A TUTORIAL ON TABU SEARCH

Alain Hertz[1], Eric Taillard[2], Dominique de Werra[1]

[1] EPFL, Département de Mathématiques, MA-Ecublens, CH-1015 Lausanne
2 Université de Montréal, Centre de Recherche sur les Transports, Montréal, Canada H3C 3J7

Keywords:

1. Introduction

Engineering and technology have been continuously providing examples of difficult optimization problems. In this talk we shall present the tabu search technique which with its various ingredients may be viewed as an engineer designed approach: no clean proof of convergence is known but the technique has shown a remarkable efficiency on many problems.

The roots of tabu search go back to the 1970's; it was first presented in its present form by Glover [Glover, 1986]; the basic ideas have also been sketched by Hansen [Hansen 1986]. Additional efforts of formalization are reported in [Glover, 1989], [de Werra & Hertz, 1989], [Glover, 1990]. Many computational experiments have shown that tabu search has now become an established optimization technique which can compete with almost all known techniques and which - by its flexibility - can beat many classical procedures. Up to now, there is no formal explanation of this good behavior. Recently, theoretical aspects of tabu search have been investigated [Faigle & Kern, 1992], [Glover, 1992], [Fox, 1993].

A didactic presentation of tabu search and a series of applications have been collected in a recent book [Glover, Taillard, Laguna & de Werra, 1992]. Its interest lies in the fact that success with tabu search implies often that a serious effort of modeling be done from the beginning. The applications in [Glover, Taillard, Laguna & de Werra, 1992] provide many such examples together with a collection of references.

A huge collection of optimization techniques have been suggested by a crowd of researchers of different fields; an infinity of refinements have made these techniques work on specific types of applications. All these procedures are based on some common ideas and are furthermore characterized by a few additional specific features. Among the optimization procedures the iterative techniques play an important role: for most optimization problems no procedure is known in general to get directly an "optimal" solution.

The general step of an iterative procedure consists in constructing from a current solution i a next solution j and in checking whether one should stop there or perform another step. Neighbourhood search methods are iterative procedures in which a neighbourhood $N(i)$ is defined for each feasible solution i, and the next solution j is searched among the solutions in $N(i)$.

The most famous neighbourhood search method which has been used for finding an approximation to the minimum value of a real-valued function f on a set S is the descent method. It is outlined below :

Descent method

Step 1. Choose an initial solution i in S.

Step 2. Find a best j in $N(i)$ (i.e. such that $f(j) \leq f(k)$ for any k in $N(i)$).

Step 3. If $f(j) \geq f(i)$ then stop. Else set $i=j$ and go to Step 2.

Such a method clearly may stop at a local but not global minimum of f. In general, $N(i)$ is not defined explicitely: we may search for j by exploring some directions from i (for instance the coordinate axes).

Simulated annealing and tabu search can be considered as neighbourhood search methods which are more elaborate than the descent method. The basic ingredients of tabu search are described in the next section.

2. Basic ideas of Tabu Search

In order to improve the efficiency of the exploration process, one needs to keep track not only of local information (like the current value of the objective function) but also of some information related to the exploration process. This systematic use of *memory* is an essential feature of tabu search (TS). Its role will be emphasized later on. While most exploration methods keep in memory essentially the value $f(i^*)$ of the best solution i^* visited so far, TS will also keep information on the itinerary through the last solutions visited. Such information will be used to guide the move from i to the next solution j to be chosen in $N(i)$. The role of the memory will be to restrict the choice to some subset of $N(i)$ by forbidding for instance moves to some neighbour solutions.

More precisely, we will notice that the structure of the neighbourhood $N(i)$ of a solution i will in fact be variable from iteration to iteration. It would therefore be more appropriate to include TS in a class of procedures called *dynamic neighbourhood search techniques.*

Formally let us consider an optimization problem in the following way: given a set S of feasible solutions and a function $f : S \rightarrow \mathbb{R}$, find some solution i^* in S such that $f(i^*)$ is acceptable with respect to some criterion (or criteria). Generally a criterion of acceptability for a solution i^* would be to have $f(i^*) \leq f(i)$ for every i in S. In such a situation TS would be an exact minimization algorithm provided the exploration process would guarantee that after a finite number of steps such an i^* would be reached.

