Belgian Journal of Operations Research, Statistics and Computer Science Vol. 32 n° 1-2

# General local search heuristics in Combinatorial Optimization : a tutorial

M. Pirlot

Faculté Polytechnique de Mons
rue de Houdain 9
B-7000 Mons - Belgium

**Abstract**

The paper presents four general heuristic search strategies that can in principle be adapted to any combinatorial optimization problem. All these techniques, Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithms (GA) and Neural Networks (NN), can be described as local search heuristics. We provide an elementary description of each of them together with examples of applications and a bibliographic and historic note. More advanced developments in the framework of each strategy are outlined.

Keywords : Combinatorial optimization, heuristic search, local search, simulated annealing, tabu search, genetic algorithms, neural networks

# 1 Introduction

## 1.1 Four general heuristics in combinatorial optimization

In recent years, much attention and many papers have been devoted to four *general* heuristics that are applicable in particular for "solving" combinatorial optimization problems. These general heuristics also called *metaheuristics*, are Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithms (GA) and Neural Networks (NN). We shall first situate the above methods in the general panoply of tools that can be used to find satisfactory solutions to combinatorial optimization problems. Doing this, we shall adopt the practitioner's viewpoint.

There is a huge amount of combinatorial optimization problems in all sectors of management. A small sample of them is: sequencing and scheduling problems in production management, routing and transportation, plant location, time-tabling, ..., and so on. Very often, the manager is not aware of or not convinced by the possible benefits of using discrete optimization methods in industry. Quite often too, he has more important tasks or the above-mentioned problems appear as detail and the possible benefits as "peanuts". However, times are changing and the international competition forces economic actors to get interested in such details.

Consider now a production manager at a relatively low level who is in charge of saving "peanuts". Suppose he is conscious of the possible benefits and wanting to use some of the existing methods for solving his combinatorial optimization problems. Very often, this good-willing person will be discouraged by one or several of the following problems (the list is not exhaustive): the effort in time and mathematical education for finding and understanding a suitable method may be judged not worth the possible return, there may be no method which is adapted to the complex constraints of the real situation, the methods may be inapplicable in practice due to inappropriate computing time or deceptive solution quality or lack of robustness.

As a consequence, quite frequently one will not try to use any optimization method but simply implement some simple rule (of good practice, rule of thumb, ...) in a program that will deal with the problem. In the most sophisticated and/or expensive solutions this program will receive the name of expert system. In such a context, the interesting point with the general heuristic strategies mentioned above is that they generally yield relatively good results for a not too important effort (in computation time, understanding, implementation, ...). In particular, they are often flexible enough to be able to deal with complex particular constraints.

These considerations seem a sufficient motivation for examining such methods. On the other hand, one should also refrain from excessive enthusiasm. These methods are new and appealing. They rely on analogies with optimization mechanisms in "Nature": physical systems able to evolve towards a state of minimal energy (simulated annealing and neural networks), incest prohibition (tabu search), natural selection and species evolution (GA). Some of these methods (SA, GA, NN) are supported by theoretical results which guarantee their good behaviour. Hence, it is no wonder that each method has eager supporters who consider that their preferred method cannot be beaten. This is certainly not a reasonable attitude because analogy cannot be considered a justification ("comparaison n'est pas raison"), the theoretical results are not in general relevant for the actual use of the methods in practice (more detail on that below) and one can (easily) find problems on which any of our four heuristics is not competitive. In conclusion, one should not consider the four general heuristics as a substitute to exact methods, or other types of heuristics: there are cases where it is not advisable to use them because there are much better specific or general approaches. They are "not a panacea but are potentially useful tools" (paraphrasing Johnson *et al.* 1991, p. 405).

From a theoretical viewpoint also these methods are interesting: the existing theoretical results though not relevant for practice suggest that trying to understand why these methods work or fail is not hopeless.

## 1.2 Organization of the paper

The four heuristic search strategies are exposed in Sections 2, 4, 6 and 8 in a not too technical language. Examples of applications are provided for each method as well as a bibliographic and historic note which aims at orienting the interested reader in the literature. The reader who only wants an overview of the subject can skip Sections 3, 5 and 7 where more advanced topics are dealt with as well as more technical issues are addressed. The ambition with this paper is to provide the reader with a good starting point for applying one or several of these methods in practice.

## 1.3 Local search strategies

All four general heuristics we are concerned with can be considered to some extent as local search strategies: it is quite clear for SA and TS, somewhat less for GA and NN. Essentially, *local search* consists in moving from a solution to another one in the neighborhood according to some definite rules. The sequence of solutions can be called a *trajectory* in the solution space.

For definiteness, let us consider the problem of minimizing a function $F(x)$ on a finite set of points $X$. This can be considered a general statement of a combinatorial optimization problem. A local search strategy starts from an arbitrary solution $x_1 \in X$ and at each step $n$, a new solution $x_{n+1}$ is chosen in the neighborhood $V(x_n)$ of the current solution $x_n$. This implies the definition of a *neighborhood* structure on $X$: to each $x \in X$ is associated a subset $V(x) \subseteq X$ called the neighborhood of $x$. For instance, if $X$ is a set of binary vectors and $\overline{x} \in X$, the neighborhood $V(\overline{x})$ of $\overline{x}$ may be the set of all solutions $x \in X$ that can be obtained from $\overline{x}$ by flipping a single coordinate from 0 to 1 or conversely. By convention, we suppose that no solution is a neighbour of itself, i. e. $x \notin V(x)$, $\forall x \in X$. Alternatively, one can say that the neighbours of $x$ are the solutions that can be obtained from $x$ by an *elementary move*. The evolution of the current solution $x_n$, $n = 1, 2, \ldots$ draws a *trajectory* in $X$.

The most common way of choosing a new solution $x_{n+1}$ in the neighborhood of $x_n$ is to pick (one of) the best one(s), i. e. a solution $x_{n+1} \in V(x_n)$ with $F(x_{n+1}) \leq F(x)$, $\forall x \in V(x_n)$. Then, $x_{n+1}$ becomes the next current solution if it is not worse than $x_n$, i. e. $F(x_{n+1}) \leq F(x_n)$. Otherwise, the search is stopped. This strategy is usually called a *descent* or a *steepest descent* strategy. For further comparison purposes we give a formal description below. $F_n^*$ will denote the best value of $F$ up to step $n$ and $x_n^*$ is such that $F(x_n^*) = F_n^*$.

**Descent Algorithm**

- Initialization: select $x_1 \in X$.
- Step $n = 1, 2, \ldots$: $x_n$ denotes the current solution.
  a) Find the best $\overline{x}$ in the neighborhood $V(x_n)$.
  b) If $F(\overline{x}) \leq F(x_n)$ then $\overline{x}$ becomes the new current solution $x_{n+1}$ at step $n+1$ and the best value $F_n^*$ of $F$ (up to step $n$) as well as $x_n^*$ are updated.
  c) Else: stop.

Note that the choice of a good neighborhood structure is generally important for the effectiveness of the process. The main weakness of the descent algorithm is its unability to escape from *local minima*. This is symbolically illustrated in Figure 1: all solutions in the neighborhood $V(x_n)$ are worse than $x_n$ although, further away, there exists a global minimum of $F$ which cannot be reached under the descent rule.

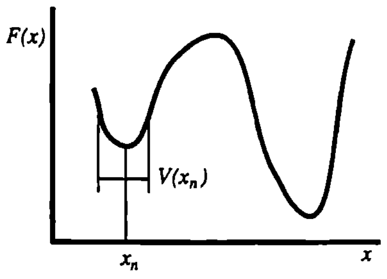

Figure 1: Trapped in a local minimum.

Simulated Annealing and Tabu Search are local search strategies explicitely designed for avoiding such a situation. This implies temporary degradation of the objective function.

# 2 Simulated Annealing (SA)

## 2.1 General presentation of Simulated Annealing

When using Simulated Annealing (SA) algorithms one does not search for the best solution in the neighborhood $V(x_n)$ of the current solution $x_n$: one simply draws at random a solution $x$ in $V(x_n)$. If $F(x) \leq F(x_n)$, $x$ becomes the new current solution. Otherwise, one of the two following alternatives is selected according to some probabilistic law: $x$ becomes the current solution with probability $p(n)$ or $x_n$ remains the current solution with the complementary probability $1 - p(n)$. Typically, $p(n)$ decreases with time ($n$) and with the size of the deterioration of $F$ ($= F(x) - F(x_n)$).

The idea of SA comes from thermodynamics and metallurgy: when a metal in fusion is cooled slowly enough it tends to solidify in a structure of minimal energy. The same principle is at work in SA: at the beginning, almost all moves (i. e. all updating of the current solution by a solution $x$ randomly chosen in its neighborhood) are accepted. This allows to "explore" the solution space. Then, gradually, "temperature" is decreased which means that one becomes more and more selective in accepting new solutions. At the end, only the moves that improve $F$ are accepted in practice. Schematically, SA is the following alteration of the Descent Algorithm.

**Simulated Annealing (SA)**

- Initialization: Select an initial solution $x_1$ in $X$

$$\begin{aligned} F^* &\leftarrow F(x_1) \\ x^* &\leftarrow x_1 \end{aligned}$$

- Step $n = 1, 2, \ldots$: $x_n$ denotes the current solution.
  - Draw $x$ at random in the neighborhood $V(x_n)$ of $x_n$.
  - If $F(x) \leq F(x_n)$ then $x_{n+1} \leftarrow x$.

    If $F(x) < F^*$ then $F^* \leftarrow F(x)$ and $x^* \leftarrow x$.
  - Else, draw a number $p$ at random in $[0, 1]$.

If $p \leq p(n)$ then $x_{n+1} \leftarrow x$.

- End: If stopping condition is fulfilled then stop.

In order to make the algorithm operational, a few tactical decisions have to be made.

a) Choice of the acceptance probability $p(n)$.

   By analogy with thermodynamics, a Boltzman-like distribution is generally chosen:

$$p(n) = \exp(-\frac{1}{T(n)}\Delta F_n)$$

   where $\Delta F_n = F(x) - F(x_n)$ and $T(n)$ is the so-called "temperature" at step $n$.

b) Choice of a "*temperature schedule*".

   Temperature $T(n)$ decreases with time in order to exclude or almost exclude "bad moves" at the end. A classical schedule for decreasing $T$ is illustrated in Figure 2. Starting from $T_0$, the temperature is maintained

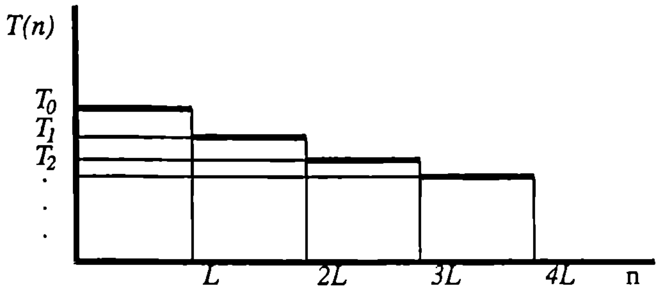

Figure 2: The most common temperature schedule.

   constant for $L$ consecutive steps. Then it is decreased by multiplying it by a constant factor $\alpha$ $(0 < \alpha < 1)$ after each series of $L$ steps. Hence, after $kL$ steps, the temperature is

$$T(kL) = T_k = \alpha^k T_0$$

This implies the setting of three parameters, $T_0$, $\alpha$ and $L$, which will be respectively referred to as *initial temperature*, *cooling rate* and *length of plateau.*

c) Choice of a stopping rule.

Here are two variants of a stopping rule that are both natural and commonly used:

- Stop 1: If $F^*$ was not improved by at least $\epsilon_1$ % after $K_1$ consecutive series of $L$ steps, the procedure is halted.
- Stop 2: If the number of accepted moves is less than $\epsilon_2$% of $L$ for $K_2$ consecutive series of $L$ steps, the procedure is halted.

For the sake of completeness, let us recall that the definition of the neighborhood structure is an important decision in this local search algorithm. The influence of a particular parameter choice will be discussed in Section 3.2.

## 2.2 Bibliographic and historic note

The idea of applying the *annealing* principle to optimization problems is due to Kirkpatrick, Gelatt and Vecchi (1983) and Cerny (1985) who worked independently. They both applied SA to the Travelling Salesman Problem. Since then, a huge amount of literature has been devoted to reporting about applications of the method in many domains and to presenting variants and enhancements of the basic technique. Extensive bibliographies can be found in Collins *et al.* (1988) and van Laarhoven and Aarts (1987). Several books have also been devoted to SA (see e. g. van Laarhoven and Aarts 1987, Azencott 1992, Vidal 1993, and Siarry and Dreyfus 1989). Special mention is due to a series of papers by Johnson and other authors (Johnson *et al.* 1989, 1991, 1992) where a very careful and methodologically sound job is done in comparing SA with other heuristics on four classical problems: graph partitioning, graph coloring, number partitioning and the travelling salesman problem. We come back to this work in the next section.

## 2.3 An example: graph partitioning

We present the graph partitioning problem investigated in Johnson *et al.* (1989): partition the vertices $V$ of the graph $G = (V, E)$ into two equal size subsets $V_1$ and $V_2$ in such a way that the number of edges with endpoints in both subsets be minimal.

