Control and Cybernetics

vol. **25** (1996) No. 1

Simulated annealing: a tutorial

by

Marc Pirlot* and **René Victor Valqui Vidal****

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

**Institute of Mathematical Modelling,
Technical University of Denmark,
DK-2800 Lyngby, Denmark

Abstract: The main purpose of this paper is to provide an overview of the ideas behind simulated annealing algorithms and to outline guidelines for the construction of these algorithms. For the sake of concreteness a simple example will be discussed to illustrate both the process of designing a simulated annealing algorithm and the achieved results. In addition, some current research problems and current perspectives on the technique will be surveyed. Finally, an outline of applications will be presented.

Keywords: Combinatorial optimization, simulated annealing, applications.

1. Introduction

Simulated annealing (SA) is one of the recent meta-heuristic techniques developed to solve combinatorial optimization problems. These problems can be formulated as follows:

Consider a finite **configuration space** (space of configurations or solution space) $S = \{x | x = (x_1, x_2, \cdots, x_m)\}$, where m is called the dimension of the space, and a **cost function** $C : S \rightarrow R$ which assigns a real number to each configuration; to be specific, in a minimization problem, we want to find a configuration $x^* \in S$, so that $\forall y \in S, C(x^*) \leq C(y)$. Maximization problems are similarly defined.

Interest in SA is intense because few important combinatorial optimization problems can be solved exactly in a reasonable computer time. Most of these problems arising in practice are **NP-complete**: all known techniques for obtaining an exact solution require an exponentially increasing number of steps as the problem becomes larger. Therefore, emphasis has been directed toward heuristic techniques, as SA, for solving these problems. The difference between an **heuristic approach** and an (exact) **algorithm** is that the first is not guaranteed to get an optimum solution; it is designed to give an acceptable solution in reasonable time. In practice, however, the terms heuristic and algorithms are often used interchangeably. In addition, SA is not an algorithm in the sense that it gives a mechanical sequence of computations to solve a specific problem, e.g., in the sense that the simplex method is an algorithm to solve linear programming problems. Rather, SA is a **strategy** for solving combinatorial optimization problems. Thus, the general statement about SA leaves several decisions which have to be made in order to implement the method for a particular problem.

There has been an enormous amount of interest in the application of the SA approach to combinatorial optimization problems following the relatively recent work of Kirkpatrick et al. (1983) and Černy (1985). This field has been the object of intensive study, in what concerns theoretical, applied and practical aspects, by mathematicians, statisticians, physicists, engineers, computer scientists and operations researchers. Extensive bibliographies can be found in Collins et al.(1988), van Laarhoven and Aarts (1987), and the recent paper of Koulamas et al.(1994). Several books have also been devoted to SA: van Laarhoven (1988), Aarts and Korst (1989), Siarry and Dreyfus (1989), Otten and van Ginneken (1990), Azencott (1992), and Vidal (1993), where extensive references are also given.

The main ideas behind SA can be introduced in three different ways. First, by interpreting SA, like many other randomized algorithms, in terms of **stochastic automata** with or without learning capabilities, see further Shragowitz and Lin (1990). This approach is not very well-known.

Second, by following the analogy between finding minimum energy states in a physical system and finding minimum cost configurations in a combinatorial optimization problem. This is the original approach dated back to the work of Metropolis et al.(1953). To understand why such a physics problem is of interest, consider how to coerce a solid into a low energy state. A low energy state usually means a highly ordered state, such as a crystal lattice; a relevant example here is the need to grow silicon in the form of highly ordered, defect-free crystals for use in semi-conductor manufacturing. To accomplish this, the material is **annealed**, i.e. heated to a temperature that permits many atomic rearrangements, then cooled carefully, slowly, until the material "freezes" into a regular good crystal. SA techniques use an analogous set of "controlled cooling" operations for non-physical optimization problems, in effect transforming a poor, unordered solution into a highly optimized, desirable solution. Thus, simulated annealing offers an appealing physical analogy for the solution of optimization problems, and more importantly, the potential to reshape mathematical insights from the domain of physics into insights for combinatorial optimization problems. Most of the books mentioned above use this analogy in the presentation and design of the SA algorithm, and in the development of theoretical results that insure the convergence of the SA algorithm to the optimal solution. This analogy is also used by Lester Ingber in his paper, dealing with the adaptive simulated annealing software, included in this special issue.