In most contexts however no guarantee can be given that such an i^* will be obtained; therefore TS could simply be viewed as an extremely general heuristic procedure. Since TS will in fact include in its own operating rules some heuristic techniques, it would be more appropriate to characterize TS as a *metaheuristic*. Its role will most often be to guide and to orient the search of another (more local) search procedure.

As a first step towards the description of TS, we reformulate the classical descent method as follows:

Step 1. Choose an initial solution i in S.

Step 2. Generate a subset V^* of solution in $N(i)$.

Step 3. Find a best j in V* (i.e. such that $f(j) \leq f(k)$ for any k in V^*) and set $i=j$.

Step 4. If $f(j) \geq f(i)$ then stop. Else go to Step 2.

In a straightforward descent method we would generally take $V^*=N(i)$. However this may often be too time-consuming; an appropriate choice of V^* may often be a substantial improvement.

The opposite case would be to take $|V^*|=1$; this would drop the phase of choice of a best j. Simulated annealing could be integrated within such a framework. A solution j would be accepted if $f(j) \leq f(i)$, otherwise it would be accepted with a certain probability depending upon the values of f at i and j as well as upon a parameter called temperature.

There is no temperature in TS; however the choice of V^* will be crucial; in order to define it at each step one will use systematically memory to exploit knowledge extending beyond the function f and the neighbourhood $N(i)$.

Except for some special cases of convexity, the use of descent procedures is generally frustrating since we are likely to be trapped in a local minimum which may be far (with respect to the value of f) from a global minimum.

So any iterative exploration process should in some instances accept also non-improving moves from i to j in V^* (i.e. with $f(j) > f(i)$) if one would like to escape from a local minimum. Simulated annealing does this also, but it does not guide the choice of j, TS in contrast chooses a best j in V^*.

As soon as non-improving moves are possible, the risk of visiting again a solution and more generally of cycling is present. This is the point where the use of memory is helpful to forbid moves which might lead to recently visited solutions. If such a memory is introduced we may consider that the structure of $N(i)$ will depend upon the itinerary and hence upon the iteration k; so we may refer to $N(i,k)$ instead of $N(i)$. With these modifications in mind we may attempt to formalize an improvement of the descent algorithm in a way which will bring it closer to the general TS procedure. It could be stated as follows (i^* is the best solution found so far and k the iteration counter):

Step 1. choose an initial solution i in S. Set $i^*=i$ and $k=0$.

Step 2. Set $k=k+1$ and generate a subset V^* of solution in $N(i,k)$

Step 3. Choose a best j in V^* (with respect to f or to some modified function $\tilde{f}$) and set $i=j$.

Step 4. If $f(i) < f(i^*)$ then set $i^*=i$.

Step 5. If a stopping condition is met then stop. Else go to Step 2.

Observe that the classical descent procedure is included in this formulation (the stopping rule would simply be $f(i) \geq f(i^*)$ and i^* would always be the last solution). Notice also that we may consider the use of a modified $\tilde{f}$ instead of f in some circumstances to be described later.

In TS some immediate stopping conditions could be the following:

- $N(i,k+1) = \emptyset$
- k is larger than the maximum number of iterations allowed

- the number of iterations since the last improvement of i* is larger than a specified number
- evidence can be given than an optimum solution has been obtained.

While these stopping rules may have some influence on the search procedure and on its results, it is important to realize that the definition of N(i,k) at each iteration k and the choice of V* are crucial.

