible to find better solutions by carrying out such local changes. In other words, they will stop as soon as they meet a *local optimum* relative

to the modifications to a solution that are allowed. In this process, nothing proves that the solution thus obtained is a *global optimum*—and, in practice, this is seldom the case. In order to find solutions better than the first local optimum met with, one can try to continue the process of local modifications. However, if precautions are not taken, one risks visiting a restricted number of solutions, in a cyclic manner. Simulated annealing and tabu search are two local search techniques which try to eliminate this disadvantage.

Some of these methods, such as, simulated annealing, have been classified as artificial intelligence techniques. However, this classification is certainly incorrect, as they are guided almost exclusively by chance—some authors even compare simulated annealing to the wandering of a person suffering from amnesia moving in fog. Perhaps others describe these methods as intelligent because, often after a large number of iterations during which they have generated several poor-quality solutions, they produce a good-quality solution which would otherwise have required a very large human effort.

In essence, tabu search is not centered on chance, although one can introduce random components for primarily technical reasons. The basic idea of tabu search is to make use of memories during the exploration of some part of the solutions to the problem, which consists in moving repeatedly from one solution to a neighboring solution. It is thus primarily a local search, if we look beyond the limited meaning of this term and dig for a broader meaning. However, some principles that enable one to carry out jumps in the solution space have been proposed; in this respect, tabu search, in contrast to simulated annealing, is not a pure local search.

3.3.2 Moves and Neighborhoods

Local searches are based on the definition of a set $N(s)$ of solutions in the neighborhood of s. But, from a practical point of view, it may be easier to consider the set M of modifications that can be applied to s, rather than the set $N(s)$. A modification made to a solution can be called a *move*. Thus, a modification of a solution of the quadratic assignment problem (see the Fig. 3.2) can be considered as a move characterized by two elements to be transposed in a permutation. Figure 3.3 gives the neighborhood structure for the set of permutations of four elements. This is presented in graphical form, where the nodes represent the solutions and the edges represent the neighbors relative to transpositions.

The set $N(s)$ of solutions in the neighborhood of s can be expressed as the set of feasible solutions that can be obtained by applying a move m to solution s, where m belongs to a set of moves M. The application of m to s can be denoted as $s \oplus m$ and one has the equivalent definition $N(s) = \{s' | = s \oplus m, m \in M\}$. When it is possible, expressing the neighborhood in terms of moves facilitates the characterization of the set M. Thus, in the above example of modification of a permutation, M can be characterized by all of the pairs (place 1, place 2) in which the elements are transposed, independently from the current solution. Note that in the case of permutations with

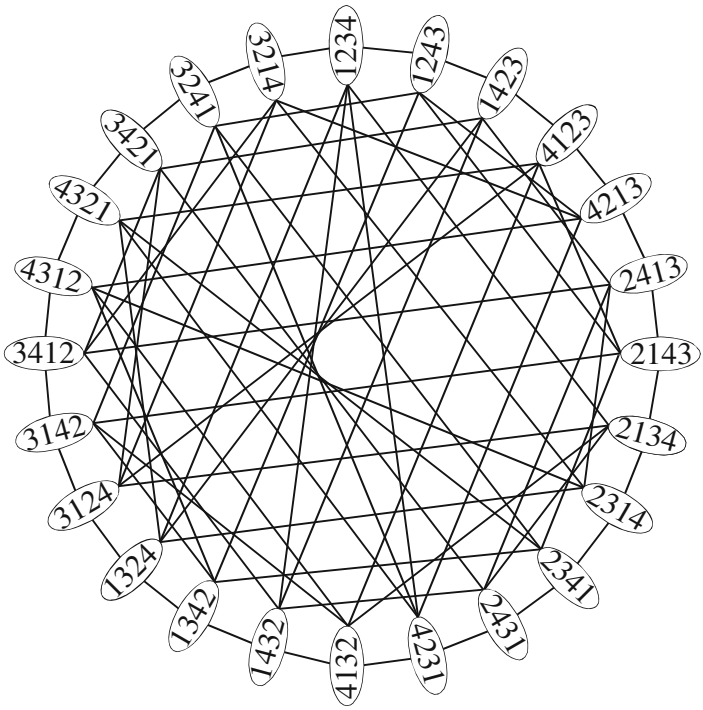

Fig. 3.3 Set of permutations of four elements (represented by nodes) with neighborhood relations relative to transpositions (represented by edges)

a transposition neighborhood, $|S| = n!$ and $|M| = {}^{n\cdot(n-1)}/_{2}$. Thus, the solution set is much larger than the set of moves, which grows as the square of the number of elements.

However, in some applications this simplification can lead to the definition of moves which would produce unacceptable solutions and, in general, we have $|N(s)| \leq |M|$, without $|M|$ being much larger than $|N(s)|$. For a given problem with few constraints, it is typically the case that $|N(s)| = |M|$.

3.3.2.1 Examples of Neighborhoods for Problems on Permutations

Many combinatorial optimization problems can be formulated naturally as a search for a permutation of n elements. Assignment problems (which include the quadratic assignment problem), and the traveling salesman and scheduling problems are representative examples of such problems. For these problems, several definitions of neighboring solutions are possible; some examples are illustrated in Fig. 3.4. Among the simplest neighborhoods, one can find the inversion of two elements placed

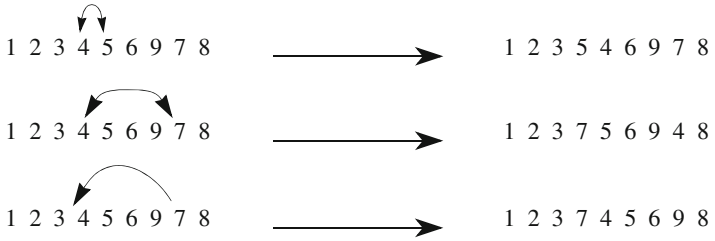

Fig. 3.4 Three possible neighborhoods with respect to permutations (inversion, transposition, displacement)

successively in the permutation, the transposition of two distinct elements, and, finally, the movement of an element to another place in the permutation. Depending on the problem considered, more elaborate neighborhoods that suit the structure of good solutions may be considered. This is typically the case for the traveling salesman problem, where there are innumerable neighborhoods suggested that do not represent simple operations if a solution is regarded as a permutation.

The first type of neighborhood shown in the example in Fig. 3.4 is the most limited one as it is of size $n-1$. The second type defines a neighborhood with, $^{n\cdot(n-1)}/_{2}$ moves, and the third is of size $n(n-2)+1$. The abilities of these various types of neighborhoods to guide a search in a few iterations towards good solutions are very different; generally, the first type shows the worst behavior for many problems since it is a subset of the others. The second type can be better than the third for some problems (such as the quadratic assignment problem), whereas, for scheduling applications, the third type often shows better performance [12].