Third, by considering SA as one of the new **meta-heuristic** methods for handling complex combinatorial optimization problems (four other ones are: genetic algorithms, neural networks, tabu search and target analysis), see further Glover and Greenberg (1989) and Pirlot (1993). This algorithmic conceptualization is the prevailing one in the fields of mathematical programming and operations research, see for instance Johnson et al. (1989, 1991). This algorithmic approach is the one we will use in this paper and is the one used in most of the papers of this special issue of *Control and Cybernetics.*

Anyone considering the use of SA today has access to a wide range of references covering both theoretical and practical aspects of this technique. This paper aims to give some fundamental guidelines to the technique presenting enough basic material to enable a beginner to get started. Other excellent tutorials have been published, see for instance Eglese et al. (1990) and Dowsland (1993), where more emphasis is given to theoretical aspects. Here, we will be focusing primarily in design, implementation and practical aspects of this technique.

In Section 2, the basic method will be presented. This will be done by showing that SA is a (stochastic) modification of the well-known local search algorithm, also known as steepest descent or greedy algorithm.

In Section 3, guidelines for the implementation of a SA algorithm are outlined. Here, we will emphasize that to solve a particular combinatorial optimization problem by the SA technique, a number of decisions have to be made. These decisions are usually divided in two groups. First, the **generic** decisions, usually denominated as the annealing or cooling scheme (schedule, process). Secondly, the **problem-specific** decisions which are closely related to the actual problem to be solved.

For the sake of concreteness, a simple example will be presented to illustrate the main points of the implementation process. This is done in Section 4, where a family of problems related to partitions of a set with certain specific properties are discussed.

In Section 5, we will present some enhancements and modifications to the original algorithm based on our own and others published experiences. These are rather important pieces of information because in practice the final design and implementation of SA demands a lot of computer experimentation and evaluation of alternative decisions to a given strategy.

SA is a powerful stochastic search technique applicable to a wide range of problems which occur in a variety of disciplines. These include mathematics, condensed matter physics, engineering problems, mathematical programming, statistics, operations research, computer sciences, etc. In Section 6, we will outline some applications of the SA algorithm specially focusing on the fields of mathematical programming and operations research.

Finally, the conclusions will be presented in Section 7.

2. Basic approaches

In this section we will show that the design process of a SA algorithm is easy and it can be done very fast. In addition, SA is also a **transparent** approach, i.e. non-specialists can easily understand its principles. Probably, the easiest way of introducing SA is by showing that it is a stochastic modification of the traditional local search heuristic.

2.1. Local search heuristics

From now on, we consider a minimization problem; of course maximization problems are treated similarly. A local search strategy begins from an initial solution $x_1 \in S$, and at each iteration i, a new solution x_{i+1} is chosen in the neighborhood $N(x_i)$ of the current solution x_i. To each $x \in S$, a subset $N(x) \subseteq S$ is defined as the neighborhood of x. For instance, if S is a set of binary vectors, then a neighborhood $N(x)$ of x may be the set of all solutions $x \in S$ that can be obtained by swapping a single coordinate from 0 to 1 or conversely. We assume that $x \notin N(x)$, $\forall x \in S$, i.e. no solution is a neighbor of itself. The evolution of the actual solution $x_i, i = 1, 2, \ldots$ depicts a trajectory in S.

The best known local search strategy is the so-called **steepest descent** algorithm. This is shown in Table 1. Here, a new solution x_{i+1} is found by picking the best one in $N(x_i)$, i.e. a solution $x_{i+1} \in N(x_i)$ so that $C(x_{i+1}) \leq C(x), \forall x \in N(x_i)$. Then, x_{i+1} becomes the next actual solution if it is not worse than x_i, i.e. $C(x_{i+1}) \leq C(x_i)$.

```
Procedure Descent algorithm
  Begin
      Initialize $(x_i, i = 1)$
      Repeat
          Find best $\overline{x} \in N(x_i)$ ;
          Calculate $\Delta C = C(x_i) - C(\overline{x})$
          if $\Delta C \leq 0$ then $x_i = \overline{x}$
          else : stop.
  End
```

Table 1. A local search strategy.