The definition of N(i,k) implies that some recently visited solutions are removed from N(i); they are considered as tabu solutions which should be avoided in the next iteration. Such a memory based on recency will partially prevent cycling. For instance keeping at iteration k a list T (tabu list) of the last |T| solutions visited will prevent cycles of size at most |T|. In such a case we could take N(i,k)=N(i)-T. However this list T may be extremely impractical to use; we therefore will describe the exploration process in S in terms of moves from one solution to the next. For each solution i in S, we define M(i) as the set of moves which can be applied to i in order to obtain a new solution j (notation: $j=i\oplus m$). Then $N(i)=\{j \,/\, \exists\, m\in M(i) \text{ with } j=i\oplus m\}$. In general we use moves which are reversible: for each m there exists a move m^{-1} such that $(i\oplus m)\oplus m^{-1}=i$. So instead of keeping a list T of the last |T| solutions visited, we may simply keep track of the last |T| moves or of the last |T| reverse moves associated with the moves actually performed. It is clear that this restriction is a loss of information and that it does not guarantee that no cycle of length at most |T| will occur.

For efficiency purposes, it may be convenient to use several lists T_r at a time. Then some constituents t_r (of i or of m) will be given a tabu status to indicate that these constituents are currently not allowed to be involved in a move. Generally the tabu status of a move is a function of the tabu status of its constituents which may change at each iteration. So we may formulate a collection of tabu conditions as follows:

$$t_r(i,m) \in T_r \qquad (r=1,...,t).$$

A move *m* (applied to a solution *i*) will be a tabu move if *all* conditions are satisfied. Illustrations of these concepts will be given on different examples in the next sections.

Another drawback of the simplification of the tabu list (replacement of solutions by moves) is the fact that we may be lead to giving a tabu status to solutions which may be unvisited so far. We are then compelled to some relaxation of the tabu status; we will then overrule the tabu status when some tabu solutions will look attractive. This will be performed by means of *aspiration level conditions*.

A tabu move *m* applied to a current solution *i* may appear attractive because it gives for example a solution better than the best found so far. We would like to accept *m* in spite of its status; we shall do so if it has an *aspiration level* $a(i,m)$ which is better than a threshold value $A(i,m)$.

Generally $A(i,m)$ can be viewed as a set of preferred values for a function $a(i,m)$. So conditions of aspiration can be written in the form

$$a_r(i,m) \in A_r(i,m) \quad (r=1,...,a).$$

If at least one of these conditions is satisfied by the tabu move *m* applied to *i*, then *m* will be accepted (in spite of its status). Illustrations of these conditions will be given below.

We now have described almost all basic ingredients of TS. There is one additional feature which is important in the exploration process; it is related to the fact that in the process f may in some instances be replaced by another function $\tilde{f}$. This will allow us to introduce some intensification and diversification of the search.

In TS memory is used in different ways to guide the search procedure; we have seen a short term memory whose role was to forbid some moves likely to drive us back to recently visited solutions. Memory is also present at a deeper level.

In the search process it is sometimes fruitful to intensify the search in some region of S because it may contain some acceptable solutions. Such an intensification can be carried out by giving a high priority to the solutions which have common features with the current solution; this can be done with the introduction of an additional term in the objective function; this term will penalize solutions far from the present one. This should be done during a few iterations and after this it may be useful to explore another region of S; diversification will thus tend to spread the exploration effort over different regions of S. Again diversification can be forced by introducing an additional term in the objective function; this term will penalize at some stage solutions which are close to the present one. Both the intensification and diversification terms have weights which are modified in the process in such a way that intensification and diversification phases alternate during the search. The modified objective function is the function $\tilde{f}$ which was mentioned earlier in the algorithm: $\tilde{f}=f$+Intensification+Diversification.

We now have briefly presented the essential characteristics of a TS procedure which can be stated as follows :

Tabu search

Step 1. choose an initial solution i in S. Set i*=i and k=0.

Step 2. Set k=k+1 and generate a subset V* of solution in N(i,k) such that
either one of the tabu conditions $t_r(i,m) \in T_r$ is violated (r=1,...,t)
or at least one of the aspiration conditions $a_r(i,m) \in A_r(i,m)$ holds (r=1,...,a).

Step 3. Choose a best $j=i\oplus m$ in V* (with respect to f or to the function $\tilde{f}$) and set i=j.

Step 4. If f(i) < f(i*) then set i*=i.

Step 5. Update tabu and apsiration conditions.

Step 6. If a stopping condition is met then stop. Else go to Step 2.