3.3.3 Neighborhood Evaluation

In order to implement an effective local search engine, it is necessary that the ratio between the quality or suitability of the moves and the computational resources required for their evaluation is as high as possible. If the quality of a type of move can be justified only by intuition and in an empirical manner, the evaluation of the neighborhood can, on the other hand, often be accelerated considerably by algebraic considerations. Let us define $\Delta(s,m) = f(s \oplus m) - f(s)$. In many cases it is possible to simplify the expression $f(s \oplus m) - f(s)$ and thus to evaluate $\Delta(s,m)$ quickly. An analogy can be drawn with continuous optimization: the numerical evaluation of $f(s \oplus m) - f(s)$ would be the equivalent of a numerical evaluation of the gradient, whereas the calculation of the simplified function $\Delta(s,m)$ would be the equivalent of the evaluation of the gradient by means of a function implemented using the algebraic expressions for the partial derivatives.

Moreover, if a move m' was applied to solution s in the previous iteration, it is often possible to evaluate $\Delta(s \oplus m', m)$ for the current iteration as a function of

$\Delta(s, m)$ (which was evaluated in the previous iteration) and to evaluate the entire neighborhood very rapidly, simply by memorizing the values of $\Delta(s, m), \forall m \in M$.

It may appear that $\Delta(s, m)$ would be very difficult and expensive to evaluate. For instance, for vehicle routing problems (see Sect. 13.1), a solution s can consist of partitioning customers demands into subsets whose weights are not more than the capacities of the vehicles. To evaluate $f(s)$, we have to find an optimal order in which one will deliver to the customers for each subset, which is a difficult problem in itself. This is the well-known traveling salesman problem. Therefore, the evaluation of $f(s)$, and consequently that of $\Delta(s, m)$ cannot reasonably be performed for every eligible move (i.e., all moves belonging to M); possibly $\Delta(s, m)$ would need to be calculated for each move selected (and in fact carried out), but, in practice, $f(s)$ is evaluated exactly for a limited number of solutions only. Hence the computational complexity is limited by estimating $\Delta(s, m)$ in an approximate manner.

3.3.3.1 Algebraic Simplification for the Quadratic Assignment Problem

As any permutation is an acceptable solution for the quadratic assignment problem, its modeling is also trivial. For the choices of neighborhood, it should be realized that moving the element into the ith position in the permutation to put it into the jth position implies a very significant modification of the solution. This is because all of the elements between the ith and the jth position are moved. The inversion of the objects in the ith and the $(i + 1)$th position in the permutation generates too limited a neighborhood. In fact, if the objective is to limit ourselves to the neighborhoods in which the sites assigned to two elements only are modified, it is only reasonable to transpose the elements i and j occupying the sites p_i and p_j. Each of these moves can be evaluated in $O(n)$ (where n is the problem size). With a flow matrix $\mathcal{F} = (f_{ij})$ and a distance matrix $\mathcal{D} = (d_{rs})$, the value of move $m = (i, j)$ for a solution $\mathbf{p}$ is given by

$$\begin{aligned}\Delta(\mathbf{p}, (i, j)) = {} & (f_{ii} - f_{jj})(d_{p_j p_j} - d_{p_i p_i}) + (f_{ij} - f_{ji})(d_{p_j p_i} - d_{p_i p_j}) \\ & + \textstyle\sum_{k \neq i, j} (f_{jk} - f_{ik})(d_{p_i p_k} - d_{p_j p_k}) + (f_{kj} - f_{ki})(d_{p_k p_i} - d_{p_k p_j})\end{aligned} \tag{3.1}$$

If solution $\mathbf{p}$ was modified into solution $\mathbf{q}$ by exchanging the objects r and s, i.e., $q_k = p_k$ $(k \neq r, k \neq s)$, $q_r = p_s$, $q_s = p_r$ in an iteration, it is possible to evaluate $\Delta(\mathbf{q}, (i, j))$ in $O(1)$ in the next iteration by memorizing the value $\Delta(\mathbf{p}, (i, j))$ of the move (i, j) that was discarded:

$$\begin{aligned}\Delta(\mathbf{q}, (i, j)) = {} & \Delta(\mathbf{p}, (i, j)) \\ & + (f_{ri} - f_{rj} + f_{sj} - f_{si})(d_{q_s q_i} - d_{q_s q_j} + d_{q_r q_j} - d_{q_r q_i}) \\ & + (f_{ir} - f_{jr} + f_{js} - f_{is})(d_{q_i q_s} - d_{q_j q_s} + d_{q_j q_r} - d_{q_i q_r})\end{aligned} \tag{3.2}$$

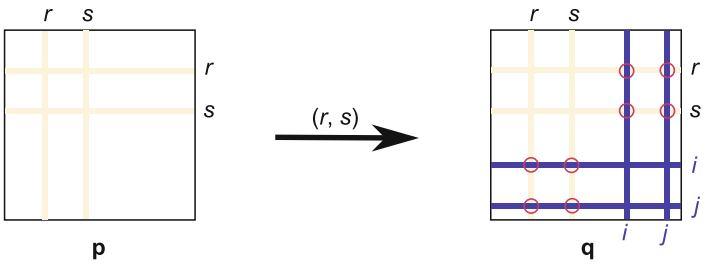

Fig. 3.5 *Left* in *light color*, the elements for which it is necessary to recalculate the scalar product of the matrices to evaluate the move (r, s) applied to **p** (giving the solution **q**). *Right* the *circled* elements are those for which it is necessary to recalculate the product to evaluate the move (i, j) applied to **q**, compared with those which would have been calculated if the move (i, j) had been applied to **p**

Figure 3.5 illustrates the modifications of $\Delta(\mathbf{p}, (i, j))$ that are necessary to obtain $\Delta(\mathbf{q}, (i, j))$ if the move selected for going from **p** to **q** is (r, s). It should be noted here that the quadratic assignment problem can be regarded as the problem of the permutation of the rows and columns of the distance matrix so that the "scalar" product of the two matrices is as small as possible.

Consequently, by memorizing the values of $\Delta(\mathbf{p}, (i, j))$ for all i and j, the complete neighborhood can be evaluated in $O(n^2)$: by using Eq. (3.2), one can evaluate the $O(n^2)$ moves that do not involve the indices r and s, and, by using Eq. 3.1, one can evaluate the $O(n)$ moves which involve precisely these indices.