This strategy seems reasonable, is simple to implement and quick to execute, but it has a serious problem: it is easily trapped in local minima, solutions that look good in some small neighborhood of the total cost surface but are not necessarily the global optimum. Standard iterative improvement is a downhill only style, each new perturbation moves to a configuration downhill from the previous one, thus, becoming trapped in a local minimum. In practice, one scheme to overcome this is simply to try many random initial solutions, improve each, and use the best configuration found. However, for very large problems, the computational expense is great here, the number of random starts needed to adequately sample the cost surface is unreasonable, and we will have no guarantee of finding a good answer. SA is a local search strategy explicitly designed to avoid the above mentioned situations.

2.2. The SA algorithm

SA offers a strategy very similar to steepest descent, with one major difference: annealing allows perturbations to move uphill in a controlled fashion. A move can now transform a configuration into a **worse** one; it is then possible to jump out of local minima and potentially fall into a more promising downhill trajectory. By following, some uphill moves in a controlled manner, SA offers a way of leaving a local minimum. SA is similar to a random descent method in which one does not search for the best solution in $N(x_i)$, but one simply draws a solution x at random in $N(x_i)$. It differs in that a neighbor giving rise to an increase in the cost function may be accepted. This acceptance will depend on a control parameter (temperature), and the size of the increase of the cost function.

The idea of SA comes from thermodynamics and metallurgy related to physical annealing of a solid. To coerce some material into a low energy state, we heat it, then cool it very slowly allowing it to come to thermal equilibrium at each temperature. Simulating this process is very similar to a combinatorial optimization task. For this physical system the goal is to find some arrangement of atomic particles (a configuration) which minimizes the energy (cost) of the system. The basic demand for simulating this process is the ability to simulate how the system reaches thermodynamic equilibrium at each fixed temperature in the **schedule** of decreasing temperatures used to anneal it.

SA works after the following principle: at the beginning (high "temperature") almost all moves (that is all updating of the current solution by a solution x randomly chosen in its neighborhood) are accepted. This permits the exploration of the solution space. Then gradually, the control parameter (temperature) is decreased which means that the algorithm becomes more and more selective in accepting new solutions. At the end, only the moves that decrease C are accepted.

In SA the probability of accepting a transition which causes an increase, ΔC, in the cost is usually called the **acceptance function** and is set to $exp(-\Delta C/T)$, where T is the control parameter which corresponds to the temperature in the analogy with the physical annealing process. In the SA algorithm, at the beginning, T has a relatively high value, to have a better chance to avoid being prematurely trapped in a local minimum. The control parameter is lowered in steps until it approaches zero. After termination the final "frozen" configuration is taken as the solution of the combinatorial problem at hand. In other words, SA can be conceptualized as a generalization of the local search

```
Procedure Inhomogeneous SA algorithm
  Begin
    Initialize $(k, T_k, i)$;
    Repeat
      Generate configuration $(i \rightarrow j)$;
      Calculate $\Delta C_{ij} = C(i) - C(j)$
      if $\Delta C_{ij} \leq 0$ then $i := j$; else
      if $exp(-\Delta C_{ij}/T_k) >$ random $[0,1]$ then $i := j$;
      $k := k + 1$;
      Update $(T_k)$;
    Until stop;
  End;
```

Table 2. The Inhomogeneous SA Algorithm.

algorithm.

The SA algorithm is illustrated in pseudo-code in Table 2 and Table 3 in its inhomogeneous and homogeneous versions, respectively. These two different versions of the SA algorithms represent two different ways of decrementing the control parameter T (annealing scheme, cooling strategy or annealing schedule):

i. the **inhomogeneous** SA algorithm, where T is decreased after each move and which, therefore, can be described theoretically by a single inhomogeneous Markov chain, and

ii. The **homogeneous** SA algorithm, where T is decreased after a number of moves, L, and which, therefore, can be described by a sequence of homogeneous Markov chains, each generated at a fixed value of T. L is usually denominated as Markov chain length or length of plateau.