Illustrations and a few refinements of tabu search will be provided in the next sections.

3. Efficiency of iterative solution methods

The efficiency of iterative solution methods depends mostly on the modelling. A fine tuning of parameters will never balance a bad choice of the neighbourhood structure or of the objective function. On the opposite, an effective modelling should lead to robust techniques that are not too sensitive to different parameter settings. In this section, we will give some general guidelines for designing efficient iterative solution methods.

3.1. Effective modelling

Iterative solution methods may be viewed as walks in a state space graph $G(S,A)$ where the vertex set S is the set of feasible solutions and there is an arc $(i,j) \in A$ from i to j if $j \in N(i)$. Choosing an initial solution is equivalent to generating a vertex in G and a step of the iterative procedure consists in moving from the current vertex i to a vertex adjacent to i. For a given optimization problem, there are usually many ways for defining the state space graph G, and it is essential to choose one that satisfies the following condition :

given any i in S, there should exist a path from i to an optimal solution i^*. (*)

If a solution that does not satisfy this condition is visited during the iterative process, then an optimal solution will never be reached.

To illustrate this fact, let us consider the graph coloring problem: given a graph H=(V,E) one has to find a coloring of its vertices with as few colors as possible such that no two vertices linked by an edge have the same color (such colorings are called feasible colorings). One could define the set S of feasible solutions as the set of feasible colorings which do not use more than a given number of colors; for example, this upper bound may be the number of colors used in the initial solution. Let us define the neighbourhood $N(i)$ of a solution i as the set of feasible colorings that can be obtained from i by changing the color of exactly one vertex. This induces a state space graph that does not satisfy condition (*). Indeed, consider the example in Figure 1: if the iterative solution method visits the feasible coloring c_1 and if the upper bound on the number of colors which may be used is 3, then an optimal coloring using only 2 colors will never be reached. A better definition of the state space graph G will be described in section 5.

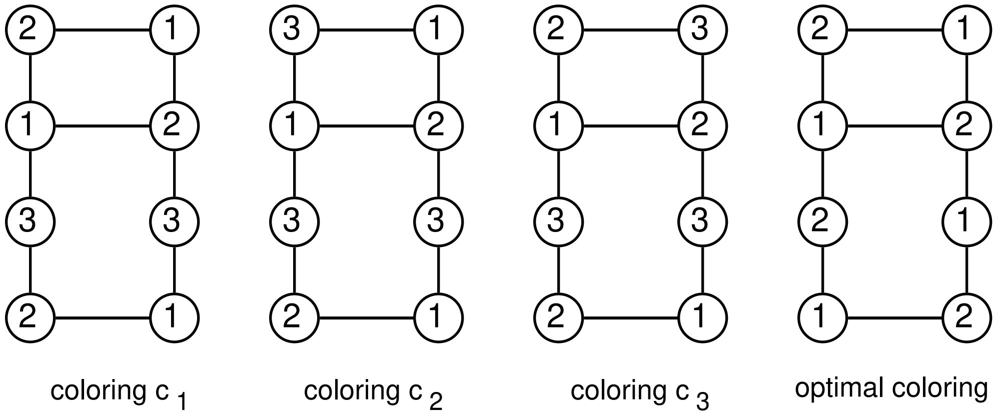

Figure 1

At this stage, it is important to remark that the set S of feasible solutions is not necessarily the set of solutions of the original problem: indeed, in some cases, it may be very difficult to define a neighbourhood structure that satisfies condition (*). If the solutions of the original problem must satisfy a set C of constraints, then it is sometimes judicious to define S as a set of solutions which satisfy a proper subset $C' \subset C$ of constraints. For example, for the graph coloring problem, an efficient adaptation of TS will be described in section 5 where the constraint that two vertices linked by an edge should receive different colors is relaxed. Of course, the violation of a relaxed constraint is penalized in the objective function.

Each vertex i of the state space graph G has a value $f(i)$. Considering these values as altitudes, this induces a topology with valleys containing local optima. It is important to define f in such a way that there are not too many valleys or large plateaus. Indeed in such a case, it is very difficult to guide the search towards an optimal solution, that is the bottom of one of the deepest valleys.