3.3.4 Neighborhood Limitation: Candidate List

Generally, a local search does not necessarily evaluate all the solutions in $N(s)$ in each iteration, but only a subset. In fact, simulated annealing only evaluate a single neighbor in each iteration. Conversely, tabu search is supposed to make an "intelligent" choice of a solution from $N(s)$. A possible way to acclerate the evaluation of the neighborhood is to reduce its size; this reduction can also have the other goal of guiding the search.

To reduce the number of eligible solutions in $N(s)$, some authors adopt the strategy of randomly selecting from $N(s)$ a number of solutions which is much smaller than $|N(s)|$. If the neighborhood is given by a static collection M of moves, one can also consider partitioning M into subsets; in each iteration, only one of these subsets is examined. In this manner, one can use a partial but cyclic evaluation of the neighborhood, which allows one to choose a move more quickly. This implies a deterioration in quality, since not all moves are not taken into consideration in each iteration. However, at a global level, this limitation might not have too bad an influence on the quality of the solutions produced, because a partial examination can generate a certain diversity in the visited solutions, precisely because the moves

which were chosen were not those which would have been chosen, if a complete examination of the neighborhood had been carried out.

Finally, in accordance with Glover's intuition when he proposed the concept of a candidate list, one can make the assumption that a good-quality move for a solution will remain good for solutions that are not too different. Practically, this can be implemented by ordering the entire set of all feasible moves by decreasing quality, in a given iteration. During the later iterations, only those moves that have been classified among the best will be considered. This is implemented in the form of a data structure called a *candidate list*. Naturally, the order of the moves will become degraded during the search, since the solutions become increasingly different from the solution used to build the list, and it is therefore necessary to periodically evaluate the entire neighborhood to preserve a suitable candidate list.

However, for some problems, a static candidate list can be used. For instance, a frequently used technique for speeding up the evaluation of the neighborhood for Euclidean traveling salesman and vehicle routing problems is to consider, for each customer, only the x closest customers. Typically, x is limited to a few dozen. So, the size of the neighborhood grows linearly with the problem size. One form of tabu search exploiting this principle is called *granular tabu search* [17].

3.3.5 Neighborhood Extension: Ejection Chains

An *ejection chain* is a technique for creating potentially interesting neighborhoods by performing a substantial modification of a solution in a compound move. The idea is to remove (eject) an element from a solution and to insert it somewhere else, ejecting another element if necessary. This is repeated until either a suitable solution is found or no suitable ejection can be performed. This process implies a need to manage solutions that are not feasible, called *reference structures* by Glover.

3.3.5.1 Lin and Kernighan Neighborhood for the Traveling Salesman Problem

The best-known ejection chain technique is certainly that of Lin and Kernighan [11] for the traveling salesman problem. The idea is as follows: An edge is removed from a valid tour to obtain a chain (a nonoriented path). One of the extremities of the chain is connected to an internal vertex. The reference structure so obtained is made up of a cycle on a subset of vertices and a chain connected to this cycle. By removing an edge of the cycle adjacent to a node of degree 3 and by connecting both nodes of degree 1, a new feasible tour is obtained. Such a modification corresponds to the traditional 2-opt move.

An interesting exploitation of this reference structure is to transform it into another reference structure. After an edge adjacent to a node of degree 3 has been removed—one gets a chain connecting all vertices—one of the extremities of the chain can be connected to an internal node, creating another reference structure. To guide the ejection chain and determine when to stop, the following rules can be applied:

- The weight of the reference structure must be lower than that of the initial solution.
- Once an edge has been added during an ejection chain, this edge cannot be removed again.
- Once an edge has been removed during an ejection chain, it cannot be added again.
- The ejection chain stops as soon as it is not possible to modify the reference structure while maintaining a weight lower than that of the starting solution or when an improved solution has been found.

This process is illustrated in Fig. 3.6.

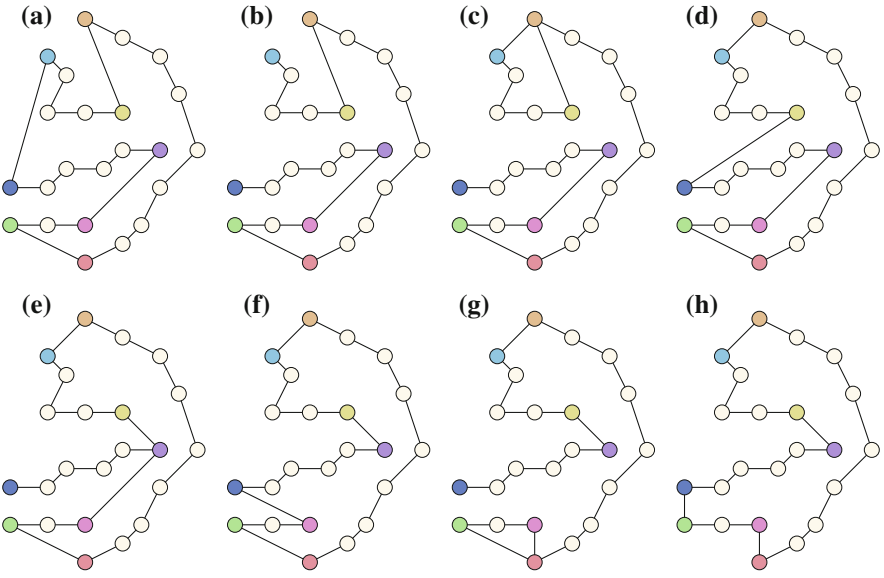

Fig. 3.6 Lin and Kernighan neighborhood for the traveling salesman problem. This neighborhood can be seen as an ejection chain. To start the chain, an edge is removed from the initial solution (**a**) to obtain a chain (**b**). This chain is then transformed into a reference structure (**c**) of weight lower than that of the initial solution by adding an edge. From the reference structure (**c**), it is possible to get either a new tour (**d**) or another reference structure (**e**) by replacing an edge by another one. The process can be propagated to construct solutions that are increasingly different from the initial solution. Solution (**d**) belongs to the 2-opt neighborhood of solution (**a**). Solution (**f**) belongs to the 3-opt neighborhood of (**a**) and solution (**h**) to its 4-opt neighborhood

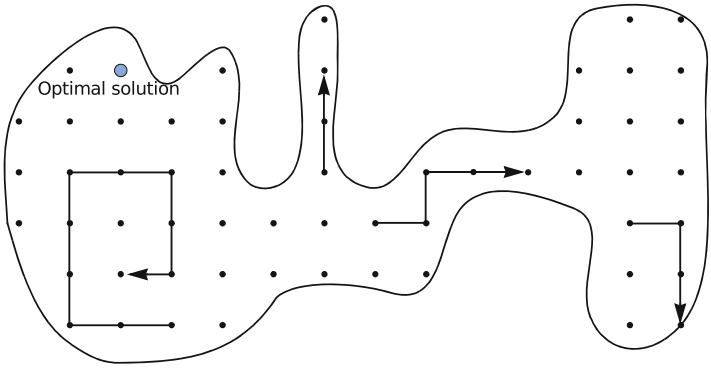

Fig. 3.7 Trajectories that block a search or disconnect it from the optimal solution in a strict tabu search