The analysis of the mathematical models of the stochastic processes generated by the SA algorithm will provide necessary and sufficient conditions derived to ensure that **asymptotically** the algorithms find a globally optimal solution with probability 1, see further Aarts and Korst (1989), and van Laarhoven and Aarts (1987). Unfortunately, these conditions cannot be satisfied in finite time. Therefore, one has to specify values of both a **finite** number of moves at each value of T, and a **finite** sequence of values of T. This means that in practice the SA algorithm will only generate near-optimal solutions which might be the global one. In addition, the way the temperature is cooled in practice is not in accordance with the hypothesis which guarantee the convergence to an optimal solution. The cooling schemes used in practice are much faster than those required by theory and hence, one may not even guarantee near-optimality. This conflict between theory and pragmatism is always present in any practical application of SA.

```
Procedure Homogeneous SA algorithm
  Begin
    Initialize $(k, T_k, i, L_k)$;
    Repeat
      For $i$ :=1 to $L_k$ do
        Begin
        Generate configuration $(i \rightarrow j)$;
        Calculate $\Delta C_{ij} = C(i) - C(j)$
        if $\Delta C_{ij} \leq 0$ then $i := j$; else
        if $exp(-\Delta C_{ij}/T_k) >$ random $[0,1]$ then $i := j$;
      End;
      $k := k + 1$;
      Update $(L_k)$;
      Update $(T_k)$;
  Until stop;
  End;
```

Table 3. The Homogeneous SA Algorithm.

2.3. Additional remarks

SA algorithms are **not** deterministic and will produce different answers each time they are run, even on the same problem. This is because of the probabilistic nature of choosing moves and accepting uphill moves. In particular, there is **no** guarantee of getting precisely the optimum solution in any annealing algorithm or even of getting the same answer on multiple runs. What SA really offers is **some** probability of getting out of **some** local minima; this is not the same as a guarantee of finding the optimal solution. In practice, it is recommended to select the seed of the random generator in order to be able to make repeatable experiments.

A major open question is to characterize the sorts of problems amenable to annealing. Not all combinatorial optimization problems can be annealed to give satisfactory solutions, e.g. the time taken to get a good solution may prove to be unreasonable. It is widely believed that something about the basic **structure** of a particular configuration space - its landscape of multi-dimensional hills and valleys - determines whether annealing is viable. Consider a mostly flat landscape with numerous, densely packed gopher holes, each of widely varying depth, but some of which lead to the very best solutions. Such a problem is probably impossible to anneal since moves will keep falling into these dead-end holes, and as the temperature cools, it will become impossible to climb out of them, thus trapping the solution in a bad local minimum. Thus, just as some materials resist physical annealing, some problems resist SA. Moreover, there are even applications where SA gives reasonable results but is outperformed my more conventional heuristic approaches. In fact, SA is simply another technique for solving combinatorial optimization problems, superior to some approaches for some problems, inferior to others. But practice has shown the wide applicability of SA, see for instance Vidal (1993).

3. Implementation

To solve a particular combinatorial optimization problem by the SA technique, a number of decisions have to be made in order to design a suitable algorithm. These are presented in Table 4, where these decisions are divided into two groups. First, the **generic** ones that must be made for any implementation of SA. They are usually denominated as the annealing or cooling scheme (schedule, process, etc.). Secondly, the **problem-specific** decisions which are closely related to the actual problem to be solved. The design of a SA algorithm demands both a suitable use of all the knowledge (theoretical and practical) available on the actual problem at hand and a suitably selected set of parameters, the "tuning" of the algorithm.

Decisions	
Generic(Cooling Scheme)	Problems-Specific
•T_o (initial temperature) •L_k (number of iterations, "length of plateau") •T_k (temperature function) • Stopping criteria	•i_o (initial solution) • Neighbor generation • Evaluation of ΔC_{ij}

Table 4. Designing the SA Algorithm.

3.1. Generic decisions

The generic decisions basically involve the cooling schedule, including the initial (T_o) and the final temperature (T_f), and the rate at which it must be reduced (the temperature function). The rate at which the temperature parameter is reduced is vital to the success of any SA algorithm. The most popular and simplest cooling schedule is the so-called **geometric** cooling scheme (Kirkpatrick et al, 1985). This involves a geometric temperature function of the form

$$T_{k+1} = \alpha T_k \quad , \quad 0 < \alpha < 1$$

Practice has shown that relatively high values of α perform best, $0.8 \leq \alpha \leq 0.99$, with a bias to the higher end of the range. This corresponds to slow geometric cooling.