There are different manners of applying SA to this problem. The most natural is the following. The solution space $X$ is the set of bipartitions $(V_1, V_2)$ of $V$ with $|V_1| = |V_2| = |V|/2$. The objective function (to minimize) is the number of edges with one endpoint in $V_1$ and one endpoint in $V_2$. The neighborhood of the current solution $(V_1, V_2)$ is the set of all bipartitions that can be obtained by exchanging a vertex from $V_1$ with a vertex from $V_2$: $V_1' = V_1 \cup \{x\} \setminus \{y\}$, $V_2' = V_2 \cup \{y\} \setminus \{x\}$ (with $x \in V_2$, $y \in V_1$). The size of the neighborhood is $|V|^2/4$. The other tactical decisions can be made as explained in Section 2.1.

Note that an alternative implementation is the following. Instead of restricting the solution space to bipartitions in subspaces of equal cardinality (balanced bipartitions), one allows for "illegal" solutions $(V_1, V_2)$ with $|V_1| \neq |V_2|$. The "solution" space is the set of *all* bipartitions and the objective function includes a term which penalizes unbalanced bipartitions:

$$\begin{aligned} F(V_1, V_2) \quad = \quad & \text{number of edges with one endpoint in } V_1 \\ & \text{and one endpoint in } V_2 + \gamma(|V_1| - |V_2|)^2 \end{aligned}$$

where $\gamma$ is a positive constant to be tuned. The neighborhood of a (non-necessarily balanced) bipartition is made of all bipartitions $(V_1', V_2')$ with $(V_1' = V_1 \cup \{x\}$ and $V_2' = V_2 \setminus \{x\})$ or $(V_1' = V_1 \setminus \{y\}$ and $V_2' = V_2 \cup \{y\})$, $x \in V_2$, $y \in V_1$. The advantage of this implementation is to provide new routes for escaping from local minima and to reduce the size of the neighborhoods.

As SA is a randomized algorithm, it is advisable to make a large number of independent runs. That's what Johnson *et al.* did in comparing the second implementation of SA with a local search algorithm (with the same neighborhood) and the Kernighan-Lin specialized heuristic (see Johnson *et al.* 1989 and also Section 3). Roughly speaking, they conclude that SA outperforms local search; it beats Kernighan-Lin on random graphs if running time is not taken into account and works slightly better if comparable times are allocated. However, on geometric random graphs (i. e. graphs with a special kind of geometric structure, see Johnson *et al.* 1989), SA is outclassed by Kernighan-Lin. Anyway, the running time which is necessary for an effective annealing is usually long compared with the time needed by deterministic heuristics.

## 3 More about Simulated Annealing

### 3.1 Introduction

"Annealing is a potentially valuable tool but in no ways a panacea". This is the conclusion of Johnson *et al.* in their second evaluation paper on SA (Johnson *et al.* 1991, p. 405). It is clear indeed that the theoretical results that insure the convergence of SA to the optimal solution are irrelevant in practice because the (necessary and sufficient) conditions on which they rely are strikingly violated in any practical application.

It is also true that SA is not competitive in some problems (an example of which is number partitioning studied in Johnson *et al.* 1992) and that a certain amount of work is needed to tailor SA to each particular application. This may appear as deceptive for the (lazy) people who dream of a final algorithm for solving all combinatorial optimization problems. Looking at SA on a more positive mood, one has to consider that its successes are remarkable, that a general heuristic can compete with the best specialized ones on a problem is surprising. But in my opinion, the main interest of SA is not there: it is that due to the conjunction of simplicity and effectiveness it offers a unique chance of *understanding why* a general heuristic can ever work or more precisely to characterize the (instances of) problems on which SA is competitive. While more relevant theory can also be expected, the simplicity of SA allows in addition for rigorous experimental work: Johnson *et al.*'s following statement is certainly a lot more valid for more complex approaches like Genetic Algorithms or even Tabu Search: "Although experiments are capable of demonstrating that the approach performs well, it is impossible for them to prove that it performs poorly. Defenders of SA can always say that we made the wrong implementation choices" (Johnson *et al.* 1989, p. 869).

As soon as solution space, objective function and neighborhood structure have been chosen, the only non-parametric choice in a SA implementation consists of selecting a cooling schedule family. As this, as well as parameters tuning, has been seriously investigated in some special cases, we report on these investigations in Sections 3.2 and 3.3 and we summarize the main practical conclusions. We also give a short non-technical overview of some existing theoretical results in Section 3.4. This is because this theory, even irrelevant in practice, has been and remains a source of inspiration in the trials for improving the method or providing generic solutions to the tactical choices involved. Note also that a similar role is played by the thermodynamic analogy in particular for suggesting adaptive cooling schedules (see Section 3.3.2).

## 3.2 Tuning the parameters of the basic model

In this section, we formulate some comments and practical recommendations in view of choosing the parameter values in the basic implementation of SA with geometric cooling schedule (see Section 2.1).

### 3.2.1 A graphical tool

Notice first that an elementary but useful tool for eliminating bad parameters choices is a *plot* of $F_n^*$ (the best observed value up to step $n$) against time, i. e. the number $n$ of steps. Such a graph usually looks like illustrated in Figure 3 in the case of acceptable geometric schedule.

An initial but not too long period of slow improvement is considered typical of a good choice of the parameters. Similarly, at the end it is advisable to wait for a clear sign that no substantial further improvement will be obtained before stopping. Note that non geometric cooling schedules yield different shapes (see Johnson *et al.* 1989, p. 884). Note also that it is an a posteriori evaluation tool.

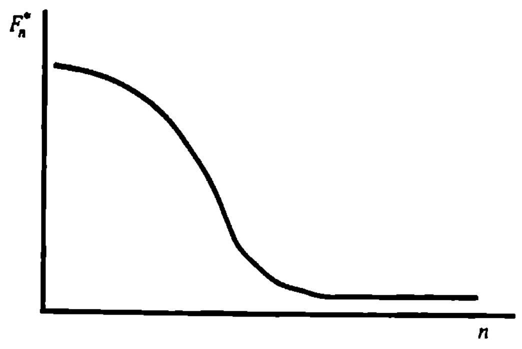

Figure 3: Typical behaviour of $F_n^*$ during annealing with geometric cooling schedule.

### 3.2.2 Initial temperature

The initial temperature $T_0$ of a geometric cooling schedule (it is also true for other schedules) is generally determined in order that the initial probability for accepting bad moves be approximately equal to a prescribed value $p_0$. This can be done by trial and error before starting the annealing process: the algorithm is run with a tentative initial temperature, the acceptance rate (of bad moves) is computed and $T_0$ adapted. Then, the algorithm is restarted with the modified value of $T_0$ and so on until an acceptance rate near $p_0$ is found. This procedure was used by Johnson *et al.*, but they report (Johnson *et al.* 1991, p. 405) that the acceptance rate computed on the basis of a few iterations is not a very robust estimator of $p_0$. They suggest for fixing $T_0$ to use problem specific algorithms or computations based on the desired value of $p_0$ and simple characteristics of the problem such as the size of a neighborhood.

How can we choose a value for $p_0$? It is commonly thought that it should be high. The idea is that the system must be allowed at the beginning to visit any region of the search space without much constraint. It was thought that this initial quasi-random walk was useful preparatory work which did not improve the objective function but would prove profitable in the long term. This seems not to be confirmed by experimentation. For example, in the graph partitioning problem, Johnson *et al.* have tested values of $p_0$ ranging from 0.1 to 0.9 and conclude that from $p_0 = 0.4$ on, there is no improvement in the final solution quality while computer use grows substantially. As a consequence, they worked with $p_0 = 0.4$. It is probably advisable to make some preliminary testing in each specific problem. If one is not too much concerned with reducing computer time, high values of $p_0$ will do no harm.

### 3.2.3 Length of plateau and cooling factor

For different reasons it is sensible to link the length $L$ of the plateau in a geometric schedule to the average size of the neighborhood. In view of effectiveness, when temperature is low and almost all bad moves are rejected it should be given a reasonable chance of trying all the neighbours of the current solution at least once. This option is also a sensible way of taking into account the size of the instance at hand as long as it leads to feasible computer times.

Turning now to the interrelations between $L$ and the cooling schedule $\alpha$, it is clear that they have positively correlated effects: increasing $L$ or $\alpha$ tends to increase the number of iterations and should also improve the final solution. More precisely, doubling $L$ or taking the square root of $\alpha$ should approximately double the total number of iterations. This is confirmed by Johnson *et al.* (1989, pp. 878-881) with the correction that increasing $\alpha$ (by taking square roots) seems to add less computer time for an equal improvement of $F^*$. These observations are only valid in a certain range of variation of $L$ and $\alpha$.

When $\alpha$ and/or $L$ are increased beyond a certain point, improvement of $F^*$ becomes so slow that better results are obtained by allocating the available computing time to several shorter runs of the algorithm (starting from different initial solutions) rather than to a single very long run. For a fixed amount of computing time, there is some evidence of the existence of an optimal compromise between the number of runs and their lengths (see Johnson *et al.* 1989, pp. 880-881). It is clear however that SA is a very computer intensive algorithm and that too short runs do not yield good results.

For being a little more specific, the values of $L$ and $\alpha$ used in graph partitioning experimentation by Johnson *et al.* are

$$L = 16 \times \text{ the average size of a neighborhood}$$

$$\alpha = 0.95$$

Those values are robust: mild variations of $L$ and $\alpha$ do not yield dramatically different results and the same values are used for all instances.

In conclusion, there is no single and general rule for choosing $L$ and $\alpha$. However, due to the (extrapolated) robustness of the algorithm, a little preliminary experimentation quickly yields acceptable values. The general qualitative remarks above can also help.

### 3.2.4 Stopping criterion

Usual values of $\epsilon_1$ and $\epsilon_2$ in the stopping criteria Stop 1 and Stop 2 (resp.) range from 1 to 5 (%). The constants $K_1$ and $K_2$ should not depend on the plateau length $L$ provided this length is linked to problem size as indicated in Section 3.2.3. Johnson *et al.* (1989) use Stop 2 with $\epsilon_2 = 2(\%)$ and $K_2 = 5$. It is advisable to control a posteriori the choice of the parameters by looking at the graph of $F_n^*$ against time and making sure that annealing was not stopped too early.

## 3.3 More structural options and some improvement opportunities

Besides parameter setting, there are relatively few degrees of freedom in a basic SA implementation: the choice of a solution space, an objective function, a neighborhood structure and a cooling schedule. We briefly give some general advice concerning the three former and extend a little more on the latter. The last subsection is concerned with a few technical improvements on the basic algorithm.

### 3.3.1 Solution space, objective function and neighborhood structure

Only general comments can be made at this level. Let us emphasize the fact already mentioned that better results can often be obtained both on the viewpoints of solution quality and computer use, by modeling some constraints as soft constraints, i. e. through penalties introduced in the objective function. Such penalties are monotone functions of the degree of violation of the constraints and vanish when the constraints are satisfied. This modeling option not only has an incidence on the objective function but also on the solution space as illegal or unfeasible solutions are considered. It is hoped that the solutions obtained at the end of the algorithm will satisfy the constraints. If not, a way out is to use a specialized heuristic to transform an illegal solution into a solution that satisfies the constraints, without loosing too much on the objective function. It can also happen that solutions which nearly satisfy the constraints are acceptable in practice. This is the case for instance in complex scheduling problems where the due dates are in general soft constraints.

The major reason for using soft constraints is to get simpler or smoother neighborhood structure. If one restricts oneself to feasible solution it may well be complicated and/or time consuming to generate a feasible neighbour to the current solution. Moreover, if the move is a complex transformation of the current solution, the change in the objective function resulting from the move may require explicit computation of the value of $F$ on the new solution which can also take time. In extreme cases, one is even unable to generate a feasible initial solution. In graph coloring, for instance, a feasible solution is a partition of the nodes in classes such that two nodes in the same class may not be linked by an edge. Whether there exists such a partition in any fixed number of classes ($>2$) is a NP-complete question.

Besides the possibility of easily generate it, another desirable property of the neighborhood structure is to allow for easy travelling throughout the space. In particular, it should be possible to access any solution from any other one by a finite chain of neighbour to neighbour moves.

### 3.3.2 Cooling schedules

The geometric cooling schedule used in the basic SA implementation is as far as one can imagine from the schedule that would theoretically insure convergence of SA to the optimal solution (see Section 3.4). In this schedule, temperature is decreased logarithmically with time i. e.

$$T_k = \frac{C}{1+\log(k)}$$

where $k = 1, 2, \ldots$ is the index of the series of $L$ steps at the same temperature $T_k$. Johnson *et al.* experimented on the graph partitioning problem with this "logarithmic schedule" as well as with the two following ones

- A schedule where temperature is decreased linearly with $k$.
- A schedule where the probability of accepting a move decreases linearly with $k$.

At least for the graph partitioning problem, there is no improvement w. r. t. the usual geometric schedule (even when allowing more than twice the same computer time for the logarithmic schedule) and on the other hand, a lack of robustness can be observed. The quality of the final solution is sensitive to the choice of the initial temperature which is a major drawback when there is no rigorous way of assigning precise values to the parameters.

In view of Johnsons's results there seems to be no reason for renounciating to the geometric schedule. There is however a different category of schedules called *adaptive* which have been proposed. The idea is to monitor dynamically the temperature evolution: decisions are taken after each observation period (a plateau) and temperature is varied on the basis of the intensity of the search during the passed period. Roughly speaking, temperature is slowly decreased or maintained constant when quick progresses are made. It is more rapidly decreased when the performance stagnates.

The following simple adaptive schedule was experimented with by Johnson *et al.* (1989, p. 882): one observes the best value and the average value in each series of $L$ steps. If a series ends up with at least one of these observed values better than the corresponding ones of the preceding period, the series of $L$ steps is repeated without temperature change. Otherwise, temperature is decreased by the usual factor $\alpha$. This schedule did not provide better final solutions although computer use was substantially increased.