3.4 Short-Term Memory

When one wishes to make use of memory in an iterative process, the first idea that comes to mind is to check if a solution in the neighborhood has already been visited. However, the practical implementation of this idea can be difficult and, even worse, it may prove not to be very effective. It requires one to memorize every solution visited and, in each iteration, for every eligible solution, to test whether that solution has already been enumerated. This could possibly be done efficiently by using hashing tables, but it is not possible to prevent a significant growth in the memory space requirement, as that increases linearly with the number of iterations. Moreover, the pure and simple prohibition of solutions can lead to absurd situations. Assume that the entire set of feasible solutions can be represented by points whose coordinates are given on a surface in the plane and that one can move from any feasible solution to any other by a number of displacements of unit length. In this case, one can easily find trajectories which disconnect the current solution from an optimal solution or which block an iterative search owing to a lack of feasible neighboring solutions if it is tabu to pass through an already visited solution. This situation is schematically illustrated in Fig. 3.7.

3.4.1 Hash Table

An initial idea for guiding an iterative search, which is very easy to implement, is to prohibit a return to a solution whose value has already been obtained during the last t iterations. Thus one can prevent a cycle of length t or less. This type of prohibition can be implemented in an effective manner: Let L be a integer, relatively large, such that it is possible to store a table of L entries in the main memory of the computer. If $f(s_k)$ is assumed to be the integer value of solution s_k in iteration k (this is not

restrictive when one is working with a computer), one can memorize the value $k + t$ in $T[f(s_k) \text{ modulo } L]$. If a solution s' in the neighborhood of the solution in iteration k' is such that $T[f(s') \text{ modulo } L)] > k'$, s' is not considered anymore as an eligible solution. This effective method of storing the tabu solutions only approximates the initial intention, which was to prevent a return to a solution of a given value, as not only any solution of the given value is prohibited during t iterations but also all those which have this value modulo L. Nevertheless, only a very moderate modification of the search behavior can be observed in practice if L is selected to be sufficiently large. A benefit of this collateral effect is that it suppresses neutral moves (moves with null cost), which can trap a local search on a large plateau.

This type of tabu condition works only if the objective function has a vast span of values. However, there are many problems where the objective function has a limited span of values. One can circumvent this difficulty by associating with the objective function, and using in its place, another function that has a large span of values. In the case of a problem on permutations, one can associate, for example, the hashing function $\sum_{i=1}^{n} i^2 \cdot p_i$, which takes a number of $O(n^4)$ different values.

More generally, if a solution of a problem can be expressed in the form of a vector $\mathbf{x}$ of binary variables, one can associate the hashing function $\sum_{i=1}^{n} z_i \cdot x_i$ with z_i, a set of n numbers randomly generated at the beginning of the search [18].

When hashing functions are used for implementing tabu conditions, one needs to focus on three points. Firstly, as already mentioned, it is necessary that the function used has a large span of possible values. Secondly, the evaluation of the hashing function for a neighboring solution should not impose a significantly higher computational burden than the evaluation of the objective function. In the case of problems on permutations with a neighborhood based on transpositions, the functions mentioned above can be evaluated in constant time for each neighboring solution if the value of the hashing function for the starting solution is known. Thirdly, it should be noted that even with a very large hashing table, collisions (different solutions with identical hashing values) are frequent. Thus, for a problem on permutations of size $n = 100$, with the transposition neighborhood, approximately five solutions in the neighborhood of the solution in the second iteration will have a collision with the starting solution, if a table of 10^6 elements is used. One technique to reduce the risk of collisions effectively, is to use several hashing functions and several tables simultaneously [16].

3.4.2 Tabu List

As it can be ineffective to restrict the neighborhood $N(s)$ to those solutions which have not yet been visited, tabu conditions are instead based on M, the set of moves applicable to a solution. This set is often of relatively modest size (typically $O(n)$ or $O(n^2)$ if n is the size of the problem) and must have the characteristic of *connectivity*, i.e., an optimal solution can be reached from any feasible solution. Initially, to simplify our analysis, we assume that M also has the property of *reversibility*: for any

move m applicable to a solution s, there is a move m^{-1} such that $(s \oplus m) \oplus m^{-1} = s$. As it does not make sense to apply m^{-1} immediately after applying m, it is possible, in all cases, to limit the moves applicable to $s \oplus m$ to those different from m^{-1}. Moreover, one can avoid visiting s and $s \oplus m$ repeatedly in the process if s is a local minimum of the function in the neighborhood selected and if the best neighbor of $s \oplus m$ is precisely s.

By generalizing this technique of limiting the neighborhood, i.e., by prohibiting for several iterations the reverse of a move which has just been made, one can prevent other cycles composed of a number of intermediate solutions. Once it again becomes possible to carry out the reverse of a move, one hopes that the solution has been sufficiently modified that it is improbable—but not impossible—to return to an already visited solution. Nevertheless, if such a situation arises, it is hoped that the tabu list would have changed, and therefore the future trajectory of the search would change. The number of tabu moves must remain sufficiently limited. Let us assume that M does not depend on the current solution. In this situation, it is reasonable to prohibit only a fraction of M. Thus, the tabu list implements a short-term memory, relating typically to a few or a few tens of iterations.

For easier understanding, we have assumed that the reverse moves of those that have been carried out are stored. However, it is not always possible or obvious to define what a reverse move is. Take the example of a problem where the objective is to find an optimal permutation of n elements. A reasonable move could be to transpose the elements i and j of the permutation $(1 \leq i < j \leq n)$. In this case, all of the M moves applicable to an unspecified solution are given by the entire set of pairs (i, j). But, thereafter, if the move (i, k) is carried out, the prohibition of (i, j) will prevent the visiting of certain solutions without preventing the cycling phenomenon: indeed, the moves $(i, j)(k, p)(i, p)(k, j)(k, i)(j, p)$ applied successively do not modify the solution. So, the tabu condition must not necessarily prohibit one to perform the reverse of a move too quickly, but it may be defined in such a way to prevent the use of some attribute of the moves or solutions. In the preceding example, if p_i is the position of element i and if the move (i, j) has been performed, it is not the reverse move (i, j) which should be prohibited, but, for example, the simultaneous placing of the element i on position p_i and the element j on position p_j. One can thus at least prevent those cycles which are of length less than or equal to the number of tabu moves, i.e., the length of the tabu list.