In the implementation of the inhomogeneous SA algorithm, the parameters to be fixed in the cooling scheme are then T_o, T_f and α. In the homogeneous version, we have also to fix the length of the plateau L_k, where the temperature T_k will be kept constant for L_k consecutive steps. Then the temperature will be decreased following the geometric cooling rule. When the homogeneous version is implemented in practice usually the length of the plateau is set to $L_k = L \ , \forall k$.

In the implementation of the homogeneous SA algorithm it is important to be aware that L and α are positively correlated: increasing L or α tend to increase the number of iterations (slow cooling) and should also improve the final solution. When α and/or L are increased beyond a certain point, improvement of C^* becomes so slow that better results are obtained by allocating the available computing time to several shorter runs of the algorithm rather than to a single very long run. For some examples, there is some evidence that for a fixed amount of computing time, there exists an optimal compromise between the number of runs and their lengths.

There is not a single and universal rule for selecting L and α. However, it is usually the case that some preliminary experimentation quickly yields acceptable and robust values.

Another simple and popular cooling scheme, usually used in connection with implementation of the inhomogeneous version of the SA technique, is the so-called **Lundy-Mees** scheme (1986). This cooling scheme is regulated by the formula

$$T_{k+1} = \frac{T_k}{1 + \beta T_k}$$

where β (a small value) is defined as

$$\beta = \frac{T_o - T_f}{M \cdot T_o \cdot T_f}$$

and M is the total number of neighbor generations.

Many others, more complicated, cooling schemes based on the thermodynamic analogy to the annealing process have been suggested in the literature. For a complete review, see Collins et al.(1988). Most of these approaches are based on the idea that quasi-equilibrium in the Markov chain should be reached before temperature is varied and, in addition, the change in temperature should be slow enough to allow to quickly reach a new quasi-equilibrium state for the new temperature.

Empirical evidence suggests that the manner of cooling is not as important as the rate. This means that there are not much to choose from, say, the geometric and the Lundy-Mees scheme, as long as they cool over the same range of temperatures at approximately the same rate. Therefore, based on the experiments reported in the literature, when using SA for a new application it is probably best to start off with one of the simple schedules and only consider the more complicated ones if these fail to provide satisfactory results. In what concerns the fixing of the values of the parameters for the schedule chosen, there is no easy way of achieving this and almost all the successful applications published in the literature state that the best parameters were determined after much experimentation. Note that one of the main assumptions in SA is that the cooling scheme is basically independent of the problem on hand, in some complex situations this might not be true. Therefore, either a modification of the cooling scheme has to be done or another approach has to be selected.

The initial temperature T_o has to be selected high in order to permit bad moves at the initial steps of the algorithm but on the other hand too high values will increase the total computation time. Usually, T_o is generally determined in order that the initial probability for accepting bad moves be approximately equal to a prescribed value χ_o. This can be done by running the algorithm with a tentative initial temperature, the acceptance rate of bad moves is computed and T_o recalculated. Then, the algorithm is restarted with a modified value of T_o and so on until an acceptance rate near χ_o is found. Dekkers and Aarts (1991) have presented a more elaborated way to determine the initial value of T_o which ensures that the algorithm converges faster because it is better adapted to the actual problem at hand.

Theoretically, the temperature should be allowed to decrease to zero before the stopping condition is satisfied, but in practice there is no need to decrease the temperature this far. As the temperature approaches zero the probability of accepting uphill moves will be indistinguishable from zero. Thus, the criterion for stopping can be expressed either in terms of a minimum value of the temperature parameter, or in terms of the "freezing" of the system at the current solution, some rules attempt to define freezing as the number of iterations or temperatures that must have passed without an acceptance. To accelerate the process this condition is sometimes weakened so that the process stops when the proportion of accepted moves drops below a given value. The simplest and most popular rule is to specify the total number of iterations and stop when this number is completed. Obviously, this simple rule has to be carefully tuned with the other parameters to ensure that it corresponds with a sufficiently low temperature to ensure convergence. More specific some stop rules are:

- If C^* has not been improved by at least $\epsilon_1\%$ after K_1 consecutive series of L steps, or
- If the number of accepted moves is less than $\epsilon_2\%$ of L for K_2 consecutive series of L steps.