More sophisticated attempts which in general rely on the thermodynamic analogy were made by several authors among which we point out Lam and

Delosme 1986, Huang *et al.* 1986, Pedersen 1990 (for further references see also van Laarhoven and Aarts 1987). All those approaches are based on the idea that quasi-equilibrium should be reached before temperature is varied and the change in temperature should be slow enough to allow to reach quickly a new quasi-equilibrium state for the new temperature. Although their authors claim that their approach leads to more or less spectacular improvement there is no methodic comparative study of such schedules in the available literature.

## 3.4 Possible technical improvements

We present a few suggestions for improving the efficiency of the basic SA algorithm. These suggestions were made by different authors and are evaluated in Johnson 1989 (pp. 881-888).

### 3.4.1 Cutoffs

Considering that it is the number of *accepted* moves rather than the number of trials that is important, it is proposed to stop the series of trials of a plateau as soon as a certain number of moves have been accepted. This number is chosen as a fixed proportion (= *cutoff parameter*) of the plateau length $L$. The main effect of a cutoff is to avoid spending much time at high temperature. In this, it is analogous to starting at lower temperature. Johnson *et al.* compared both approaches: for equal running times there seems to be little difference in solution quality. There can be an advantage in using a cutoff if it is more difficult to estimate a good initial temperature than a good cutoff parameter.

### 3.4.2 More efficient choice of the moves

If solutions in the neighborhood of the current one are chosen purely at random, there is a risk in the case only one neighbour is acceptable, that we miss it in a series of $L$ trials. A way of avoiding this in the *special case* of the graph partitioning problem is to choose the moves according to a *random permutation* of the set $V$ or vertices: in each series of $|V|$ trials each vertex will be considered for a move exactly once (in the order defined by the random permutation). This procedure seems to give better solution quality in equal computation time. It amounts to introducing an additional element of optimization in the local search (see also Section 5.3 for the description of a "locally optimized SA" algorithm which does not choose the moves at random but selects the best solution in a randomly generated subneighborhood of the current solution). Note that this approach is only applicable when there is a uniform description of all possible moves from any solution (i. e. a description which is invariant when the current solution is varied).

Another trick for improving effectiveness in the end of the schedules is the *rejection-free annealing* of Green and Supowit (1986). At low temperature much time is lost in considering moves that will be rejected. The proposed alternative is the following: for each move $i$, compute its probability of acceptance $p_i$. Let $P$ be the sum of all $p_i$'s. This yields a probability distribution on the set of all moves, each move $i$ receiving a probability $p_i/P$. Select a move according to this distribution and accept it automatically. Green and Supowit prove that this is equivalent to ordinary annealing. It leads to time savings at low temperature. This interesting proposition is however difficult to apply in general as the possibility of efficiently computing and updating the $p_i$'s is problem dependent.

### 3.4.3 Approximate exponentiation

As SA needs long runs to be efficient, a non-negligible speedup can be obtained by replacing the exact evaluation of the acceptance probability $p_n = \exp(-\Delta F_n/T(n))$ by an approximation. The best thing to do is to use a table of precomputed exponentials. Johnson *et al.* do as follows. The domain of variation of $\Delta F_n$ is bounded by $T(n)/200$ (move with negligible probability of acceptance: $\sim 0.5\%$) and $5T(n)$ (move with negligible probability of rejection: $\sim 0.5\%$). Moves for which $\Delta F_n$ is outside these limits are deterministically rejected or accepted. One computes $200\Delta F_n/T(n)$, rounds it down and looks at the corresponding value of the exponential in the table. For the graph partitioning problem, using this table saves 1/3 of the computing time without altering solution quality.

### 3.4.4 Better than random initial solutions

Starting from better than random initial solutions is inviting. SA theorists have claimed that the initial solution was "forgotten" (due to the ergodicity of a Markov chain, see Section 3.5) and it seems to be generally the case: e. g. in the graph partitioning problem, starting from the solution of the Kernighan-Lin algorithm does not lead to better final solutions than starting from random solutions. However, in the "geometric" graphs experimented with in Johnson *et al.* (1989), initial solutions obtained through a special heuristic that takes into account the graph structure can be used to get better final results. The initial temperature must be chosen low enough in order not to destroy the initial structure of the solution (see Johnson 1989, pp. 885-887 for more details).

## 3.5 An outline of some theoretical results

The behaviour of SA has been theoretically studied by many authors. We try to give a brief overview of the main trends.

The dynamics of evolution of the current solution in the search space during SA execution is that of a rather well-behaved Markov chain: the transition from the current solution $x$ to another solution $y$ (at current temperature $T$) is governed by the transition probability matrix $P_T(x, y)$ defined by

$$P_T(x,y) = \begin{cases} 0 & \text{if } y \notin V(x) \text{ and } y \neq x \\ \frac{1}{|V(x)|} & \text{if } y \neq x, y \in V(x) \text{ and } \Delta F \leq 0 \\ \frac{1}{|V(x)|} \cdot \exp(-\Delta F/T) & \text{if } y \neq x, y \in V(x) \text{ and } \Delta F > 0 \\ 1 - \sum_{z \neq x} P_T(x,z) & \text{if } y = x \end{cases}$$

This corresponds indeed exactly to what is done in practice: a solution $y$ is selected at random in $V(x)$, each $y \in V(x)$ having probability $1/|V(x)|$ of being chosen. Then, $\Delta F = F(y) - F(x)$ is computed and if $\Delta F \leq 0$, $y$ is accepted (case 2 in the definition of $P_T$). Otherwise, a second random decision is made independently with probability $\exp(-\Delta F/T)$ which yields the third case in the definition of $P_T$. The last case, the probability of remaining in $x$, is obtained by complementation to 1.

Two simple hypotheses insure that the Markov chains (at any *fixed* temperature $T$) tend to an equilibrium:

- **Hypothesis 1**: Symmetry of the neighborhood system i. e. $y \in V(x)$ iff $x \in V(y)$.
- **Hypothesis 2**: Connectivity, i. e. it is possible to reach any solution from any solution in a finite number of neighbour to neighbour moves.

Under these two hypotheses, the evolution governed by $P_T$ at any constant temperature $T$ tends to a stationary distribution

$$p_T(x) = \frac{\exp(-\frac{F(x)}{T})}{\sum_{z \in X} \exp(-\frac{F(z)}{T})}$$

This means that when enough time has elapsed, the probability of being in solution $x$ is approximately $p_T(x)$ or equivalently, the proportion of the time spent in each solution $x$ when the evolution is stabilized is $p_T(x)$. If one looks at the behaviour of probability $p_T$ when $T$ is decreased to 0, one sees that $p_T$ concentrates more and more on solutions of "low energy" i. e. values of $F$ near to the minimum. In the limit, $p_0$ is concentrated *exclusively* on the global minima:

$$p_0(x) = \begin{cases} 0 & \text{if } F(x) \neq F_{min} \\ \frac{1}{|X_{min}|} & \text{if } x \in \{y \in X : F(y) = F_{min}\} =: X_{min} \end{cases}$$

What does a SA algorithm in practice? It is not a single (homogeneous) Markov chain as $T$ varies and hence the transition probability matrix $P_T$. One can look at SA in two ways: as a *series of homogeneous* Markov chains with different transition probability matrices or as a single *inhomogeneous* Markov chain (with transition probability matrix evolving with time).

### 3.5.1 Series of homogeneous Markov chains

On each plateau of the cooling schedule illustrated in Figure 2, a Markov chain with constant transition probability matrix $P_T$ and unique stationary distribution $p_T$ is run for $L$ steps (= transitions). If $L$ is long enough, one may consider that one is not too far from equilibrium at the end of the plateau. Suppose that $T$ is not too abruptly decreased at the end of each plateau i. e. when the system is in "quasi-equilibrium". It is conceivable that the system will not be too much perturbated and will quickly reach a new quasi-equilibrium state for the new temperature $\alpha T$. Iterating the process until temperature is low, the probability of any solution $x$ will be approximately $p_T(x)$ with small $T$ which means that solutions far from the minimal value of $F$ will almost be impossible.

Such an intuitive reasoning was formalized by several authors: Aarts and van Laarhoven 1985, Lundy and Mees 1986, Otten and van Ginneken 1984, Mitra *et al.* 1985. Some conditions on the cooling schedule must of course be imposed: those by Mitra *et al.* (1985) are the most general and least restrictive (see van Laarhoven and Aarts 1987, pp. 17-26).

### 3.5.2 Inhomogeneous Markov chain

In this approach, the dynamics is that of a single Markov chain with time-dependent transition probability matrix: $T$ is not bounded to follow a schedule as illustrated in Figure 2. It may even be adapted at each step. The problem is to give conditions under which the chain will end up in a global minimum with probability 1. Geman and Geman 1984, Gidas 1985, and Hajek 1988 investigated the problem. Later work is concerned with more general processes. We present here Hajek's necessary and sufficient conditions on the cooling schedule that guarantee convergence to a global minimum.

We first define the notion of *reachability at a certain height*: solution $y$ is reachable from solution $x$ at height $h$ if there is a sequence of solutions $x = x_0, x_1, \ldots, x_n = y$ for some $n \geq 1$ such that $x_{i+1} \in V(x_i)$, $i = 0, \ldots, n-1$ and $F(x_i) \leq h$ for all $i = 0, \ldots, n$. This is a very intuitive notion: it means that one can walk from $x$ to $y$ along a path that does not let climb above height $h$. The *weak reversibility property* is the following: $y$ is reachable from $x$ at height $h$ iff $x$ is reachable from $y$ at height $h$ (for all $x$, $y$, $h$).

*If* the neighborhood structure is connected and weakly reversible and *if* the cooling schedule $T(n)$ is a non-increasing sequence of strictly positive real numbers tending to 0, *then* the Markov chain with non-homogeneous transition probability matrix $P_{T(n)}$ will end up in a global minimum of $F$

with probability 1 *iff* the cooling schedule $T(n)$ is such that

$$\sum_{n=1}^{\infty} \exp(\frac{-d^*}{T(n)}) = \infty$$

where $d^*$ is the maximal difference of height one can have to climb for escaping from a local minimum by the easiest path.

Note that the condition on the schedule essentially assumes logarithmic decrease. For instance

$$T(n) = \frac{C}{\ln(n+1)}$$

satisfies the condition provided $C \geq d^*$.

# 4 Tabu Search (TS)

## 4.1 General presentation of Tabu Search

Tabu Search (TS) is another local search strategy designed for escaping from local minima: even if there is no better solution than the current solution $x_n$ in its neighborhood $V(x_n)$, one moves to the *best possible* solution $x$ in $V(x_n)$ or a sub-neighborhood $V'(x_n) \subseteq V(x_n)$ in the case where $V(x_n)$ is too big to be explored efficiently. If the neighborhood structure is symmetric, i. e. if $x_n$ belongs to the neighborhood $V(x)$ of $x$ whenever $x \in V(x_n)$, there is a danger of *cycling* when at the next step we explore $V(x)$: there is indeed a chance that $x_n$ could be the best solution in $V(x)$ in which case we would come back to $x_n$ and from then on, oscillate between $x$ and $x_n$. To avoid this situation and more general cycling situations, the idea is to store the last pairs $(x_n, x)$ of solutions in a list called "tabu list". If the pair $(x_n, x)$ is in the list, the move $x \rightarrow x_n$ is forbidden for a certain number of subsequent moves. This principle raises some technical problems: storing a complete description of the last solutions visited and testing for each candidate move whether it is the converse of a move recorded in the list might be rather time consuming. An alternative is to store a characteristic or an *attribute* of the moves (it can be the transformation performed on the current solution, e.

g. flipping the ith coordinate value of a binary vector from 0 to 1). This can prove too restrictive or alternatively too permissive as illustrated in the examples below (borrowed from de Werra 1988).

**Example 1**: Consider the search space $X$ which is the set of pairs of distinct elements of the set $A = \{a, b, c, d, e\}$ and have a look at the dynamics illustrated in Table 1. Suppose the converse of the last 3 transformations

| current solution | transformation | tabu list |
|---|---|---|
| $ab$ | $b \leftarrow c$ | $c \leftarrow b$ |
| $ac$ | $a \leftarrow d$ | $d \leftarrow a$, $c \leftarrow b$ |
| $cd$ | $c \leftarrow e$ | $c \leftarrow e$, $d \leftarrow a$, $c \leftarrow b$ |
| $de$ | | |

Table 1: Example 1.

are tabu. A move is the substitution of an element of $A$ by another in a solution: e. g. "$b \leftarrow c$" means that $b$ is replaced by $c$. Starting from $ab \in X$ and performing successively the moves represented in the second column of Table 1 yields the solutions below $ab$ in the first column. The tabu list is updated at each step, the attribute of the last move ("transformation") enters the list and the most ancient attribute in the list leaves it (as soon as it is full i. e. contains 3 transformations). When solution $de$ is reached, the tabu list prohibits to move to $ae$ which was *not* visited before: tabu is too restrictive in this case. ■

**Example 2**: Let $X$ be the set of triplets of distinct elements of the same set $A = \{a, b, c, d, e\}$ and consider the dynamics shown in Table 2. The maximal length of the tabu list is fixed to 3. The tabu list is unable here to prevent cycling. ■

Despite the possible lack of effectiveness illustrated in the second example above, the tabu list is usually a list of one or several attributes of the

| current solution | transformation | tabu list |
|---|---|---|
| $abc$ | $c \leftarrow d$ | $d \leftarrow c$ |
| $abd$ | $b \leftarrow c$ | $c \leftarrow b$,<br>$d \leftarrow c$ |
| $acd$ | $d \leftarrow b$ | $b \leftarrow d$,<br>$c \leftarrow b$,<br>$d \leftarrow c$ |
| $abc$ | | |

Table 2: Example 2.

recently visited solutions or (of the converse) of the most recent moves: these attributes should be chosen accurately. As shown in the first example above, the prohibition of solutions with a given attribute is likely to be restrictive in excluding much more solutions than the just visited one. To correct the bad consequences of this (not all consequences are undesirable), one offers the possibility of overwriting the tabu status of a move when it leads to a good enough solution. More formally, we define an *aspiration level* that describes what is a good enough solution. Two elementary examples of aspiration criteria are:

- Asp 1: A solution is above the current aspiration level if it is better than any solution met before (as measured by the objective function $F$).
- Asp 2: (This criterion may be used when the tabu list consists of pairs associated with moves $x \longrightarrow y$ and composed of an attribute of the converse move $y \longrightarrow x$ and the value $F(x)$) If the attribute of the move $y' \longrightarrow x'$ is in the tabu list, one allows for the move if $F(x') < F(x)$.