3.4.3 Duration of Tabu Conditions

Generally speaking, the short-term memory will prohibit the performance of some moves, either directly by storing tabu moves or tabu solutions, or indirectly by storing attributes of moves or attributes of solutions that are prohibited. If the minimization problem can be represented as a landscape limited to a territory which defines the feasible solutions and where altitude corresponds to the value of the objective function, the effect of this memory is to visit valleys (without always being at the bottom

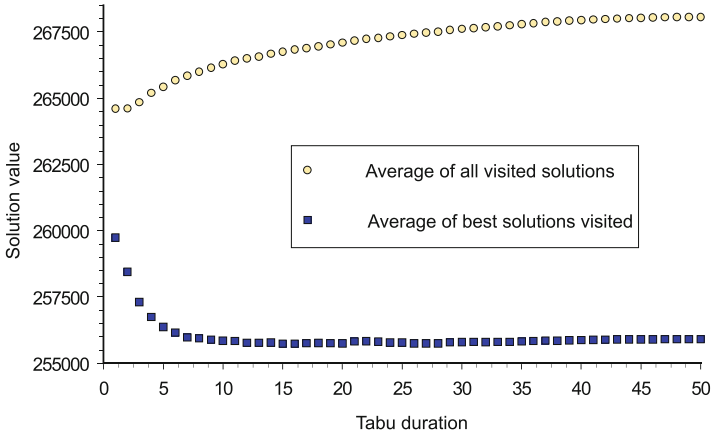

Fig. 3.8 Influence of the number of iterations during which moves are tabu

of the valley, because of tabu moves) and, sometimes, to cross a pass leading to another valley.

The higher the number of tabu moves is, the more likely one is to cross the mountains, but the less thoroughly the valleys will be visited. Conversely, if moves are prohibited for only a few iterations, there will be fewer chances of crossing the passes surrounding the valleys because, almost surely, there will be an allowed move which will lead to a solution close to the bottom of the valley; but, on the other hand, the bottom of the first valley visited will most probably be found.

More formally, for a very small number of tabu moves, the iterative search will tend to visit the same solutions over and over again. If this number is increased, the probability of remaining confined to a very limited number of solutions decreases and, consequently, the probability of visiting several good solutions increases. However, the number of tabu moves must not be very large, because it then becomes less probable that one will find good local optima, for lack of available moves. To some extent, the search is guided by the few allowed moves rather than by the objective function.

Figure 3.8 illustrates these phenomena in the case of the quadratic assignment problem: for each of 3000 instances of size 12, drawn at random, 50 iterations of a tabu search were performed. This figure gives the following two statistics as a function of the number of iterations during which a reverse move is prohibited: firstly, the average value of all the solutions visited during the search and, secondly, the average value of the best solutions found by each search. It should be noted that the first statistic grows with the number of prohibited moves, which means that the average quality of the visited solutions degrades. On the other hand, the quality of the best solutions found improves with an increase in the number of tabu moves, which establishes the fact that the search succeeds in escaping from comparatively poor local optima; then, their quality worsens, but this tendency is very limited here.

Thus, it can be concluded that the size of the tabu list must be chosen carefully, in accordance with the problem under consideration, the size of the neighborhood, the problem instance, the total number of iterations performed, etc. It is relatively easy to determine the order of magnitude that should be assigned to the number of tabu iterations, but the optimal value cannot be obtained without testing all possible values.

3.4.3.1 Random Tabu Duration

To obtain simultaneous benefits from the advantages of a small number of tabu moves—for through exploration of a valley—and a large number—for the ability to escape from the valley—the number of tabu moves can be modified during the search process. Several methodologies can be considered for this choice: for example, this number can be decided at random between a lower and an upper limit, in each iteration or after a certain number of iterations. These limits can often be easily identified; they can also be increased or decreased on the basis of characteristics observed during the search, etc. These were the various strategies employed upto the end of the 1980s [12, 13, 16]. These strategies were shown to be much more efficient than the use of tabu lists of fixed size (often implemented in the form of a circular list, although this may not be the best option, as can be seen in Sect. 3.5.2).

Again for the quadratic assignment problem, Fig. 3.9 gives the average number of iterations necessary for the solution of 500 examples of problems of size 15 generated at random, when the technique was to choose the number of tabu moves at random between a minimum value and that value increased by $Delta$. The size of the dark disks depends on the average number of iterations necessary to obtain optimal solutions for the 500 problems. An empty circle indicates that at least one of the problems was not solved optimally. The size of these circles is proportional to the number of problems for which it was possible to find the optimum. For $Delta = 0$, i.e., when the number of tabu moves is constant, cycles appear. On the other hand, the introduction of a positive $Delta$, even a very small one, can ensure much more protection against cycling. As can be noted in Fig. 3.8, the lower the tabu list size is, the smaller is the average number of iterations required to obtain the optimum. However, below a certain threshold, cycles appear, without passing through the optimum. From the point of view of robustness, one is thus constrained to choose sizes of tabu lists slightly larger than the optimal value (for this size of instances, it seems that this optimal value should be [7, 28] (minimum size $= 7$, $Delta = 21$), but it can be noticed that for [8, 28] a cycle appeared).

This technique of randomly selecting the number of tabu moves can thus guide the search automatically towards good solutions. However, such a mechanism could be described as myopic because it is guided mainly by the value of the objective function. Although it provides very encouraging results considering its simplicity, it

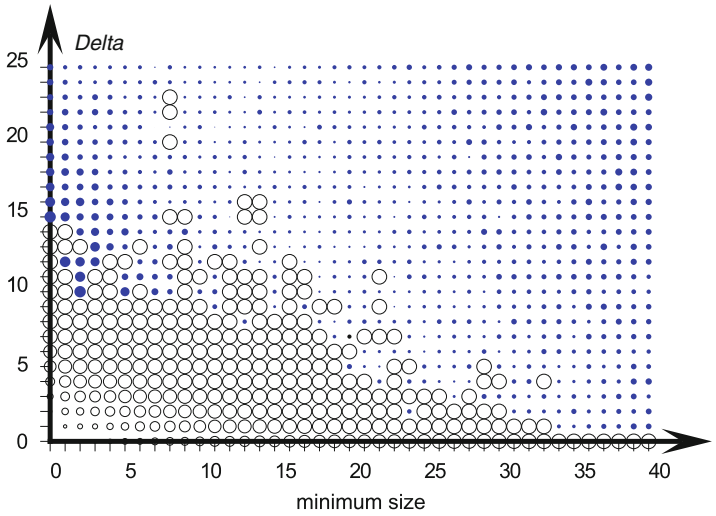

Fig. 3.9 The effect of random selection of the number of iterations during which moves are prohibited, for instances of the quadratic assignment problem of size 15 drawn at random. The number of iterations during which the reverse of a move was prohibited was drawn at random, uniformly between a minimum value and that value increased by $Delta$. The size of the filled disks grows with the average number of iterations necessary for the resolution of the problem until the optimum is found. An *empty circle* indicates that a cycling phenomenon appeared. The size of the circles is proportional to the number of problem instances solved optimally

cannot be considered as an intelligent way of guiding the search, but must rather be viewed as a basic tool for implementing the search process.