For the graph coloring problem, defining the value $f(i)$ of a coloring i as the number of colors which are used in i induces very large plateaus since all solutions have an integer value which belongs to the small interval $[f(i^*), u]$, where $f(i^*)$ is the value of an optimal solution an u is the upper bound.

As mentioned in section 2, the objective function f may be changed during the search. These changes correspond to a modification of the topology. During the diversification phase, for example, it is tried to replace the valleys often visited by high mountains; hence the search is directed towards unexplored regions.

3.2. Effective computing

At each step of an iterative solution method, many solutions have to be evaluated and it is important to perform this computation in an efficient way.

For a solution i and a move m, it is often easier to compute $\Delta(i,m)$ defined as $f(i\oplus m)-f(i)$ than $f(i\oplus m)$. In some cases, it is possible to store the values $\Delta(i,m)$ for all $m\in M(i)$. In such a case, if a move m may be applied to two consecutive solutions i and j, ($m\in M(i)\cap M(j)$), then it happens frequently that $\Delta(j,m)$ may be computed even more easily. For some problems, given a solution i and a move $m\in M(i)$, it is difficult to compute $f(i\oplus m)$. For example, for the vehicle routing problem (VRP), let us define a solution i as the assignment of the customers to the vehicles, and a move m as the transfer of exactly one customer from one vehicle to another one. If the objective is to minimize the total distance travelled, then computing $f(i\oplus m)$ is equivalent to solving several travelling salesman problems (TSP). As well known, the TSP is an NP-hard problem.

In such cases it is necessary to use fast heuristic methods for evaluating the neighbour solutions. Once a best neighbour solution has been determined, a more elaborate technique may be used to improve it. For example, for the VRP, a simple insertion method may be used for finding the best neighbour $j\in N(i)$. All routes of i which have been modified for getting j are then re-optimized using improvement methods such as r-opt [Lin & Kernighan, 1973] or even exact methods when there are not too many customers in the routes.

In the case where all the values $\Delta(i,m)\ \forall m\in M(i)$ can be stored, it is sometimes possible to keep a sorted list of these values with a small computational effort.

In section 5 we will describe an adaptation of TS to the problem of finding an independent set of given size k in a graph with n vertices. The $O(nk)$ neighbours will be efficiently evaluated by storing two sorted lists containing pertinent information. One step of TS (that is finding the best neighbour and updating the sorted lists) can then be performed in time $O(n)$.

4. Efficient use of memory

In section 2 a general description of the TS procedure has been presented and we have seen that the use of memory is an essential feature of TS. The tabu conditions may usually be considered as a short term memory which prevents cycling at some extent. We describe in this section some efficient policies for the management of the tabu lists.

We shall also show how the use of memory may help to intensify the search in "good" regions or to diversify the search towards unexplored regions.

4.1. Variable tabu list size

We have seen that it may be convenient to use several tabu lists at a time. We will restrict the discussion to the case of a unique tabu list but the following may be extended to the general case.

The basic role of the tabu list is to prevent cycling. If the length of the list is too small, this role might not be achieved; conversely a too long size creates too many restrictions and it has been observed that the mean value of the visited solutions grows with the increase of the tabu list size. Usually, an order of magnitude of this size may be easily determined. However, given an optimization problem it is often difficult or even impossible to find a value that prevents cycling and does not excessively restrict the search for all instances of the problem of a given size.

An effective way for circumventing this difficulty is to use a tabu list with variable size. Each element of the list belongs to it for a number of iterations that is bounded by given maximal and minimal values.

4.2. Intensification of the search

In order to intensify the search in promising regions, we first have to come back to one of the best solution found so far. Then the size of the tabu list may be simply decreased for a "small" number of iterations.

In some cases, more elaborate techniques may be used. Some optimization problems can be partitioned into subproblems. Solving these subproblems optimally and combining the partial solutions leads to an optimal solution. The difficulty with such a strategy obviously consists in finding a good partition.