Summarizing the above, we have the following scheme for a typical TS algorithm.

**Tabu Search (TS)**

- Initialization: Select an initial solution $x_1$ in $X$

$$\begin{aligned} F^* &\leftarrow F(x_1) \\ x^* &\leftarrow x_1 \end{aligned}$$

  Tabu list $TL$ is empty.

- Step $n = 1, 2, \ldots$: $x_n$ denotes the current solution. $\overline{F}$ is used to store the best accessible value of $F$ met during the exploration of the subneighborhood $V'(x_n)$. $\overline{x}$ denotes the solution in $V'(x_n)$ for which $F(\overline{x}) = \overline{F}$. Initialize $\overline{F}$ to $\infty$ at the beginning of each step.

  For all $x$ in $V'(x_n)$,

  > If $F(x) < \overline{F}$ and (If the move $(x_n \rightarrow x)$ is not tabu or if the move is tabu but passes the aspiration criterion) then $\overline{F} \leftarrow F(x)$ and $\overline{x} \leftarrow x$.

  $x_{n+1} \leftarrow \overline{x}$

  If $\overline{F} < F^*$, then $x^* \leftarrow \overline{x}$ and $F^* \leftarrow \overline{F}$.

  The appropriate characteristic of the move $(x_n \rightarrow x_{n+1})$ enters the tabu list once the first entered characteristic has been removed from the list if the list was full.

- End: If the stopping criterion is fulfilled, then stop.

The list of the tactical choices that have to be made is somewhat longer than for SA. In addition, the usual choices are less standard. More positively, there is more room for creativity in a TS application. The principal decisions to be made are:

- The specification of a neighborhood structure and possibly of a subneighborhhood structure (for the subneighborhood $V'(x_n)$ a generic possibility is to pick at random a fixed number of solutions in $V(x_n)$).

- The choice of the move attributes to be recorded in the tabu list.

- The choice of the tabu list length.
- The choice of an aspiration criterion.
- The selection of a stopping rule (usually the total number of iterations is fixed a priori).

## 4.2 Bibliographic and historic note

The idea of TS is due to F. Glover (Glover 1986). Similar views were developed by P. Hansen (Hansen 1986) who formulates a steepest ascent/mildest descent heuristic principle. A comprehensive description of TS can be found in Glover *et al.* (1992) as well as a bibliography of about 70 papers and books including many application reports. Although the literature about TS is not as extensive as about SA, it remains that the method has been applied to a large variety of combinatorial optimization problems like scheduling, transportation, electronic circuit design, graph coloring, neural networks, ..., and so on. As far as I know, there are no theoretical results such as convergence results about TS.

## 4.3 An example: graph coloring

The best way of understanding exactly how TS works is to look at an application. A simple and convincing example in graph coloring is due to Hertz and de Werra (1987). The problem is to find a coloring of the vertices $V$ of a non-oriented graph $G = (V, E)$ such that the same color is not assigned to two adjacent vertices and the minimal number of colors are used. Hertz and de Werra consider alternatively the problem of coloring the vertices with a fixed number $l$ of colors and minimizing the number of faults, i. e. adjacent vertices which are painted the same color. Hence, coloring adjacent vertices with the same color is accepted but penalized. The authors' strategy consists in finding a perfect coloring (without faults) for a large initial value of $l$ then find successive perfect colorings for smaller and smaller $l$. The procedure stops when the algorithm is not able to find a coloring with the current number of allowed colors.

In this implementation, a "solution" $x$ is any $l$-partition of $V$: $x = (V_1, \ldots, V_l)$. Clearly, in this approach, most of the solutions are not admissible (i. e. are not perfect colorings). This is just like in the second implementation of SA in graph partitioning. The objective function $F$ (to minimize) is defined on all $l$-partitions $x = (V_1, \ldots, V_l)$ of $V$:

$$F(x) = \sum_{i=1}^{l} |E_i|$$

where $E_i$ is the set of edges whose endpoints are both in $V_i$. The neighborhood $V(x)$ is the set of all $l$-partitions which differ from $x = (V_1, \ldots, V_l)$ by the transfer of exactly one "bad vertex" from some class $V_i$ to some other class $V_j$ (a "bad vertex" in $V_i$ is a vertex linked with another vertex of $V_i$). During local search, not all solutions of $V(x)$ are evaluated (there are too many of them). A sample $V'(x)$ is drawn at random from $V(x)$: the size of the subneighborhood $V'(x)$ is a parameter of the algorithm. Only non-tabu solutions or tabu solutions that satisfy the aspiration criterion are accepted in $V'(x)$. The best solution from $V'(x)$ becomes the new current solution. The tabu list (TL) records the vertices transferred during the last $k$ iterations together with their color before the transfer. The TL prevents a vertex transferred during the last $k$ iterations from coming back to its original color. The aspiration criterion is satisfied (and the tabu status of a transfer overwritten) when a move from solution $x$ to solution $x'$ is such that $F(x') < F(x)$ and never in the past has a move been improving a solution of value $F(x)$ to one of value as good as $F(x')$.

Note that some tricks are used to reduce computation time: the neighborhood random generation is stopped as soon as is found a non-tabu solution which is better than the current best solution. Special star-shaped configurations are searched for and reduced, i. e. some local optimization work is done on the current solution (see Hertz and de Werra 1987).

The authors experimented on random graphs whose number of nodes ranges from 100 to 1000 and whose edge density is 0.5 (= probabiLty of presence of each edge). They took the "magic number 7" for the length $k$ of their tabu list, the size of the randomly generated subneighborhoods grows with the number of nodes of the instance at hand. The results are compared with those obtained by an implementation of SA which was run on the same instances: better results are obtained with TS and using less computer time

(Johnson *et al.* 1991 do not confirm this appreciation, see also Section 5.1 below). The same authors also developed a so-called combined method which looks interesting and will be briefly discussed in Section 5.2.

# 5 Tabu Search and mixed heuristics

## 5.1 Comparing heuristics

Let us come back to and extend a statement already made in Section 3.1 about SA: none of the current "general" heuristic principles is a panacea and all of them need substantial work to be tailored to a problem. Experimental work indeed tends to show that the way different heuristics compare, strongly depends on their implementation and the problem instance to which they are applied. In their exemplary investigations on the application of SA to graph coloring, Johnson *et al.* (1991) compare implementations of SA using different neighborhood structures and a classical heuristic that has been randomized (XLRF). Their conclusion indicates that the competition winner varies according to factors like the instance size, the edge density (for random graphs), the geometry of the graph (see Johnson *et al.* 1991, Table VIII, p. 399). Johnson *et al.* also experimented with the TS algorithm described in Section 4.3 but do not confirm Hertz and de Werra's conclusions: they find no *general* domination of TS over SA. A suggested explanation is that their implementation of SA is substantially faster than Hertz and de Werra's.

The clearest conclusion to be drawn from these experiments is probably that pragmatism should prevail: there is no reason for claiming the absolute superiority of any existing general heuristic. What we want to illustrate in this section is that it is not only allowed but can be profitable to depart from the orthodoxy and build up more complex but competitive algorithms that integrate some original principles of TS, SA or other heuristics: neither SA nor even TS is tabu! And this is also true for Genetic Algorithms and Neural Networks. Experiments which tend to mix the different methods will probably develop in the next years and some new general heuristic search principles and algorithms will hopefully emerge. This could also contribute to bridge the gap between Operational Research and Artificial Intelligence.

## 5.2 A combined algorithm for graph coloring

For large graphs (more than 500 vertices), Hertz and de Werra in their already mentioned paper (Hertz and de Werra 1987) suggest to use TS to solve partial problems included in another coloring graph algorithm (presented in Chams *et al.* 1987). In this approach, TS not only helps in coloring subgraphs but also in constructing a large independent set, i. e. a subset of vertices such that no pair of them is linked by an edge. The global graph coloring algorithm works as follows. Fix an integer $q$. Construct an independent set $V_1$ which is as large as possible. Repeat this construction in $V \setminus V_1$, yielding $V_2$. Then, in $V \setminus (V_1 \cup V_2)$ yielding $V_3$ and so on, until the number of vertices in $V \setminus (V_1 \cup V_2 \cup V_3 \cup \ldots)$ is smaller than $q$. All vertices of each set $V_1$, $V_2$, $\ldots$, receive the same color and the remaining ($\leq q$) vertices are colored by using the TS algorithm described in Section 4.3.

TS is able to find a large independent set in the following way. Let $G' = (V', E')$ be the graph in which we search for a maximal independent set. A (non necessarily admissible) solution $x$ is a bipartition $(S, \overline{S})$ of $V'$, where $S$, the candidate independent set, is evaluated by $F'(x) = F'(S) =$ the number of edges linking two nodes within $S$. Solutions $x$ in which no "bad" edges are present ($F'(x) = 0$) and $|S|$ is as large as possible are searched for. The algorithm starts with an estimation of the size $p$ of the maximal independent set in $V'$ (theoretical results on random graphs are used). The solution space $X$ is the set of all bipartitions $(S, \overline{S})$ of $V'$ with $|S| = p$. The neighbours of a solution $x = (S, \overline{S})$ are obtained by exchanging a vertex from $S$ with a vertex from $\overline{S}$. Two tabu lists of equal length are kept: $T(S)$ (resp. $T(\overline{S})$) is the list of the last $k$ vertices which arrived in $S$ (resp. $\overline{S}$). Let $x_n = (S_n, \overline{S}_n)$ denote the current solution at the beginning of step $n$. One ranks the vertices of $S_n$ in decreasing order of their number of neighbours in $S_n$ and the vertices of $\overline{S}_n$ in increasing order of their number of neighbours in $S_n$ (not in $\overline{S}_n$!). The first vertex in $S_n \setminus T(S_n)$ is exchanged with the first in $\overline{S}_n \setminus T(\overline{S}_n)$ The algorithm stops as soon as $F' = 0$ or after the maximal number of allowed iterations. If an independent set has not been found ($F' > 0$), the algorithm is restarted with $p = p - 1$.

According to Hertz and de Werra who report on experiments made on samples of random graphs (density 0.5), it is advisable to use the combined algorithm when the number of nodes is well above 500. In such a case, the search for large independent sets is stopped when the number of not yet colored nodes falls below $q = 500$. Johnson *et al.* (1991, p. 400) suggest that there is much room for further improvement both with new implementations of TS and SA and with hybrid algorithms like the one just described.

## 5.3 About heuristic search principles

It has been shown in the preceding section that it could be helpful to integrate general heuristics into a specialized algorithm. We will now illustrate the possible fertilization due to mixing techniques borrowed from different general heuristics, namely TS and SA. The particular examples described below can be interpreted as successful applications of general principles of heuristic search.

According to Glover (1986) a good heuristic search strategy is a succession of phases of ***diversification*** and ***intensification***, the alternance being governed by the exploitation of historic information. Typically, the descent algorithm lacks diversification phases while the "intensification" is permanent (local optimization). Both TS and SA allow to explore different regions of the search space instead of being trapped in local minima: the tabu list as well as high initial temperature are diversification devices. In SA, the relative importance of intensification w. r. t. diversification is increasing with time as temperature is decreasing. Tabu Search has built-in elements of diversification and intensification and it is easy to imagine additional ones, e. g.

- To penalize already visited regions. For instance, in looking for a maximal independent set, one could replace $F(x) = F(S)$ = number of edges in $S$ by

  $$\tilde{F}(x) = F(x) + \alpha \sum w_i$$

  where $w_i$ is the number of iterations with node $i$ in $S$ and $\alpha$ is a weighting factor that generally varies with time (taking historic information into account or not).

- Vary the length of the tabu list during the search (taking historic information into account or not).

A systematic review of various ways of implementing the above principle in TS is provided in Glover *et al.* (1992). The principle however applies, in principle, to any kind of heuristic and is not specific to TS. To illustrate this, we briefly describe an application in which the above ideas were applied to SA and to TS.