3.4.3.2 Type of Tabu List for the Quadratic Assignment Problem

A solution to the quadratic assignment problem can be represented in the form of a permutation **p** of n elements. A type of move very frequently used for this problem is to transpose the positions of two objects i and j. It is possible to evaluate effectively, in $O(n^2)$, the entire set of moves applicable to a solution.

As was discussed earlier, one technique for guiding the search in the short-term is to prohibit, for t iterations, the application of the reverse of the moves which have just been carried out. If a move (i, j) is applied to the permutation **p**, the reverse of the move can be defined as a move which simultaneously places the object i on the site p_i and the object j on the site p_j. There are other possible definitions of the reverse of a move, but this is one of the most effective ones for preventing cycles and appears to be the least sensitive one to the value of the parameter t, the number of iterations during which one avoids applying the reverse of a move. A fixed value of t does not produce a robust search, because the cycles may appear (see Fig. 3.9)

even for large values of t. To overcome this problem, it was proposed in [13] that t should be drawn uniformly at random, between $\lfloor 0, 9 \cdot n \rfloor$ and $\lceil 1, 1 \cdot n + 4 \rceil$. In fact, experiments have shown that a tabu duration equal to the size of the problem, or slightly larger for small examples, seems rather effective. This paved the way for the idea of selecting the value of t in a dynamic manner during the search, by choosing a maximum value slightly higher and an average value lower than the value which would have been ideal in the static case.

To implement this tabu mechanism in practice, a matrix $\mathcal{T}$ can be used whose entry t_{ir} gives the iteration number in which the element i was last moved from the site r (to go to the site p_i); this is the number to which one adds the tabu duration t. Thus, the move (i, j) is prohibited if both of the entries t_{ip_j} and t_{jp_i} contain values higher than the current iteration number.

Let us consider the small 5×5 instance of the quadratic assignment problem known in the literature as NUG5, with flow matrix $\mathcal{F}$ and distance matrix $\mathcal{D}$:

$$\mathcal{F} = \begin{pmatrix} 0 & 5 & 2 & 4 & 1 \\ 5 & 0 & 3 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 5 \\ 1 & 2 & 0 & 5 & 0 \end{pmatrix}, \quad \mathcal{D} = \begin{pmatrix} 0 & 1 & 1 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 2 & 1 & 0 \end{pmatrix}$$

With the tabu duration fixed at $t = 5$ iterations, the evaluation of the tabu search is the following.

Iteration 0. On the basis of the initial solution $\mathbf{p} = (5, 4, 3, 2, 1)$, meaning that the first element is placed in position 5, the second in position 4, etc., the value of this solution is 64. The search starts by initializing the matrix $\mathcal{T} = \mathbf{0}$.

Iteration 1. The value of $\Delta(\mathbf{p}, (i, j))$ is then calculated for each transposition m specified by the objects (i, j) exchanged:

m	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(2, 3)	(2, 4)	(2, 5)	(3, 4)	(3, 5)	(4, 5)
Cost	−4	−4	16	4	2	14	16	0	14	2

It can be seen that two moves can produce a maximum profit of 4, by exchanging either objects (1, 2) or objects (1, 3). We can assume that it is the first of these moves, (1, 2), which is retained, meaning that object 1 is placed in the position of object 2, i.e., 4, and object 2 is placed in the position of object 1, i.e., 5. It is forbidden for $t = 5$ iterations (i.e., up to iteration 6) to put element 1 in position 5 and element 2 in position 4 simultaneously. The following tabu condition matrix is obtained:

$$\mathcal{T} = \begin{pmatrix} 0\,0\,0\,0\,6 \\ 0\,0\,0\,6\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{pmatrix}$$

Iteration 2. The move chosen in iteration 1 leads to the solution $\mathbf{p} = (4, 5, 3, 2, 1)$, of cost 60. The computation of the value of every move for this new solution gives

m	$(1,2)$	$(1,3)$	$(1,4)$	$(1,5)$	$(2,3)$	$(2,4)$	$(2,5)$	$(3,4)$	$(3,5)$	$(4,5)$
Cost	4	10	22	12	−8	12	12	0	14	2
Tabu	Yes									

For this iteration, it should be noted that the reverse of the preceding move is now prohibited. The allowed move (2, 3), giving the minimum cost, is retained, for a profit of 8. The matrix $\mathcal{T}$ becomes

$$\mathcal{T} = \begin{pmatrix} 0\,0\,0\,0\,6 \\ 0\,0\,0\,6\,7 \\ 0\,0\,7\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{pmatrix}$$

Iteration 3. The solution $\mathbf{p} = (4, 3, 5, 2, 1)$, of cost 52, is reached, which is a local optimum. Indeed, at the beginning of iteration 3, no move has a negative cost:

m	$(1,2)$	$(1,3)$	$(1,4)$	$(1,5)$	$(2,3)$	$(2,4)$	$(2,5)$	$(3,4)$	$(3,5)$	$(4,5)$
Cost	8	14	22	8	8	0	24	20	10	10
Tabu					Yes					

The move (2, 4) selected in this iteration has zero cost. It should be noted here that the move (1, 2), which was prohibited in iteration 2, is again allowed, since the element 5 was never in the third position. The matrix $\mathcal{T}$ becomes

$$\mathcal{T} = \begin{pmatrix} 0\,0\,0\,0\,6 \\ 0\,0\,8\,6\,7 \\ 0\,0\,7\,0\,0 \\ 0\,8\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{pmatrix}$$

Iteration 4. The current solution is $\mathbf{p} = (4, 2, 5, 3, 1)$, of cost 52, and the data structure situation is as follows:

m	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(2, 3)	(2, 4)	(2, 5)	(3, 4)	(3, 5)	(4, 5)
Cost	8	14	22	8	8	0	24	20	10	10
Tabu						Yes				

However, it is not possible anymore to choose the move $(2, 4)$ corresponding to the minimum cost, which could bring us back to the preceding solution, because this move is prohibited. Hence we are forced to choose an unfavorable move, $(1, 2)$, that increases the cost of the solution by 8. The matrix $\mathcal{T}$ becomes

$$\mathcal{T} = \begin{pmatrix} 0 & 0 & 0 & 9 & 6 \\ 0 & 9 & 8 & 6 & 7 \\ 0 & 0 & 7 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Iteration 5. The solution at the beginning of this iteration is $\mathbf{p} = (2, 4, 5, 3, 1)$. The computation of the cost of the moves gives

m	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(2, 3)	(2, 4)	(2, 5)	(3, 4)	(3, 5)	(4, 5)
Cost	−8	4	0	12	10	14	12	20	10	−10
Tabu	Yes									

It can be noticed that the move degrading the quality of the solution in the preceding iteration was beneficial, because it now facilitates arriving at an optimal solution $\mathbf{p} = (2, 4, 5, 1, 3)$, of cost 50, by choosing the move $(4, 5)$.