The values of ϵ_1 and ϵ_2 (usually between 1 and 5 %), and K_1 and K_2 should be fixed by experimentation.

In the experimentation process necessary to fix the parameters it is proved very useful to use the plot of C_k versus k (the number of iterations) as a graphical tool. At the beginning slow improvements are achieved (not necessarily monotonically). Similarly, at the end it is advisable to wait for a clear signal showing that the annealing process has converged.

3.2. Problem-specific decision

Obviously, only some general comments can be made at this point. If for a given combinatorial optimization problem it is possible to design a local search heuristic approach, then, it is rather easy to modify it to a SA algorithm as we have seen in Section 2.

In many applications an initial solution is rather easy to generate. Then, this initial solution can be varied as a parameter in order to generate different solutions. But in some problems it is difficult to find an initial solution because of the fact that many constraints have to be satisfied. In such situations, excellent results have been found by modelling some (difficult) constraints as **soft** constraints, e.g. through penalty terms added to the objective function. Such penalties are functions of the degree of violation of the constraints and vanish when the constraints are satisfied. This modification of the model has an incidence on the solution space as illegal or unfeasible solutions are considered.

Another main reason for using soft constraints is to get simpler or smoother neighborhood structures. If, we restrict to generate only feasible solutions it might be complicated and time consuming to (randomly) generate a feasible neighbor to the actual solution.

In other types of applications where it is possible to achieve some theoretical results about the problem structure, then a neighbor solution can be easily generated having the adequate properties (feasible, near-optimal) and experiences have shown that in such situation the SA algorithm can be considerably improved.

Another important aspect to be taken into consideration is the fact that ΔC has to be calculated at each iteration. It is, therefore, important that the cost function and the neighborhood structure are chosen in such a way that the calculation can be carried out quickly and efficiently. It is often the case, in well-designed SA algorithms that this does not necessitate recalculation of the complete cost function for the new solution, and any shortcuts should be considered when deciding forms of the costs and neighborhoods.

The determination of suitable problem-specific decisions demand a great deal of experimentation and suitable use of the available knowledge about the problem in question. It is in this sense that we can stipulate that the design of a suitable SA algorithm is both an art and hard engineering work. Looking at examples is certainly the best way for getting some feeling in these matters. In the next section, we describe a family of examples and then concentrate on one of them.

4. A family of problems

As we have seen above a main specific decision in building a SA algorithm is to define neighbors in order to get an efficient exploration of the solution space, i.e. to ensure that the "good regions" be visited at reasonable computing cost. In general, the choice of the solution space, objective function and neighborhood structure are interrelated decisions. An important feature of the moves from a solution to a neighbor is to be able to transform any solution into any other one in the solution space in a finite number of steps.

Our aim in the present section is to illustrate these points with some examples. We discuss a family of problems where solutions are partitions of a set with certain specific properties. Two types of transformations suggest themselves: moves of a point from a class of the partition to another one and exchanges of two points belonging to different classes of the partition. A typical example is clustering. Consider a set of vectors in $\mathcal{R}^N$ which describe individuals w.r.t. N characteristics. Clustering is a technique in data analysis which aims at partitioning the individuals in subsets according to their similarities. Usually classes are built which minimize the distances between individuals belonging to the same class and maximize the distances between classes (with an appropriate definition of the word "distance"). The problem of minimizing the sum of intra-class distances even with an *a priori* fixed number of classes, is a difficult one (NP-hard). In this situation one can think of using SA. The search space is the set of all partitions in K classes, K being determined by inspection, intuition or previous knowledge. A neighbor of the current solution may be obtained either by moving an individual from a class to another or exchanging two individuals which belong to different classes (see for instance the paper by Dzemyda in this special issue).