The problem can be characterized as a homogeneous grouping one. A set of $J$ objects described by $I$ characteristic properties has to be partitioned in groups of equal size $K$ in such a way that the mean value of each character is approximately the same in each group. The problem can be obviously treated by SA or TS with a natural neighborhood structure: the solution space is the space of all partitions in subsets of size $K$ and the neighborhood of a solution $x$ (i. e. a partition $x$) is the set of all solutions that can be obtained from $x$ by exchanging two objects belonging to two different groups. A modification of the classical SA procedure, called "locally optimized SA" is obtained by replacing at each step the solution drawn at random in the neighborhood of the current one by *the best* solution in a subneighborhood of the current solution (like in TS). This constitutes an additional intensification element inspired from TS. On the other hand, in the basic implementation of TS for this problem, the determination of the objective function (which measures the degree of homogeneity of a partition) after one move is bounded above: if the best move in a subneighborhood causes the objective function to deteriorate by more than an a priori fixed amount $A$, one generates another subneighborhood and searches for another move. A straightforward modification of this scheme consists in varying $A$ during the course of the algorithm. This was done by introducing a "cooling schedule" for $A$. In this case as well, it was another intensification element that was added. Both modified algorithms led to substantial improvement as compared to the original SA and TS simpler versions (see Liégeois *et al.* 1992).

# 6 Genetic Algorithms (GA)

## 6.1 General presentation of Genetic Algorithms

A genetic algorithm (GA) may be described as a mechanism that mimics the genetic evolution of a species. The main difference with the two former approaches, SA and TS, is that GA deal with *populations* of solutions rather than with single solutions. An obvious advantage is intrinsic parallelism but it goes beyond letting solutions evolve independently in parallel: solutions do *interact*, mix together and produce "children" that hopefully retain the good characteristics of their parents. GA can be viewed as a form of local search but in a generalized sense. It is not the neighborhood of a single solution which is explored but the neighborhood of a whole population: this is something different from the union of the individual neighborhoods due to interaction. The main operators used to generate and explore the neighborhood of a population and select a new generation are *selection*, *crossover*, and *mutation*. We describe those operators below. Note that the GA literature is rich in terms borrowed from Genetics, sometimes with a little of pedantry: we shall limit our use of the genetic jargon to a minimum.

The first peculiarity of GA is that the genetic operators do not operate directly on the solution space: solutions have to be *coded* as finite-length strings over a finite alphabet. This makes little difference with common optimization practice in some situations like mathematical programming problems with 0-1 variables as a natural coding of a solution is a bitstring containing the values of each of the boolean variables in some predefined order. However, this is less easy in some other situations and above all, the straightforward coding is not always the most appropriate. From now on in the framework of GA, when we write "solution" we mean "coded representation of a solution" unless otherwise stated. In the GA literature a string representation of a solution is named a "*chromosome*". The feature associated with each string coordinate is a "*gene*" and the value of a gene is an "*allele*". Each position in a string is a "*locus*" (in the simplest cases, each gene is associated with a locus).

A GA starts with an initial population of say $N$ solutions and let it evolve yielding a population of the same size $N$ at each iteration. Very roughly, the $(n+1)$th generation of solutions is obtained from the $n$th generation $X^{(n)}$ through the following procedure: the best individuals (= solutions) from $X^{(n)}$ are selected, crossover operations are performed on pairs of them. This yields an offspring of solutions which will replace the bad individuals of the current population. Mutation is generally performed on a small proportion of the "children".

Let us go a little more into the detail. Each solution of the current population $\{x_1, \ldots, x_N\}$ is evaluated by its "fitness" which can simply be the value of the objective function in a maximization problem. More generally, the *fitness* of a solution is an appropriate monotone transformation of its evaluation by the objective function. Let $F$ denote, in this section, the fitness function which we want to *maximize* over the solution space.

The selection of the "best" individuals from a given population is done according to their fitness but not in a deterministic way: solutions are drawn at random with replacement from the current population with a probability that increases with their fitness. A simple choice for such a probability is as follows: for all $i = 1, \ldots, N$, $x_i$ is selected with probability

$$\frac{F(x_i) - F_{min}}{\sum_{j=1}^{N}(F(x_j) - F_{min})}$$

where $F_{min} = \min\{F(x_j), j = 1, \ldots, N\}$. Pairs of selected individuals are then submitted (with some probability $\chi$) to the crossover operation. There are lots of possibilities for defining this operator depending on the problem and its coding. The commonest example called "2-point crossover" works as follows. Suppose the solutions are coded in bitstrings of length 8 and that the following pairs of individuals were selected

$$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

Two positions, say 3rd and 5th, are chosen at random and the characters between those two positions (inclusively) are swapped, yielding two "children" solutions. In our example, we swap the characters at the 3rd, 4th and 5th positions in both strings

$$\begin{array}{cc|ccc|ccc} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{array} \rightarrow \begin{array}{cc|ccc|ccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \end{array}$$

Each child is then submitted to mutation (with some probability $\mu$). The simplest mutation operator consists of choosing a position at random and substituting the character in that position by another character from the alphabet. For instance, working on the 8-positions bitstring

$$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}$$

and performing a mutation at position 6 yields

$$0 \quad 1 \quad 0 \quad 1 \quad 1 \quad \underline{1} \quad 0 \quad 1$$

The final step in the generation of a new population is the substitution of "bad" individuals of the current population by the (possibly mutated) children. The "bad" individuals are selected according to their fitness in a randomized way (much as was done for selecting the good individuals): solutions are drawn at random without replacement with a probability that *decreases* with their fitness. This procedure yields the $(n+1)$th generation. The algorithm generally stops after a preassigned number of generations have been produced.

As a summary, we present a schematic description of a typical GA. Note that function $F$ (= fitness) is to be *maximized* on the space $X$ of *coded* solutions. Many variants of this basic scheme can be found in the literature.

**Genetic Algorithm (GA)**

- Initialization:

  Select an initial population $X^{(1)} = \{x_1^{(1)}, \ldots, x_N^{(1)}\} \subseteq X$

  $$\begin{aligned} F^* &\leftarrow \max\{F(x_i^{(1)}), i = 1, \ldots, N\} \\ x^* &\leftarrow \arg\max\{F(x_i^{(1)}), i = 1, \ldots, N\} \end{aligned}$$

- Step $n = 1, 2, \ldots$: $X^{(n)}$ denotes the current population of solutions

  a) Selection of good individuals from $X^{(n)}$:
  Let $\{y_j, j = 1, \ldots, 2M\}$ be $2M$ individuals drawn with replacement from $X^{(n)}$, the probability of choosing $x_i^{(n)}$ being an increasing function of $F(x_i^{(n)})$.

  b) Crossover:
  For $k = 1, \ldots, M$, the crossover operator is applied to the pairs $(y_{2k}, y_{2k+1})$ with probability $\chi$: this yields $M$ pairs of children $(z_{2k}, z_{2k+1})$ (which are identical to their parents with probability $(1-\chi)$).

c) Mutation:

For $j = 1, \ldots, 2M$, the mutation operator is applied to $z_j$ with probabilty $\mu$: this yields $2M$ (possibly mutated) children $w_j, j = 1, \ldots, 2M$ (which are identical to $z_j$ with probability $(1 - \mu)$).

d) Substitution of the bad individuals:

Draw $2M$ individuals from $X^{(n)}$ without replacement, the probability of choosing $x_i^{(n)}$ being a decreasing function of $F(x_i^{(n)})$. $X^{(n+1)}$ is obtained by substituting the $2M$ selected "bad" individuals from $X^{(n)}$ by the children $\{z_j, j = 1, \ldots, 2M\}$.

For all $j = 1, \ldots, N$, if $F(x_j^{(n+1)}) > F^*$, then

$$\begin{aligned} F^* &\leftarrow F(x_j^{(n+1)}) \\ x^* &\leftarrow x_j^{(n+1)} \end{aligned}$$

- End: If $(n + 1) \geq$ a fixed number of iterations, then stop.

## 6.2 A didactic example and some general remarks

We illustrate the above complicated procedure on a simple example borrowed from Goldberg (1989, pp. 14 sq.). Consider the problem of maximizing $F(x) = x^2$ on the set of integers $\{0, 1, \ldots, 31\}$ and let a GA be used for this purpose. An easy coding of the solutions is by bitstrings of length 5. Let us start with the initial population of 4 solutions in Table 3: it was drawn at random by coin tossing.

In this example, the whole population is replaced by children at each iteration (i. e. $2M = N = 4$). The good individuals selected for reproduction are shown in Table 4. Note that $x_2^{(1)}$ appears twice. A one-point crossover operator is used: all characters positioned after the selected cutpoint are swapped. The cutpoints for the two pairs of parents are shown in Table 4. The crossover probability $\chi$ is assumed to be 1 and the mutation probability is $\mu = 10^{-3}$. No mutation is simulated.

The average fitness progressed from 293 in $X^{(1)}$ to 439 in $X^{(2)}$. Proceeding in this way and due to the selection mechanism designed to favour

| $j$ | population $\{x_j^{(1)}, j = 1,\ldots,4\}$ | solution | $F(x)$ |
|---|---|---|---|
| 1 | 0 1 1 0 1 | 13 | 169 |
| 2 | 1 1 0 0 0 | 24 | 576 |
| 3 | 0 1 0 0 0 | 8 | 64 |
| 4 | 1 0 0 1 1 | 19 | 361 |

Table 3: Initial population.

| | selected "good" individuals $\{y_j, j = 1,\ldots,2M\}$ | crossover site | children $z_j = w_j$ | solution | $F(z_j)$ |
|---|---|---|---|---|---|
| 1 | 0 1 1 0 \| 1 | 4 | 0 1 1 0 0 | 12 | 144 |
| 2 | 1 1 0 0 \| 0 | 4 | 1 1 0 0 1 | 25 | 625 |
| 3 | 1 1 \| 0 0 0 | 2 | 1 1 0 1 1 | 27 | 729 |
| 4 | 1 0 \| 0 1 1 | 2 | 1 0 0 0 0 | 16 | 256 |

Table 4: First iteration.

the fittest individuals it can be hoped that the final population will contain very good solutions. In the example a very good solution (11011 = 27) is already produced after one step. It is crucial however that sufficient diversity be maintained in the population in order to permit the exploration of as many "good regions" of the solution space as possible and not to restrict the search to the vicinity of a (local) maximum. Here again Glover's concepts of diversification and intensification which were alluded to in Section 5 are relevant. The persistence of a diversified population can be achieved by a careful tuning of the (many) parameters of the algorithm, namely:

- Population size,
- Replacement rate, i. e. number of children substituting old solutions at each generation,
- Crossover probability ($\chi$) and mutation probability ($\mu$),
- Number of iterations.

But more structural choices have to be made as well, e. g.:

- A good coding for the solutions,
- Adequate crossover and mutation operators.

These are probably essential decisions as it is generally believed among the GA community that the success of GA is due to the progressive proliferation of good *schemata* in the population, i. e. specific substrings which would be associated with *properties* which *characterize* optimal or near-optimal solutions. This clearly implies that the coding of the solutions should be in some sense *meaningful*, i. e. should implicitly give a semantic description of what is a good solution. For instance, in the above simple example, solutions with a "1" in the first position are better than any solution with a "0" in the first position. The reader interested in this theoretical debate is referred to the next section for some more precise elements and more generally to the books by Holland (1975) and Goldberg (1989).

## 6.3 Bibliographic and historic note

The origins of GA lay in the foundation of a theory of adaptive systems essentially initiated by J. H. Holland. His book of 1975 "Adaptation in natural and artificial systems" (Holland 1975) is the Bible of the GA community. An introductory account of the theory as well as its main developments and applications can be found in the excellent introduction to GA by Goldberg (1989). In fact, function optimization is the most trivial application of the theory which ambition is to be relevant in fields like data structure design, algorithm design, computer operating system adaptive control, ..., all fields belonging to computer science. Since De Jong's thesis in 1975 however, a large part of the activity of the GA community has been devoted to the less ambitious but more accessible subject of function optimization (see Goldberg 1989, pp. 126-127 for a list of applications of GA to optimization problems, see also the proceedings of specialized conferences on GA: Grefenstette 1985, 1987, Schaffer 1989, Belew and Booker 1991, Schwefel 1991, Männer and Manderick 1992). De Jong is known for being the first who seriously investigated in an experimental manner the potentialities of GA and he started with the simplest possible situation, i. e. function optimization (De Jong 1975, see also Goldberg 1989, pp. 106 et sq. for an account).

The original motivations and backgrounds of the pioneers of GA probably induced a development at the frontiers of computer science. By now, research in GA can be considered to belong to the field of artificial intelligence (AI). The GA community appears as a sociological microcosm even though about half of the applications belong to the field of OR. It is remarkable that even when dealing with classical problems of OR, the GA researchers seldom compare their results in a systematic manner with those obtained by OR researchers. This is probably a behaviour which will tend to attenuate in the future (see e. g. the paper by Yamada and Nakano 1992 where the authors apply GA techniques to Job Shop Scheduling test problems).