As mentioned before, for reasons related to computational time, fast heuristics and a neighbourhood of reasonable size are used at each step of TS. Ways for intensifying the search are the use of more elaborate heuristics or even exact methods or the enlargement of the neighbourhood.

It is also possible to perform an intensification based on a long term memory. Each solution or move can be characterized by a set of components. The components of "good" moves or "good" solutions are memorized. During the intensification phase the moves or the solutions are evaluated taking into account their amount of "good" components. This long term memory may be viewed as a kind of learning process.

4.3. Diversification

In order to avoid that a large region of the state space graph remains completely unexplored it is important to diversify the search. The simplest way to do it is to perform several random restarts. A different way which guaranties the exploration of unvisited regions is to penalize frequently performed moves or solutions often visited.

This penalty is set large enough to ensure the escape from the current region. The modified objective function is used for a given number of iterations. It is also possible to use a penalty on frequently performed moves during the whole search procedure.

In the case where the feasible solutions have to satisfy a set of constraints, these appear as unpassable mountains (of infinite height). Relaxing these constraints and penalizing their violation corresponds to reducing the height of the mountains. It is then possible to pass this barrier and reach another valley in few steps. During this diversification phase, the solutions visited are not necessarily feasible (since constraints have been relaxed). For obtaining a feasible solution again, the penalty for the violation of the relaxed constraints is gradually increased.

5. Some applications of TS

TS has been applied to many combinatorial optimization problems. The aim of this chapter is to describe some of these applications in order to illustrate the ingredients of TS and to give some practical ways of implementing them. More details on these applications and references to other works may be found in: [Hertz & de Werra, 1990], [Glover, Taillard, Laguna & de Werra, 1992].

5.1. The graph coloring problem [Hertz & de Werra, 1987]

Given a graph $G=(V,E)$ we have to find a coloring of its vertices with as few colors as possible. The unique constraint is that two vertices linked by an edge should not receive the same color.

In other words, a coloring of the vertices of G in k colors is a partition of the vertex set V into k independent sets $V_1,...,V_k$. For defining the set S of feasible solutions, the unique constraint has been relaxed and TS is used to find if possible a coloring in a given number k of colors. In other words a feasible solution is any partition $(V_1,...,V_k)$ of V into k sets. By defining $E(V_r)$ as the set of edges having both endpoints in V_r, the objective is to minimize $\sum_{r=1}^{k}|E(V_r)|$.

Hence a solution of value 0 corresponds to a coloring of G in k colors. As already mentioned, defining S as the set of all partitions of V into independent sets (that is the unique constraint of the problem is not relaxed) and the objective function as the number of colors used implies that a large number of moves (that change the color of a unique vertex) have the same value (and very few are improving ones). It is difficult to guide the search in such a situation.

In the efficient modelling of TS described above the neighbourhood $N(i)$ of a solution i consists of all solutions generated from i by the following local modification : move a vertex which is endpoint of a monochromatic edge (both endpoints are in the same V_r) to a set V_q ($q \neq r$). When a vertex v is moved from V_r to V_q, the pair (v,r) is introduced in the tabu list T. This means that it is forbidden to introduce v in V_r during $|T|$ iterations. For more details, the reader is referred to [Hertz & de Werra, 1987].

5.2. The Maximum Independent set problem [Friden, Hertz & de Werra, 1989], [Friden, Hertz & de Werra, 1990].

Given a graph $G=(V,E)$ we have to determine a subset X of V as large as possible such that $E(X)=\emptyset$ (where $E(X)$ is a set of edges having both endpoints in X).

TS has been adapted for trying to determine in G an independent set of a given size k. Once again, the unique constraint of the problem has been relaxed. A solution is any subset X of V of size k and the objective function is $|E(X)|$. The neighbourhood consists in exchanging a vertex $v \in X$ with a vertex $w \in V\text{-}X$. The size of the neighbourhood of a solution is equal to $k(n\text{-}k) \in O(nk)$ where $n=|V|$.