It can happen that the above obvious suggestions are not appropriate or raise some problems. It is the case e.g. in graph coloring which is an important tool for solving time-tabling problems . A solution ("legal" coloring) is a partition of the vertices with no edge linking vertices assigned to the same class. The goal is to find a legal coloring with the minimal number of classes. Moving from a legal coloring to another legal coloring is in general a rather complicated transformation. In Chams, Hertz and de Werra's (1987) application of SA to this problem, the search proceeds in the space of *all* partitions with fixed number of classes by moving one *bad* vertex from a class to another one ("bad vertex" means a vertex which is linked to at least another vertex of the same class). The goal of the search is to minimize the number of illegal edges i.e. the number of edges linking vertices belonging to the same class. If a partition without any illegal edge can be reached then a legal coloring has been found which needs as many colors as there are classes in the partition. The search is then restarted in a space of partitions with fewer classes. The process halts when no legal coloring can be found with the allowed number of colors. Note that more complicated neighborhood structures were experimented with on the same problem by Johnson et al. 1991 (the so-called Kempe chain approach, where moves preserve the legality of the coloring)

4.1. Homogeneous grouping of nuclear fuel cans

Let us now describe in more detail the implementation of a SA algorithm for another partitioning problem which was raised by a company producing nuclear fuel. Full detail about this application can be found in Tuyttens et al. 1994 or Liégeois et al. 1992. The feed material is stored in small cans and the cans must be grouped in batches before being used. Each can is characterised by 7 quantities, its weight and its content in 6 isotopes of an element. The goal is to assign the cans to batches in such a way that the batches are as homogeneous as possible w.r.t. the seven characteristics. The objective function C measures the lack of homogeneity of an assignment. As the range of variation of the weights is not very large, the number of cans in a batch is almost always the same (say 4 or 5) and an assignment (a solution) can be described by a matrix with 5 columns and as many rows as necessary; each row is associated to a batch and contains the identifiers of the cans associated to it (with possibly a dummy can in case the batch comprises only 4 cans).

Starting with an initial assignment, we draw at random two batches and two cans in these batches; if exchanging these cans results in a lower value of the objective function C, we do the exchange. Else we compute $exp[-\Delta C/T]$. If a random number drawn in the $[0,1]$ interval is smaller than this value we accept the move, otherwise we try another exchange. The initial temperature T_0 is chosen in order to accept about 50% of the bad moves (i.e. moves which don't improve C). The temperature is decreased by a factor $\alpha = 0.95$ after each series of L steps. The algorithm is stopped after we have observed 50 consecutive series of L steps with improvement of C by less than 1% in each step.

Except for L the parameters have been set in a fairly standard way. Only L is adjusted according to the size of the problem instance, i.e. the number of cans. In our case we do not select any form of functional dependency linking L and the number of cans; as the goal is to be able to deal with lists of about 150 to 200 cans, we just selected a value of L which yields satisfactory results for problems of this size. A crucial issue is to define what is a "satisfactory" result. In the present application we were lucky enough to benefit from previous work done in the firm. Indeed a local search algorithm of the steepest descent type based on the same neighborhood structure was used in the firm. This algorithm could only deal with lists of up to 50 cans; larger lists resulted in unbearable computing times. As a feasibility test for the SA approach, we began to work on a 50 cans example which could be treated by the local search algorithm. We were readily able to find better results with much less use of computer time. For these preliminary trials we set arbitrarily the value of L to 250 iterations. In our subsequent trials on full-size instances (170 cans) we tried the values $L = 250, 500, 750, 1000$ and obtained results which were considered satisfactory by the managers of the firm as soon as L was as large as 500. In this case, computing times are almost proportional to L.

In the sequel, we tried to find "optimal" values for the parameters. Obviously this is a multicriteria problem (and even an ill-posed one); in particular one has to tradeoff between quality of solution and computing times. Usually there is a point were progress in quality becomes extremely slow and can only be obtained at the expense of heavy computing times. In order to explore the parameter space around the point where the first satisfactory values had been reached, we made further experimentation with all combinations of the following values for the parameters.

$T_0 = 50, 100, 200$(50 is the initially chosen value)
$\alpha = 0.90, 0.95, 0.975$
$L = 250, 500, 750, 1000$

In order to take into account the variability due to the choice of the initial solution, the algorithm has been run with each set of parameter values on the same sample of 100 randomly generated initial solutions. The original set of parameters (with $L = 750$) was found to be the best one; of course slightly better values of C were obtained with L=1000, but the improvement was not judged to be worth the (linear) increase in computing time. The trials made with this parameter setting are summarized in Table 5 by the following quantities: minimal, maximal and mean values of C over 100 runs, standard deviation of C, average computing time in seconds (on a DECstation 5000/133), mean total number of series of L iterations and mean number of series of L iterations at which the best value of C was met.