## 6.4 A genetic algorithm for the travelling salesman problem (TSP)

Several attempts at "solving" the famous travelling salesman problem (TSP) have been made using different implementations of GA's. We present here the most elementary of these algorithms in order to illustrate further the possible application of GA's. In the TSP, a traveller has to visit $C$ cities exactly once and go back to his starting point. The problem (which is NP-hard) is to find a tour of the cities at lowest cost (a shortest tour in the case of a euclidean TSP). In this GA for solving the TSP, a solution, i. e. a tour, is represented by an ordered list of the cities. For instance, if there are $C = 9$ cities,

$$3 \quad 5 \quad 7 \quad 1 \quad 2 \quad 4 \quad 8 \quad 6 \quad 9$$

represents the tour that passes successively in the cities $3, 5, 7, \ldots, 9$ and then comes back from 9 to 3. A first difficulty is to define a crossover-operator as the usual 2-point crossover (see Section 6.1) is not applicable. Indeed, consider that we have to produce children from the parents

$$\begin{array}{rccccccccc} A = & 3 & 5 & 7 & 1 & 2 & 4 & 8 & 6 & 9 \\ B = & 1 & 9 & 2 & 3 & 4 & 6 & 8 & 7 & 5 \end{array}$$

Suppose that we select the third and sixth positions for 2-point crossover. That would yield the following "tours" as children

$$\begin{array}{cc|cccc|ccc} 3 & 5 & 2 & 3 & 4 & 6 & 8 & 6 & 9 \\ 1 & 9 & 7 & 1 & 2 & 4 & 8 & 7 & 5 \end{array}$$

but those tours are meaningless as some cities appear twice while some other cities are not visited at all. One of the crossover operators specially designed for dealing with "chromosomes" that represent a permutation (as is the case here) is named OX (for order crossover) and works as follows. As for 2-point crossover, two positions are selected, let say the 3rd and the 6th as on the example above, both parents $A$ and $B$ are then prepared in order to make it possible to transfer the "genes" 3 to 6 of $A$ in the corresponding section of $B$ and vice versa. To prepare $B$, for instance, holes ($H$) are created at the places where are the genes that will come from $A$, i. e.

$$B: \; H \;\; 9 \,|\, H \;\; 3 \;\; H \;\; 6 \,|\, 8 \;\; H \;\; 5$$

Then, the holes are filled in by moving non-holes that are on their right in the chromosome. This starts from the second cut position (i. e. from gene in the 7th position on). When the last gene is attained one goes back to the gene in the first, second, ..., position, in circular order:

$$B: \; 3 \;\; 6 \,|\, H \;\; H \;\; H \;\; H \,|\, 8 \;\; 5 \;\; 9$$

At the end of this process, there are only "holes" in the exchange section and the genes from the exchange section of $A$ can be imported, yielding

$$B': \; 3 \;\; 6 \,|\, 7 \;\; 1 \;\; 2 \;\; 4 \,|\, 8 \;\; 5 \;\; 9$$

Preparing $A$ similarly, one gets:

$$A': \; 7 \;\; 1 \,|\, 2 \;\; 3 \;\; 4 \;\; 6 \,|\, 8 \;\; 9 \;\; 5$$

This operator seems particularly well adapted because it introduces the least possible perturbation in the relative positions of the cities by preserving as far as possible the circular order of the cities. Other types of crossovers belong to this class which is called "reordering operators": see Goldberg 1989 (pp. 166-179).

The problem of creating meaningless solutions arises also with mutation. That is why it is replaced by *inversion* in the application to the TSP. Starting from a chromosome $A$, two sites are selected at random, say the 3rd and 6th positions, and the order of the cities between the two selected positions is reversed:

$$A'': 3\ \ 5\,|\,4\ \ 2\ \ 1\ \ 7\,|\,8\ \ 6\ \ 9$$

Applying these special operators as they are described above does not seem to allow to treat successfully problems of a reasonable size. There is a need for crossover operators that do not blindly mix pairs of tours but use problem specific knowledge and local optimization in the crossover operation. This is the case in the algorithm proposed by Grefenstette *et al.* (1985) whose results can compete with those obtained by SA on 200-city problems. Other approaches also combining local search with genetic search are reported to yield near optimal results on the Padberg 532-city problem (see the references in Mathias and Whitley 1992). This tends to show that obtaining good results with GA on combinatorial optimization problems often requires a rather sophisticated interpretation of the basic scheme.

## 7 More on GA

### 7.1 Overview of the theory

A major piece of theory about GA remains the book by Holland (1975). However, GA in general and in that book in particular are not primarily designed for function optimization but as models of efficient adaptive behaviour. Hence, it is no wonder that the results are not concerned with convergence to a global optimum as for SA but with optimal or near-optimal sequences of decisions in the context of an unknown and uncertain environment. This distinction and its consequences are emphasized in a very convincing manner in De Jong (1992).

Holland's theory essentially deals with the notion of *schema*. A schema is a family of strings (i. e. coded solutions) that share the same values at certain positions while they take any value everywhere else. For instance, in a space

$X$ where the solutions are encoded into 9-character bitstrings, $*10*1*0**$ denotes the set of strings whose 2nd, 3rd, 5th and 7th coordinates are equal to 1, 0, 1 and 0 respectively. The "wild card" character "$*$" indicates that any value is allowed, either 0 or 1, at its position in the string. In a space $X$ of $K$-character bitstrings there are $2^K$ strings (chromosomes) and $3^K$ schemata. Each string belongs to $2^K$ different schemata and in a population of $N$ strings, $N2^K$ (non necessarily different) schemata are represented.

To a schema $H$ are associated three quantities:

- The schema *order* $O(H)$ which is the number of fixed values in $H$ (i. e. the number of "non-wild card" characters).
- The schema *defining length* $\delta(H)$ which is the distance between the first and last fixed positions in $H$.
- The schema *value* $F(H)$ which is the average fitness on the set of solutions belonging to schema $H$.

The so-called fundamental theorem of GA is concerned with the evolution with time $t$ of the number $m(H,t)$ of solutions belonging to a given schema $H$. This result is based on 1-point crossover, i. e. a crossover operator which exchanges the substrings situated after the single cutting point. The theorem establishes that the following inequality approximately holds:

$$m(H,t+1) \geq m(H,t)\frac{F(H)}{\overline{F}}[1-\chi\frac{\delta(H)}{K-1}-\mu O(H)] \qquad (1)$$

where $\overline{F}$ is the average fitness of the population, $\chi$ is the probability of crossover, and $\mu$ is the probability of mutation ($2M = N$ in the parameter values introduced in Section 6.1).

This result is usually interpreted as follows. The schemata with above average fitness, small order and short defining-length will proliferate. For such schemata, the factor multiplying $m(H,t)$ in (1) has the best chances to be larger than 1. Formula (1) is rather intuitive: schemata with large defining length are more easily disrupted by 1-point crossover while schemata of high order are destroyed by mutation.

Note that as long as the factor multiplying $m(H,t)$ in (1) remains larger than 1, the proliferation of $H$ is exponential. Is it a desirable feature? Well, it is certainly not bad that schemata with above average fitness will proliferate in the next generation but this does not guarantee that globally optimal solutions will ever be reached. In order to understand the theory about GA one must remember that their original aim is not optimization but that GA are models of adaptive systems. A GA produces sequential decisions in a decision process where uncertainty is present in the form of lack of a priori knowledge, noisy feedback and even time-varying payoff function (indeed in natural evolution there is no global objective function to optimize, but locally, one can define what is good and what is bad). The objective of the whole decision process is to maximize the overall expected gain (or minimize expected loss). Hence, the striking difference with function optimization is that what is important here is the evolution *process* of the population (the trajectories of the set of initial individuals) and not the presence or absence "at the end" of optimal or near-optimal individuals.

Replacing GA in the framework of sequential decision theory and looking there at sequential games with uncertainty as the $k$-armed bandit (the well-known slot machine but with $k$-arms each of them characterized by a different unknown probability distribution of payoffs), one can define an optimal strategy that minimizes the expected losses in a given number of trials. In order to allocate trials optimally, one has to give slightly more than exponentially increasing number of trials to the best observed arm. Transposing this to GA (which was done by Holland) shows that those are near-optimal strategies as they allocate exponentially growing representation in the population to the good schemata.

In the above, we viewed GA as processing schemata. How efficient are they in doing that job? Computations due to Holland evaluate to approximately $N^3$ the number of schemata usefully processed (i. e. the number of above average, low order, short defining length schemata that are not destroyed) in a population of $N$ strings at each step. This result is known under the name of "implicit parallelism" as $N^3$ objects are processed for the price of $N$.

The above considerations have some consequences for the use of GA as function optimizers. First, the coding of the solutions should be as much as possible meaningful, i. e. schemata should actually be associated with features of the solutions which make them good according to the objective function. In addition, these good schemata should have characteristics that allow for efficient reproduction, i. e. small order and short defining length. This is for instance the case with the simple example of Section 6.2: the schema $1 * * * *$ is a high quality one (it is the best of order 1 and the best of defining length 0). Finally, a good or optimal string should be characterized by its belonging to a large number of good schemata ("building block hypothesis").

A second consequence is that using GA as function optimizers implies some adaptation of the original scheme. An elementary modification (also of application with SA) is to keep track of the best solution met. One can also force the best solution met to remain in the population. Another problem with original GA used as function optimizers is that they do in general succeed in locating potentially optimal regions (probably by means of good schemata) but are less efficient for the final step of optimization which consists of locating more precisely the optimal solution. Roughly speaking, if the fitness varies in $[0, 100]$, population is likely to stabilize quickly in the range $[99, 100]$ but GA don't make much difference between solutions that differ by one unit only. A solution to this problem consists of adapting the selection mechanism by dynamically rescaling the fitness function or alternatively using the *ranking* of the solutions in a population according to their fitness rather than fitness itself.

A third possible adaptation of original GA to optimization problems is to incorporate specific knowledge on the problem: see for instance Grefenstette (1985) for the TSP.

In conclusion, the same statement about theoretical results is valid for GA as for SA. They are a source of inspiration for some tactical choices that have to be made but do not describe the behaviour of the actually implemented optimization algorithms nor do they help understanding what really makes them work (or fail). Note finally that some work is currently being done for establishing convergence results for GA in much the same spirit as for SA, i. e. using Markov chain theory (see e. g. Davis and Principe 1991).

## 7.2 Beyond simple GA

There are several more advanced techniques in the domain of genetic search. Let us mention for instance the "*niche* and *speciation*" techniques which aim at maintaining some part of the population (a species) in certain regions ("niche"). This is done in order to get a better idea of the general topography of the search space and to locate the different regions with "high peaks". In certain circumstances, high quality local maxima can be preferred to global maxima because of additional characteristics they could have due e. g. to their location in search space. The technique consists in only allowing for the replacement of a solution (parent) by another (child) when the new solution is similar (i. e. close in the search space) to the replaced one (for detail see Goldberg 1989 and references therein).

Another interesting development is the parallelization of GA. We will not enter into more detail on this subject: the interested reader is referred to Goldberg (1989) for an overview and to Mühlenbein (1989) for a seemingly promising approach.

Until the end of this section, we give a short introduction to the advanced topic of genetic-based machine learning and to the most common systems in the field, the *classifier systems*. Those are essentially algorithms which maintain and let evolve a population of rules called classifiers. Hence, it can be considered an expert system with learning capabilities. Among the few applications of such systems to optimization, let us mention scheduling (Hilliard *et al.* 1987, Bouffouix 1990). To be more concrete, we present the general structure of a classifier system in the context of scheduling. We follow Bouffouix' work.

The classifier system works in interaction with an environment which is in this example a scheduling problem of a certain type and an elementary scheduler. By this, we mean an algorithm able to produce a schedule provided a certain number of conflicts have been solved by an external instance. In Bouffouix' work, the scheduling problem is of the Job shop type. A list of tasks composed of different operations are to be executed on different machines. Each operation must be done on a specific unique machine and the processing order of the operations within a task is fixed. What remains to be decided is the order and time of processing of each operation on its assigned machine.

Suppose our elementary scheduler tries to schedule the operations on each machine at their earliest possible processing date (which takes into account the precedence constraints between operations within a task and possibly ready dates associated to the tasks). Then, we will be confronted with *conflicts* between operations that compete for the same resource. At each elementary conflict, i. e. a conflict involving two operations, a message describing the conflict is sent to the classifier system which sends back a decision giving priority to one of the two competing operations. At the end of the process, when a whole schedule has been completed, the environment sends an evaluation of the schedule (e. g. the makespan) to the classifier system. The interactions of the system with its environment are summarized in Figure 4.

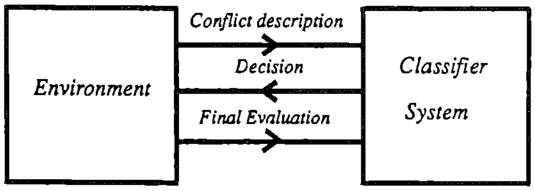

Figure 4: Interactions between a classifier system and its environment.

The internal structure of the classifier system is illustrated in Figure 5. The rule base (i. e. the set of classifiers) is the central part of the classifier system. The rules usually are of the type:

$$\text{If } < \text{condition} > \text{ then } < \text{action} >$$

To each rule is assigned an evaluation, called fitness of the rule, which evolves with time. When a message describing a conflict comes to the system, the "matcher" looks in the set of rules for the rules whose condition matches the conflict description. The "selector" chooses one of the matched rules according to a probability which is a non decreasing function of the rule fitness. The "action" of the selected rule is the decision which is communicated to

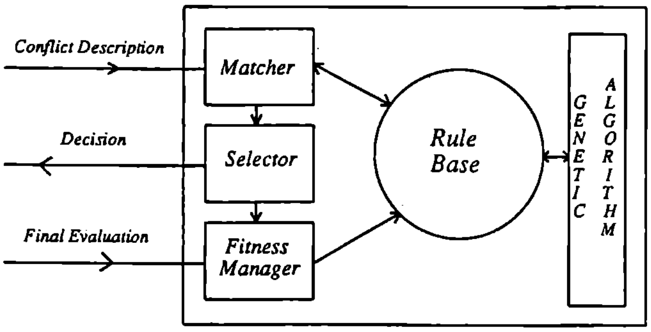

Figure 5: Internal structure of the classifier system.

the environment: in our scheduling problem, it consists in choosing one of the two operations in conflict to be scheduled first. The rule fitness evolves according to a complex mechanism of taxes and rewards. The fitness of all rules whose condition matches a message is decreased by a "tax". The rule which is finally selected pays a tax for being selected. When a schedule has been completed, the fitnesses of all rules which have been selected for solving conflicts for this schedule are increased or decreased according to the overall evaluation sent by the environment and all rules in the rule base pay a tax "for existing". At fixed periods, i. e. after a fixed number of completed schedules, a genetic algorithm is used to produce a new generation of rules from the current population. This genetic algorithm works as described in Section 6 as the rules are usually coded by means of bitstrings. The rule fitness is used in the selection process.