For a vertex $v \in V$, let us denote $\Gamma_X(v)$ the set of vertices w in X that are linked to v by an edge. The vertices v in X are sorted according to non increasing values of $\Gamma_X(v)$ and the vertices $w \in V\text{-}X$ are sorted according to non decreasing values of $\Gamma_X(w)$. Keeping these two lists sorted requires time in $O(n)$. Three tabu lists are used. The first one T_1 stores the last visited solutions, and the second one T_2 (resp. the third one T_3) contains the last vertices introduced into X (resp. removed from X). As shown in [Friden, Hertz & de Werra, 1989], by using hashing techniques for implementing T_1 and choosing a small constant value for $|T_2|$ and $|T_3|$, a best neighbour may be found in practice almost in constant time. Hence one step of TS takes time in $O(n)$ instead of $O(nk)$.

A completely different adaptation of TS to the maximum independent set problem has been described in [Friden, Hertz & de Werra, 1990]. TS is used for getting bounds in a branch & bound algorithm. At each node of the enumerative algorithm, let us define two sets I and O of vertices that are forced to be respectively inside and outside of a maximum independent set. Let X be the largest independent set found so far. TS is used for covering with $|X|\text{-}|I|$ cliques as many vertices as possible in the graph H induced by $U=V\text{-}\{I \cup O\}$. Let W be the set of vertices covered by the cliques. If $|W|=|U|$ then backtracking occurs since H cannot contain a stable set of size larger than $|X|\text{-}|I|$. Otherwise, we generate a new subproblem for each vertex $v \in U\text{-}W$ by introducing v in I and by adding in O all vertices adjacent to v.

Using a less elaborate technique for covering the vertices of H by cliques requires less time, but the number of branches generated at each node of the branch & bound algorithm is generally much larger. It has been shown that it is worth spending time using TS at each node for reducing the number of branchings. More details can be found in [Friden, Hertz & de Werra, 1990].

5.3. The Course scheduling problems [Hertz, 1991], [Hertz 1992], [Costa, 1994].

Basically, course scheduling problems can be formulated as graph coloring problems: the courses are the vertices of a graph, the periods correspond to the colors and two vertices are linked by an edge if the corresponding courses cannot be given at the same time. In real life problems many additional constraints have to be taken into account and most optimization techniques can hardly deal with all the specific requirements of each school.

TS has been successfully applied to this problem by extending its adaptation to the graph coloring problem. A feasible solution is defined as a schedule that satisfies a subset C of constraints and a neighbour solution is obtained by modifying the schedule of one course. If all courses last one period, this is equivalent to change the color of a vertex. In order to decide whether a specific constraint should be included in C, the following guidelines can be used: a course should never be assigned to a period if such an assignment induces only course schedules that do not satisfy all the constraints. Moreover a conflict between two courses should not be a sufficient condition for hindering an assignment.

For example, let us consider two courses c_1 and c_2 involving common students. If c_1 can not be scheduled at period p for some reason (the teacher is perhaps not available at that time, for example), then we shall never visit a solution where c_1 is given at time p. However, if this constraint does not exist, and c_2 is scheduled at time p, then we are allowed to schedule c_1 at time p, even if this violates the constraint of no overlapping of courses involving common students. The reason is that c_1 is possibly given at time p in an optimal schedule and the violation of the constraint can be cancelled by moving course c_2 to another period. As mentioned in [Hertz, 1992], such a model can handle timetabling problems with option courses, time windows and pre-assignment requirements, compactness, geographical and precedence constraints, and the courses can have different durations.

6. Conclusions

The above examples have shown how wide the range of applications of Tabu Search is; more descriptions of such cases can be found in [Glover, Taillard, Laguna & de Werra, 1992].

Although basically simple the technique owes its efficiency to a rather fine tuning of an apparently large collection of parameters. In fact experiments have shown that the degrees of freedom in most of these choices are not so restricted. Theoretical considerations based on a probabilistic model of Tabu Search seem to confirm partially this statement [Faigle & Kern, 1992]: when moves from a solution i to a solution j are performed according to a probability distribution p_{ij}, then under rather mild assumptions one can prove that convergence is obtained to an optimal solution for probabilities p_{ij} which are allowed to be chosen in a large subinterval of [0,1]. This approach may lead to a better understanding of the efficiency of Tabu Search.