3.4.4 Aspiration Conditions

Sometimes, some tabu conditions are absurd. For example, a move which leads to a solution better than all those visited by the search in the preceding iterations does not have any reason to be prohibited. In order not to miss this solution, it is necessary to disregard the possible tabu status of such moves. In tabu search terminology, this move is said to be *aspired*. Naturally, it is possible to assume other aspiration criteria, less directly related to the value of the objective to be optimized.

It should be noted here that the first presentations on tabu search insisted heavily on aspiration conditions, but, in practice, these were finally limited to allowing a tabu move which helped to improve the best solution found so far during the search. As this

later criterion became implicit, little research was carried out later on defining more elaborate aspiration conditions. On the other hand, aspiration can also sometimes be described as a form of long-term memory, consisting in forcing a move that has never been carried out over several iterations, irrespective of its influence on the objective function.

3.5 Long-Term Memory

In the case of a neighborhood defined by a static set of moves, i.e., when it does not depend on the solution found in the process, the statistics of the moves chosen during the search can be of great utility. If some moves are chosen much more frequently than others, one can suspect that the search is facing difficulties in exploring solutions of varied structure and that it may remain confined in a "valley." In practice, problem instances that include extended valleys are frequently observed. Thus, these can be visited using moves of small amplitude, considering the absolute difference in the objective function. If the only mechanism for guiding the search is to prohibit moves which are the reverse of those recently carried out, then a low number of tabu moves implies that it is almost impossible to escape from some valleys. A high number of tabu moves may force the search procedure to reside often on a hillside, but even if the search can change between valleys, it cannot succeed in finding good solutions in the new valley because numerous moves are prohibited after the visit to the preceding valley. It is thus necessary to introduce other mechanisms to guide a search effectively in the long term.

3.5.1 Frequency-Based Memory

One technique to ensure some diversity throughout the search without prohibiting too many moves, consists in penalizing moves that are frequently used. Several penalization methods can be imagined, for instance a prohibition of moves whose frequency of occurrence during the search exceeds a given threshold, or the addition of a value proportional to their frequency when evaluating moves. Moreover, the addition of a penalty proportional to the frequency has a beneficial effect for problems where the objective function takes only a small number of values, as that situation can generate awkward equivalences from the point of view of guiding the search when several neighboring solutions have same evaluation. In these situations, the search will then tend to choose those moves which are least employed rather than select a move more or less at random.

Figure 3.10 illustrates the effect of a method of penalization of moves which adds a factor proportional to their frequency of usage at the time of their evaluation. For this purpose, the experiment carried out to show the influence of the tabu duration (see Fig. 3.8) was repeated, but this time; the coefficient of penalization was varied;

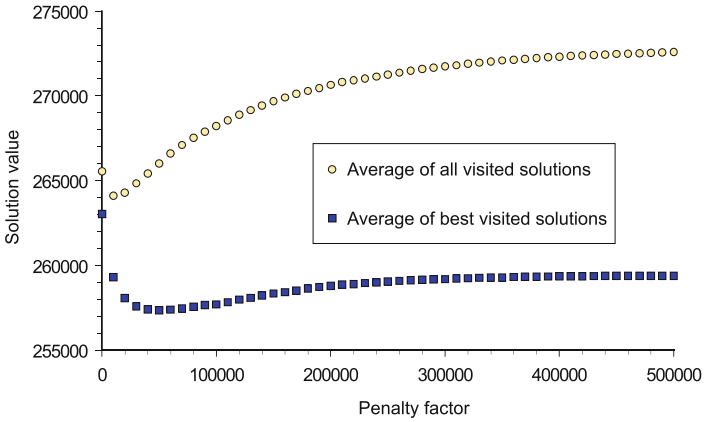

Fig. 3.10 Effect of a coefficient of penalization based on the frequencies of moves

the moves were thus penalized, but never tabu. This experiment relates again to the 3000 quadratic assignment instances of size 12 generated at random. In Fig. 3.10, the average of the best solutions found after 50 iterations and the average value of all the solutions visited are given as functions of the coefficient of penalization. It can be noticed that the behavior of these two statistics is almost the same as that shown in Fig. 3.8, but overall, the solutions generated are worse than those obtained by the use of a short-term memory.

Just like what has been done for short-term memory, this method can be generalized to provide a long-term memory of a nonstatic set of moves, i.e., where M depends on s: in this case the frequency with which certain characteristics of moves have been employeed to is recorded rather than the moves themselves. Here, the similarities in implementing these two forms of memory should be noticed: one method stores the iteration in which one can use a characteristic of a move again, whereas the other memorizes the number of times this characteristic has been used in the chosen moves.

3.5.1.1 Value of the Penalization

It is necessary to tune the value associated with a penalization method based on frequencies. This tuning can be carried out on the basis of the following considerations. Firstly, if $freq(m)$ denotes the frequency of usage of move m, it seems reasonable to penalize that move by a factor proportional to $freq(m)$, though other possible functions can be used, for instance $freq^2(m)$.

Secondly, if the objective function is linear and if a new problem instance is considered where all the data are multiplied by a constant, it is not desired that

a mechanism of penalization based on frequencies should depend on the value of the constant. In the same way, the mechanism of penalization should not work in a different manner if one adds a constant to the objective function. Consequently, it also seems legitimate to use a penalization which is proportional to the average amplitude of two neighboring solutions.

Thirdly, the larger the neighborhood is, the more the distribution of the frequencies becomes concentrated on small values. The penalty should thus be multiplied by a function that is strictly increasing with the size of the neighborhood, so that the penalty does not become zero when the size of the problem increases. If the identity function proves to be too large in practice (cf. [14, 15]), one can consider, for example, a factor proportional to $\sqrt{|M|}$.