This type of complex system is clearly intellectually appealing but the difficulties for making it work properly are in proportion of the ambition. This is due mainly to the large number of parameters to be tuned and the collective character of the rule evaluation (reward or penalty after a complete schedule). Another weakness is the difficulty of dealing with rules that are other than elementary, i. e. with complex conditions taking into account a non-trivial description of a conflict context. Finally, the rules produced through the genetic algorithm action on the rule base are difficult to interpret: one is generally unable to give them a sensible meaning in the context of the problem at hand. This leaves the door open for the suspicion that the emerging rules are closely dependent on the problem instance and will change when another instance will be submitted. The above problems will probably not be solved easily except perhaps for very specific problems (see e. g. Goldberg 1989 for a list of applications in various fields). An interesting result of the scheduling classifier system described above is that it is able to "learn" a definite rule: if the system is rewarded when it behaves for instance like the SPT rule (Short Processing Time first), one usually finds this rule in the rule base at the end even if it was not there at the start.

# 8 Neural Networks (NN) for combinatorial optimization

## 8.1 Discrete Hopfield nets

General neural networks (NN) used in combinatorial optimization are known as Hopfield nets. We first describe their discrete state version, the continuous one being presented in Section 8.3. Consider a graph (or network) whose $N$ vertices are called *neurons* and are associated a state value which is either $-1$ or $+1$ (0 or 1 can also be considered). The state of the whole network which evolves in discrete time $t = 1, 2, \ldots$, is characterized by a *state vector* $\overline{x}(t) = (x_1(t), \ldots, x_N(t))$. Each connection between two neurons (= edge of the graph) is weighted: let $(T_{ij}; i, j = 1, \ldots, N)$ denote the *connection weights matrix.*

The dynamics of evolution of the neural network is given by the following equation

$$x_i(t+1) = \operatorname{sgn}[\sum_{j=1}^{N} T_{ij} x_j(t) - I_i] \tag{2}$$

where $(I_i, i = 1, \ldots, N)$ is a *threshold vector* associated with the neurons and sgn[...] is equal to $+1$ or $-1$ according to the sign of the expression between brackets. The expression (2) receives the following interpretation in the language of NN: a neuron $i$ is in its upper level $(+1)$ if the sum of all "inputs" in $i$, $\sum_j T_{ij} x_j$, passes a certain threshold $I_i$.

There are several modes for updating the state vector $\overline{x}(t)$ when time elapses: the *synchronous* mode where all state variables are updated at the same time $t$ on the basis of the values $\overline{x}(t-1)$, and the *asynchronous* mode where state variables are updated one after the other and the updated values are used as soon as they are available. The asynchronous case has several variants according to the order in which the neurons are considered for being updated: in the order of their labels ($i = 1, 2, \ldots, N$), according to a random permutation, ..., and so on.

The *energy* $E$ of such a system has been defined by the quadratic form

$$E(\overline{x}) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} T_{ij}x_ix_j + \sum_{i=1}^{N} I_ix_i \tag{3}$$

This quantity is related to the updating equation (2): the energy is non increasing along any trajectory $\overline{x}(t)$ if the matrix $T$ has some properties ($T$ is symmetric and its diagonal elements are all 0). The evolution (2) of the system naturally drives it to stable states (i. e. state vectors such that $\overline{x}(t+1) = \overline{x}(t)$) which are *local minima* of $E$. This feature can be exploited for optimization purposes.

## 8.2 Examples

How can we map a combinatorial optimization problem onto a NN as described above? There is no universal method for doing this. We present two examples of problem formulations below.

### 8.2.1 The graph bipartitioning problem

We consider once again the example illustrating SA in Section 2.3. In the NN formulation of this problem, each node is assigned a neuron $i$ whose state $x_i$ is $+1$ or $-1$ according to its belonging to $V_1$ or $V_2$ (respectively). $(V_1, V_2)$ is a bipartition of the nodes. The connection weight $\gamma_{ij}$ of two neurons $i$ and $j$ is equal to 0 if there is no edge between $i$ and $j$ in the graph and to a fixed positive constant if $(i,j)$ belongs to the edge set. Hence, $\gamma_{ij}x_ix_j$ will be 0 if $i$ and $j$ are not linked by an edge. If $(i,j)$ is an edge, then $\gamma_{ij}x_ix_j$ will be positive or negative according to whether $i$ and $j$ belong to the same class of the partition or not. Minimizing $\sum_i \sum_j \gamma_{ij}x_ix_j$ would lead to put all vertices in one of the two subsets $V_1$ or $V_2$. In order to get a balanced partition $|V_1| = |V_2|$, one introduces the term $\frac{A}{2}(\sum_i x_i)^2$ as a penalty in the energy function which finally reads

$$\begin{aligned} E(x) &= -\frac{1}{2}\sum_i \sum_j \gamma_{ij} x_i x_j + \frac{1}{2} A (\sum_i x_i)^2 \\ &= -\frac{1}{2}\sum_i \sum_j (\gamma_{ij} - A) x_i x_j \end{aligned}$$

The complete connection weight is thus

$$T_{ij} = \gamma_{ij} - A$$

The threshold vector is null in this case.

### 8.2.2 The Travelling Salesman Problem (TSP)

We consider a *symmetric* $N$-city TSP which means that the cost matrix $C = (c_{\alpha\beta}; \alpha, \beta \in$ set of $N$ cities) is symmetric. The NN formulation is based on the representation of a tour by a permutation matrix $x = (x_{\alpha i}; \alpha \in$ set of cities, $i = 1, \ldots, N)$ where $x_{\alpha i} = 1$ if city $\alpha$ is the $i$th city which is visited in the tour (w. r. t. an arbitrary starting point) and 0 otherwise. We have $\sum_\alpha x_{\alpha i} = \sum_i x_{\alpha i} = 1$. Note that the representation is badly redundant each tour being represented by $2N$ different matrices (as there are no privileged origin nor orientation).

One neuron is associated to each variable of the matrix. The state of the $N^2$ neurons is described by the matrix $x = (x_{\alpha i})$ whose elements are either 0 or 1. The energy function $E(x)$ is the sum of two types of terms: the cost of the tour $x$ and several penalty terms that tend to force the respect of the constraints

$$E(x) = aF_1 + bF_2 + cF_3 + dL$$

where $a$, $b$, $c$, $d$ are weighting factors (to be adjusted) and

$$\begin{aligned} F_1(x) &= \sum_\alpha \sum_i \sum_{j \neq i} x_{\alpha i} x_{\alpha j} \\ F_2(x) &= \sum_i \sum_\alpha \sum_{\beta \neq \alpha} x_{\alpha i} x_{\beta i} \end{aligned}$$

$$\begin{aligned} F_3(x) &= (\sum_\alpha \sum_i x_{\alpha i} - N)^2 \\ L(x) &= \frac{1}{2} \sum_\alpha \sum_{\beta \neq \alpha} \sum_i c_{\alpha\beta} x_{\alpha i}(x_{\beta,i+1} + x_{\beta,i-1}) \end{aligned}$$

where $c_{\alpha\beta} = c_{\beta\alpha}$ is the cost for going from city $\alpha$ to city $\beta$.

The first three terms $F_1$, $F_2$, and $F_3$, modelize the constraints on the matrix $x$ that will make it a permutation matrix. $F_1$ is minimal ($F_1 = 0$) iff there is at most one "1" on each row and $F_2$ plays a similar role for the columns. $F_3$ is minimal ($F_3 = 0$) iff there are exactly $N$ "1"s in matrix $x$. The last term $L$ expresses the length of a tour (it uses the symmetric character of the TSP we are dealing with).

In expliciting the connection weights in this NN, we get (using $\delta_{ij} = 1$ if $i = j$ and 0 otherwise)

$$\begin{aligned} T_{\alpha i,\beta j} &= -a\delta_{\alpha\beta}(1 - \delta_{ij}) - b\delta_{ij}(1 - \delta_{\alpha\beta}) \\ & \quad -c - d\delta_{\alpha\beta}(\delta_{j,i+1} + \delta_{j,i-1}) \end{aligned}$$

The terms of the threshold matrix $I_{\alpha i}$ are all equal to $cN$. As the state variables in this model take 0-1 values, note that Equation (2) has to be transformed by means of the following affine transformation that maps $\{-1, 1\}$ onto $\{0, 1\}$

$$x = \frac{x' + 1}{2}$$

## 8.3 Continuous state neural networks

The networks that are actually used for efficient optimization are not exactly those described before. The solution space, i. e. the set of state vectors of the above networks, is the set of vertices of a hypercube $\{-1, 1\}^N$ in the $N$-dimensional space (where $N$ is the number of neurons). Due to this discretization, the possibilities of escaping a local minimum are relatively scarce and the trajectories, i. e. the sequences of hypercube vertices visited due to the dynamics of the network, frequently remain stuck in a local minimum.

One way of getting better results is to apply SA to the discrete network: elementary transformations are defined (e. g. flipping a "+1" in "−1" or conversely in the state vector). At each step such a transformation is chosen at random and accepted according to the SA rule, $\Delta F$ being here the difference in the energy function due to the transformation. For applications of this technique, see e. g. Hérault (1991).

Alternatively, another possibility for improving the situation is to allow the state variables to vary continuously in the $[-1,+1]$ (or $[0,1]$) interval. This makes our neurons become *analog* neurons rather than discrete ones. The modelization of a problem remains unchanged but the state variables must now be considered as continuous ones. What is changed is the dynamics that governs the evolution of the network. In analog neurons, one usually distinguishes the potential $u_i$ of the neuron and its output $x_i$ (which is also its state). Time varies continuously and the potential evolves according to the following differential equation

$$\frac{du_i}{dt} = -u_i + \sum_{j=1}^{N} T_{ij}x_j - I_i \tag{4}$$

Note that at equilibrium, i. e. when $\frac{du_i}{dt} = 0$, Equation (4) reduces to Equation (2) if $x_i$ is set equal to $+1$ or $-1$ according to the sign of $u_i$. In general, the relationship between $u_i$ and $x_i$ is the famous *sigmoïd* curve which is classically used to model the relation between stimulus and response in real brain neurons. We have

$$x_i(t) = \tanh(\frac{u_i(t)}{T}) \tag{5}$$

where $T$ determines the shape of this function at the origin. When $T$ tends to 0, $x_i$ tends to be $\mathrm{sgn}(u_i)$. There are different ways of simulating the above dynamics in discrete time (see e. g. Takeda and Goodman 1986). The simplest transition mechanism postulates equilibrium (i. e. $\frac{du_i}{dt} = 0$). Hence (4) yields:

$$u_i(t) = \sum_j T_{ij}x_j(t) - I_i$$

Then, the discrete time version of (5) allows to drop the potential $u_i$:

$$\begin{aligned} x_i(t+1) &= \tanh(\frac{u_i(t)}{T}) \\ &= \tanh[\frac{1}{T}(\sum_j T_{ij}x_j(t) - I_i)] \qquad (6) \end{aligned}$$

Alternatively, (6) can be obtained by importing an approximation technique used in statistical physics, namely the "Mean Field Theory". In this theory, the state $x_i$ of neuron $i$ is approximated by $< x_i >_T$, the mean state value of the neuron when the network has reached its equilibrium for a fixed value of $T$. Statistical mechanics tells us that at equilibrium, the probability of neuron $i$ to be in the state $x_i = \pm 1$ is

$$P(x_i = \pm 1) = \frac{1}{1 + \exp(\mp \frac{2u_i}{T})}$$

Hence $< x_i >_T = P(x_i = +1).1 + P(x_i = -1).(-1) = \tanh(\frac{u_i}{T})$. This proves that the mean field approximation leads to the dynamics described by Equation (6) which can be used either in synchronous or asynchronous mode.

The mean field approach mainly applied to combinatorial optimization by Peterson, seems to be the most promising one. Results of experimentation on the graph bipartitioning problem are reported by Peterson and Anderson (1988). The authors considered graphs with number of vertices ranging from 20 to 2000 and obtain results that are slightly worse than with ordinary SA but in *much* shorter time. The method is not very sensitive to the choice of $A$ and $T$. The imbalance in the final partition is generally small and removed by means of a greedy heuristic (see also Peterson and Söderberg 1989 for a more efficient implementation). This good opinion is confirmed by Hérault (1991) who also experiments with "mean field annealing", a method which consists in slowly lowering $T$ in Equation (6) during the evolution. This last method gives quicker results of a slightly worse quality.

The TSP was also investigated by Peterson through its mean field approach. The results were found comparable to (slightly worse than) those obtained by SA on problem sizes up to 200 cities (see Peterson and Söderberg 1989).