For the moment, we may just recognize that this technique looks more like an engineering approach to difficult and large size problems of optimization than like an elegant simple mathematical algorithm.

With tabu search, complexity is not only present in the problems but in the technique itself. It is clearly not an advantage, but this situation may be viewed as an exploratory step in a new field of techniques which will likely become more elegant and hopefully more simple when memory, artificial intelligence and exploration procedures will be better integrated in a technique which will probably no longer bear the strange name of tabu.

References

[1] Battiti R., Tecchiolli G. -*The reactive tabu search.*- preprint, Department of Mathematics, University of Trento, Trento, Italy (1992)

[2| Costa D. -*A Tabu Search Algorithm for Computing an Operational Time Table* - European Journal of Operational Research 76, (1994), pp. 98-110.

[3] Faigle U., Kern W. -*Some Convergence Results for Probabilistic Tabu Search* - ORSA Journal on Computing 4, (1992), pp. 32-37.

[4] Fox B.L. -*Integrating and accelerating tabu search, simulated annealing and genetic algorithms* - Annals of Operations Research 41, (1993) pp. 47-67.

[5] Friden C. , Hertz A. , de Werra D. -*STABULUS: a technique for finding stable sets in large graphs with tabu search* - Computing 42, (1989) pp. 35-44.

[6] Friden C. , Hertz A. , de Werra D. -*TABARIS: an exact algorithm based on tabu search for finding a maximum independent set in a graph* - Computers and Operations Research 17, (1990), pp. 437-445.

[7] Gendreau M., Hertz A., Laporte G. -*A Tabu Search Heuristic for the Vehicle Routing Problem*- Management Science 40/10, (1994), pp. 1276-1290.

[8] Glover F. -*Future Paths for Integer Programming and Links to Artificial Intelligence*- Computers and Operations Research 13, (1986), pp. 533-549.

[9] Glover F. -*Tabu Search, Part I* - ORSA Journal on Computing 1, (1989), pp. 190-206.

[10] Glover F. -*Tabu Search, Part II* - ORSA Journal on Computing 2, (1990), pp. 4-32.

[11] Glover F. -*Private communication* - (1992)

[10] Glover F., Taillard E., Laguna M., de Werra D. -*Tabu Search* -Volume 41 of the Annals of Operations Research, (1993)

[12] Hansen P. -*The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming* - Presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy, (1986)

[13] Hertz A. -*Tabu Search for Large Scale Timetabling Problems* - European Journal of Operational Research 54/1, (1991), pp. 39-47.

[14] Hertz A. -*Finding a Feasible Course Schedule Using Tabu Search* - Discrete Applied Mathematics 35, (1992), pp. 255-270.

[15] Hertz A., de Werra D. -*Using Tabu Search Techniques for Graph Coloring* - Computing 39, (1987), pp. 345-351.

[16] Hertz A., de Werra D. -*The Tabu Search Metaheuristic: how we used it* - Annals of Mathematics and Artificial Intelligence 1, (1990), pp. 111-121.

[17] Lin S. , Kernighan B.W. -*An Effective Heuristic Algorithm for the Traveling-Salesman Problem* - Operations Research 21, (1973), pp. 498-516.

[18] Osman I.H. -*Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem* - Annals of Operations Research. 41, (1993), pp. 421-451

[19] Semet F., Taillard E. -*Solving real-life vehicle routing problems efficiently using tabu search* - Annals of Operations Research 41, (1993), pp. 469-488.

[20] Taillard E. -*Robust Taboo Search for the Quadratic Assignment Problem* - Parallel Computing 17, (1991), pp. 443-455.

[21] Taillard E. -*Parallel Iterative Search Methods for Vehicle Routing Problems* - Network 23, (1993), pp. 661-673.

[22] de Werra D., Hertz A. -*Tabu Search Techniques: A tutorial and an application to neural networks* - OR Spektrum, (1989), pp. 131-141.