Naturally, the concept of using penalization based on frequencies also requires taking the mechanism of aspiration into consideration. If not, then it is highly likely that we may miss excellent solutions.

3.5.2 Forced Moves

Another long-term mechanism consists in performing a move which has never been used during a large number of iterations, irrespective of its influence on the quality of the solution. Such a mechanism can be useful for destroying the structure of a local optimum, and therefore escaping from the valley in which it was confined. This is also valid for high-dimensional problems, as well as instances that have a more modest size but are very structured (i.e., for which the local optima are separated by very bad solutions).

In the earlier example of the quadratic assignment problem, it is not even necessary to introduce a new data structure to implement this mechanism. In fact, it is enough to implement the tabu list in the form of a matrix with two dimensions (element, position), whose entries indicate in which iteration each element is allowed to occupy a given position, either to decide if a move is prohibited (the entries in the matrix corresponding to the move contain values larger than the number of the current iteration) or, instead, to decide if a given element has not occupied a given position during the last v iterations. If the matrix contains an entry whose value is lower than the number of the current iteration decreased by the parameter v, the corresponding move is chosen, independent of its evaluation. It may happen that several moves could be simultaneously chosen because of this rule. This problem can be solved by imagining that, before the search was started, one had carried out all $|M|$ moves (a static, definite neighborhood of all M moves is assumed) during hypothetical iterations $-|M|, -|M|+1, \ldots, -1$. Of course, it is necessary that the parameter v be (sufficiently) larger than $|M|$, so that these moves are only imposed after v iterations.

3.6 Convergence of Tabu Search

Formally, one cannot speak about "convergence" for a tabu search, since in each iteration the solution is modified. On the other hand, it is definitely interesting to pass at least once through a global optimum. This was the focus of discussion in [9], on a theoretical level, using an elementary tabu search. It was shown that the search could be blocked if one prohibited passing through the same solution twice. Consequently, it is necessary to allow the same solution to be revisited. By considering a search which memorizes all the solutions visited and which chooses the oldest one that has been visited if all of the neighboring solutions have already been visited, it can be shown that all of the solutions of the problem will be enumerated. This is valid if the set of solutions is finite, and if the neighborhood is either reversible (or symmetric: any neighboring solution to s has s in its neighborhood) or strongly connected (there is a succession of moves that enables one to reach any solution s' starting from any solution s). Here, all of the solutions visited must be memorized (possibly in an implicit form), and it should be understood that this result remains theoretical.

Another theoretical result on the convergence of tabu search was presented in [2]. The authors of that study considered probabilistic tabu conditions. It is then possible to choose probabilities such that the search process is similar to that of simulated annealing. Starting from this observation, it can be expected that the convergence theorems for simulated annealing can easily be adapted to a process called *probabilistic tabu search*. Again, it should be understood that the interest of this result remains of a purely theoretical nature.

3.7 Conclusion

Only some of the basic concepts of tabu search have been presented in this chapter. Other principles may lead to a more effective and intelligent method. When possible, a graphical representation of the solutions visited successively during the search should be used, as it will actively stimulate the spirit of the designer and will suggest, often in an obvious way, how to guide the search more intelligently. The development of a tabu search is an iterative process: it is almost impossible to propose an excellent method at the first attempt. Adaptations, depending on the type of problem as well as on the problem instance considered, will certainly be required. This chapter has described only those principles which should enable a designer to proceed towards an effective algorithm more quickly. Other principles, often presented within the framework of tabu search suggested by F. Glover, such as scatter search, vocabulary building and path relinking, will be presented in Chap. 13, devoted to methodology.

3.8 Annotated Bibliography

Reference [6]	This book is undoubtedly the most important reference on tabu search. It describes the technique extensively, including certain extensions which will be discussed in this book in Chap. 13.
References [4, 5]	These two articles can be considered as the founders of the discipline, even if the name "tabu search" and certain ideas already existed previously. They are not easily accessible; hence certain concepts presented in these articles, such as path relinking and scatter search, were studied by the research community only several years after their publication.

References

1. Burkard, R.E., Fincke, U.: Probabilistic properties of some combinatorial optimization problems. Discrete Applied Mathematics **12**, 21–29 (1985)
2. Faigle, U., Kern, W.: Some convergence results for probabilistic tabu search. ORSA Journal on Computing **4**, 32–37 (1992)
3. Glover, F.: Future paths for integer programming and links to artificial intelligence. Computers and Operations Research **13**, 533–549 (1986)
4. Glover, F.: Tabu search—Part i. ORSA Journal on Computing **1**, 190–206 (1989)
5. Glover, F.: Tabu search—Part ii. ORSA Journal on Computing **2**, 4–32 (1990)
6. Glover, F., Laguna, M.: *Tabu Search*. Kluwer, Dordrecht (1997)
7. Glover, F., Laguna, M., Taillard, É.D., de Werra, D.: Annals of or 41. In: *Tabu Search*. Baltzer (1993)
8. Glover, F., Taillard, É.D., de Werra, D.: A user's guide to tabu search. Annals of Operations Research **41**, 1–28 (1993). http://mistic.heig-vd.ch/taillard/articles.dir/GloverTW1993.pdf. doi:10.1007/BF02078647
9. Hanafi, S.: On the convergence of tabu search. Journal of Heuristics **7**(1), 47–58 (2001)
10. Hertz, A., de Werra, D.: The tabu search metaheuristic: How we used it. Annals of Mathematics Artificial Intelligence **1**, 111–121 (1990)
11. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman. Operations Research **21**(2), 498–516 (1973)
12. Taillard, E.D.: Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research **47**(1), 65–74 (1990)
13. Taillard, E.D.: Robust taboo search for the quadratic assignment problem. Parallel Computing **17**, 443–455 (1991)
14. Taillard, E.D.: Parallel iterative search methods for vehicle routing problems. Networks **23**, 661–673 (1993)
15. Taillard, E.D.: Parallel taboo search techniques for the job shop scheduling problem. ORSA Journal on Computing **6**(2), 108–117 (1994)
16. Taillard, E.D.: Comparison of iterative searches for the quadratic assignment problem. Location Science **3**(2), 87–105 (1995)
17. Toth, P., Vigo, D.: The granular tabu search and its application to the vehicle-routing problem. INFORMS Journal on Computing **15**, 333–346 (2003)
18. Woodruff, D.L., Zemel, E.: Hashing vectors for tabu search. In: G. Glover, M. Laguna, E.D. Taillard, D. de Werra (eds.) *Tabu Search*, no. 41 in Annals of Operations Research, pp. 123–137. Baltzer, Basel (1993)