## 8.4 Bibliographic and historic note

The main application of NN is not optimization. NN were originally designed for simulating the brain behaviour. A pioneering work on the potentialities of formal NN is due to McCulloch and Pitts (1943) who show that any logic function can be realized by a NN. The main technical interest of NN is their ability to process information (input signal) with a high degree of parallelization, the computations being distributed throughout the structure. In recent years, NN have been increasingly used for dealing with classification problems (in the wide sense: pattern recognition, vision, voice recognition, ...) where their learning capabilities as well as their computing power (due to parallelism) are determinant features. A general and recent reference on NN is Khanna 1989.

Optimization has been up to now a marginal topic in NN theory: see Másson and Wang (1990) and Chee-Kit Looi (1992) for recent reviews on this subfield. The first cited is specially illuminating. The conception of NN designed for optimization is due to Hopfield (Hopfield and Tank 1985). These are particular cases of *recursive* networks: the state of the network at time $t+1$ is determined by its state at time $t$. For competitive optimization by means of NN, it seems necessary to modify the original dynamics of Hopfield nets as outlined in Section 8.3. A major trend consists in considering stochastic evolutions (among which are the so-called "Boltzmann machines", see Aarts and Korst 1989 and Másson and Wang 1990). SA applied to the discrete Hopfield net as well as the mean field theory (with or without SA) are relevant to this trend (see Peterson and Anderson 1987, 1988, Peterson and Söderberg 1989, Hérault 1991). Stochastic evolution makes NN relevant to statistical mechanics theories: Hopfield nets with stochastic evolution are known in physics as spin glasses (Ising or Potts spin glasses according to the discrete or continuous character of the state space). Theoretical results about the evolution of such systems can be found e. g. in Mézard *et al.* 1987.

Let us finally mention a *particular* NN approach for the euclidean TSP: it is known under the name of "elastic net algorithm" (Durbin and Willshaw 1987, Fort 1988, Hueter 1988, Angéniol *et al.* 1988) and is implemented on another type of NN, the Kohonen NN. Though it seems relatively competitive for the TSP, we did not present it in this section as the present paper is devoted to *general* heuristics.

## 9 Conclusion

In the conclusions of this long tutorial, I will distinguish the viewpoints of the practitioner and the theoretician. From a practical viewpoint, the only concern is the effectiveness of the method used for solving a problem. According to the effectiveness criterion, no universal hierarchy can be made neither between the four heuristics we considered nor between these and other methods. In front of a specific problem, it is not easy for the practitioner to choose an algorithm mainly because of lack of complete, pertinent and objective information on the available solutions. In the case of difficult problems (let say NP-hard optimization problems) one could consider the use of one of the above heuristics from the start or for improving an existing specific heuristic (specially if this is of descent type). It is however always advisable to inform oneself on whether good specific heuristics do exist for the particular problem at hand (as the Kernighan-Lin algorithm for the graph bipartitioning problem: see Johnson 1989). If the decision of implementing a general heuristic is made, I would advocate to begin with a simple one like SA or TS and then turn if necessary to the more complicated GA or NN or to other methods. This is because the time needed to implement SA or TS seems attractively short. For choosing further between SA and TS, let me summarize my opinion on how they compare in the following statements: SA and TS often yield solutions of similar quality; TS is in general much faster than SA; TS involves more tactical choices and hence needs more time to be implemented and adequately tuned. The choice will depend on the relative importance of computing time and implementation effort. So far for the first phase of the implementation of a heuristic. In a second phase, one can try to refine the first implementation if the balance between the expected benefits and the additional research is favourable to more effort. In this case, many possibilities are open. If one remains in the domain of general heuristics, a good and not too expensive idea is to combine good heuristic principles like in Liégeois *et al.* (1992).

Finally, we turn to the theoretical viewpoint. There, the challenge is to explain why the heuristics do work or more precisely for what kind of problems (for which geometry of the search space) our "general" heuristics are efficient. If this can be understood, the way will be open to the design of new general heuristics adapted to families of problems for which the present heuristics are not effective.

## 10 References

Aarts, E. H. L., and J. Korst (1989). *Simulated Annealing and Boltzmann Machines.* Wiley, New York.

Aarts, E. H. L., and P. J. M. van Laarhoven (1985). Statistical cooling: a general approach to combinatorial optimization problems *Philips J. of Research*, 40, pp. 193-226.

Angéniol, B., G. de La Croix Vaubois and J. Le Texier (1988). Self-organizing feature maps and the travelling salesman problem. *Neural Networks*, 1, pp. 289-293.

Azencott, R. (Ed.) (1992). *Simulated Annealing. Parallelization Techniques.* John Wiley, New York.

Belew, R. K., and L. B. Booker (Eds.) (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann Publ., San Mateo, U.S.A.

Bouffouix, S. (1990). *Contribution à l'ordonnancement de production: apport des algorithmes génétiques et des systèmes à apprentissage.* Travail de fin d'études, Faculté des Sciences Appliquées, Université Libre de Bruxelles, Belgium.

Cerny, V. (1985). A thermodynamical approach to the travelling salesman problem: an efficient simulated algorithm. *J. Optim. Theory Appl.* 45, pp. 41-51.

Chams, M., A. Hertz, and D. de Werra (1987). Some experiments with simulated annealing for coloring graphs. *Eur. J. Op. Res.*, 32, pp. 260-266.

Chee-Kit Looi (1992). Neural network methods in combinatorial optimization. *Computers Ops. Res.*, 19, 3/4, pp. 191-208.

Collins, N. E., R. W. Eglese and B. L. Golden (1988). Simulated annealing: an annotated bibliography. Report No. 88-019, College of Business and Management, University of Maryland, College Park, Md.

Davis, T. E., and J. C. Principe (1991). A simulated annealing like convergence theory for the simple genetic algorithm. In Belew and Booker 1991, pp. 174-181.

De Jong, K. A. (1975). *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* Doctoral Thesis, University of Michigan, Ann Arbor, U.S.A.

De Jong, K. A. (1992). Are genetic algorithms function optimizers? In Männer and Manderick 1992, pp. 3-14.

de Werra, D. (1988). Communication at a Conference on Tabu Search. Vrije Universiteit Brussel, Brussels, December 1988, Unpublished.

Durbin, R., and D. Willshaw (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326, pp. 689-691.

Fort, J. C. (1988). Solving a combinatorial problem via self-organizing process: An application of the Kohonen algorithm to the travelling salesman problem. *Biological Cybernetics*, pp. 33-40.

Geman, S., and D. Geman (1984). Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. *IEEE Proc. Pattern Analysis and Machine Intelligence.* PAMI-6, pp. 721-741.

Gidas, B. (1985). Non-stationary Markov chains and convergence of the annealing algorithm. *J. Statis. Phys.* 39, pp. 73-131.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 5, pp. 533-549.

Glover, F., M. Laguna, D. de Werra, and E. Taillard (1992). *Tabu Search.* Annals of Operations Research, Vol. 41, J. C. Baltzer Publ., Basel, Switzerland.

Glover, F., and H. J. Greenberg (1989). New approaches for heuristic search: a bilateral linkage with artificial intelligence. *Eur. J. Op. Res.*, 39, pp. 119-130.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, Mass.

Goldberg, D. E., and R. Lingle (1985). Alleles, loci and the travelling salesman problem. In Grefenstette 1985, pp. 154-159.

Green, J. M., and K. J. Supowit (1986). Simulated annealing without rejected moves. *IEEE Trans. Computer-Aided Design.* CAD-5, pp. 221-228.

Grefenstette, J. J. (Ed.) (1985). *Proceedings of an International Confer-*

*ence on Genetic Algorithms and their Applications.* Hillsdale, NJ, Lawrence Erlbaum Associates, U.S.A.

Grefenstette, J. J. (Ed.) (1987). *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms.* Hillsdale, NJ, Lawrence Erlbaum Associates, U.S.A.

Grefenstette, J. J., R. Gopal, B. J. Rosmalta, and D. Van Gucht (1985). Genetic algorithms for the travelling salesman problem. In Grefenstette 1985, pp. 160-168.

Hajek, B. (1988). Cooling schedules for optimal annealing. *Math. Oper. Res.*, 13, pp. 311-329.

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. Presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy.

Hérault, L. (1991). *Réseaux de neurones récursifs pour l'optimisation combinatoire.* Thèse de Doctorat, Institut National Polytechnique de Grenoble, France.

Hertz, A., and D. de Werra (1987). Using tabu search techniques for graph coloring. *Computing*, 29, pp. 345-351.

Hilliard, M. R., G. E. Liepins, M. Palmer, M. Morrow, and J. Richardson (1987). A classifier based system for discovering scheduling heuristics. In Grefenstette 1987, pp. 231-235.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, U.S.A.

Hopfield, J. J., and D. W. Tank (1985). "Neural" computation of decisions in optimization problems. *Biol. Cybernetics.* 52, pp. 141-152.

Huang, M. D., F. Romeo and A. Sangiovanni-Vincentelli (1986). An efficient general cooling schedule for simulated annealing. *Proc. IEEE Int. Conf. on CAD (ICCAD 86)*, pp. 381-384, Santa Clara, Calif.

Hueter, G. J. (1988). Solution of the travelling salesman problem with an adaptive ring. *IEEE International Conference on Neural Networks*, San Diego, CA, July 24-27.

Johnson, D. S., C. R. Aragon, L. A. McGeoch and C. Schevon (1989).

Optimization by simulated annealing: an experimental evaluation - Part I (Graph partitioning). *Opns. Res.*, 37, 6, pp. 865-892.

Johnson, D. S., C. R. Aragon, L. A. McGeoch and C. Schevon (1991). Optimization by simulated annealing: an experimental evaluation - Part II (Graph coloring and number partitioning). *Opns. Res.*, 39, 3, pp. 378-406.

Johnson, D. S., C. R. Aragon, L. A. McGeoch and C. Schevon (1992). Optimization by simulated annealing: an experimental evaluation - Part III (The travelling salesman problem). To appear in *Opns. Res.*

Khanna, T. (1989). *Foundations of Neural Networks*. Addison-Wesley, Reading, Mass.

Kirkpatrick, S., C. D. Gelatt, Jr. and M. P. Vecchi (1983). Optimization by simulated annealing. *Science*, 220, pp. 671-680.

Lam, J., and J. M. Delosme (1986). Logic minimization using simulated annealing. *Proc. IEEE Int. Conf. on CAD (ICCAD 86)*, pp. 348-351, Santa Clara, Calif.

Liégeois, B., M. Pirlot, J. Teghem, E. Trauwaert, and D. Tuyttens (1992). Homogeneous grouping of nuclear fuel cans through simulated annealing and tabu search. Submitted.

Lundy, M. and A. Mees (1986). Convergence of an annealing algorithm, *Math. Prog.*, 34, pp. 111-124.

Männer, R., and B. Manderick (1992). *Parallel Problem Solving from Nature*, 2. North-Holland, Amsterdam, The Netherlands.

Másson, E., and Y.-J. Wang (1990). Introduction to computation and learning in artificial neural networks. *Eur. J. Op. Res.*, 47, pp. 1-28.

Mathias, K., and D. Whitley (1992). Genetic operators, the fitness landscape and the travelling salesman problem. In Männer and Manderick 1992, pp. 219-228.

McCulloch, W. S., and W. A. Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5, pp. 115-133.

Mézard, M., G. Parisi, and M. A. Virasoro (1987). *Spin glass theory and beyond.* Lecture Notes in Physics 9, World Scientific, NJ, U.S.A.

Mühlenbein, H. (1989). Parallel genetic algorithms, population genetics and combinatorial optimization. In Schaffer 1989, pp. 416-421.

Mitra, D., F. Romeo and A. L. Sangiovanni-Vincentelli (1986). Convergence and finite-time behavior of Simulated Annealing. *Adv. Appl. Prob.*, 18, pp. 747-771.

Otten R. H. J. M. and L. P. P. P. van Ginneken (1984). Floorplan design using Simulated Annealing. *Proc. IEEE Int. Conference on Computer-Aided Design*, Santa Clara, November 1984, pp. 96-98.

Pedersen, J. M. (1990). *Simulated Annealing and Finite-Time Thermodynamics.* Ph. D. Thesis, University of Copenhagen, Denmark.

Peterson, C., and J. R. Anderson (1987). A mean field theory learning algorithm for neural networks. *Complex Systems*, 1, pp. 995 sq.

Peterson, C., and J. R. Anderson (1988). Neural networks and NP-complete optimization problems: a performance study on the graph-bisection problem. *Complex Systems*, 2, pp. 59 sq.

Peterson, C., and B. Söderberg (1989). A new method for mapping optimization problems onto neural networks. *Int. J. Neural Syst.*, 1, pp. 3-22.

Schaffer, F. D. (Ed.) (1989). *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann Publ. San Mateo, U.S.A.

Schwefel, H. P. (Ed.) (1991). *Parallel Problem Solving from Nature*, 1, Lecture Notes in Computer Science, 496, Springer, Berlin, Germany.

Siarry, J., and G. Dreyfus (1989). *La méthode du recuit simulé.* Paris, IDSET.

Takeda, M., and J. W. Goodman (1986). Neural networks for computation: number representations and programming complexity. *Applied Optics*, 25, 18, pp. 3033-3046.

van Laarhoven, P. J. M., and E. H. L. Aarts (1987). *Simulated annealing: Theory and Practice*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Vidal, R. V. V. (Ed.) (1993). *Applied Simulated Annealing.* Lecture Notes in Economics and Mathematical Systems, Springer, Berlin, to appear.

Yamada, T., and R. Nakano (1992). A genetic algorithm applicable to large-scale Job-shop problems. In Männer and Manderick 1992, pp. 281-290.