Alg.1 (S.A.)									
T_0	α	L	C_{min}	C_{max}	C_{mean}	*std*	*time*	*tot_it*	*opt_it*
50	0.95	750	109.08	145.48	126.74	7.92	19	175	147

Table 5. Best empirical values of the parameters and results for SA being applied to a problem with 170 cans

A general remark which is confirmed by this particular example is that performances of a SA algorithm are rather robust w.r.t. parameter setting. Note also that variants of the basic SA algorithm were tried on the same problem. We shall come back to this in subsequent sections.

5. Variants and technical improvements

Several alternatives for choosing the solution space, the objective function and the neighborhood structures generally exist and making a good choice is of course a decision of crucial importance. This fact was already suggested in the previous section and it will be further emphasized in the next section by means of selected examples.

In the present section, we briefly review possible technical improvements and some sophisticated variants of SA which hopefully are representative of the evolution of practice among SA users. Note that our viewpoint is purely pragmatic. We believe that the link between the theoretical results about SA and what is actually observed are rather weak and it is an area that deserves more research.

5.1. Choice of a cooling schedule

Recall that only very slow cooling schedules guarantee theoretical convergence to the optimum. In view of this and current practice, which in contrast uses geometric cooling, Johnson et al.(1989) experimented with a variety of cooling schedules on the graph partitioning problem. Namely, they compare the behaviour of the algorithm under the following schedules:

- the usual geometric schedule;
- the logarithmic schedule (which is in agreement with theoretical convergence requirements)
$$T_k = \frac{C}{1+\log(k)}$$
where $k = 1, 2, \ldots$ is indexing the series of L steps done at temperature k;
- a schedule where T_o is decreased linearly with k;
- a schedule where the probability of accepting a bad move decreases linearly with k.

In their experiments with graph partitioning, Johnson et al.did not see any advantage in using schedules different from the usual geometric one. They observe in addition a "*lack of robustness with other schedules*". For instance, the quality of the final solution is more sensitive to the choice of the initial temperature. This conclusion, formulated in the particular case of graph partitioning, should of course be taken with the usual grain of salt; the statement of Johnson et al.(1989, p. 869) applies here: "*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*".

There is another category of schedules, called *adaptive*, where the temperature is monitored after the evolution of the cost function. Most of these schedules are based on the idea that "quasi–equilibrium" should be reached at a given temperature before it is decreased . This consideration generally results in tuning dynamically the length L of the "plateau". For example, Johnson et al.(1989) experimented with a simple adaptive schedule on graph partitioning (repeating a series of L steps without temperature change if either the best value or the average value has improved during the last series of L steps) This particular schedule did not result in any significant improvement, but as the variety of adaptive schedules is virtually unlimited, one can hope that other approaches could be more fruitful. 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. In another paper in this special issue, L. Ingber presents his experiments with his adaptive simulated annealing code (ASA) that has been publicly available for over two years.

An alternative attempt is based on the idea of "*strategic oscillation*" which belongs to the arsenal of another well-known general heuristic called "*Tabu Search*". This idea has been investigated by Osman in a series of problems such as for instance vehicle routing (Osman, 1993) and capacitated clustering (Osman and Christofides, 1994).

In Osman's approach, the system is cooled in a non-monotonic way. Starting from an initial temperature T_{00}, the cooling process begins in the usual manner. When progress becomes slow, due to freezing, temperature is raised to a higher value T_{01}, e.g. half the previous initial temperature T_{00}. Again the system is cooled in the usual manner until slow progress is reached. The k^{th} "oscillation" consists in a cooling which starts at an initial temperature

$$T_{0k} = \beta T_{0(k-1)},$$

a fraction $\beta < 1$ of the initial temperature of oscillation $(k-1)$.

For a description of elaborated versions of this idea, the reader is e.g. reported to the paper by Osman and Christofides (1994).

5.2. Possible technical improvements

We only mention very briefly a few suggestions which have been proposed in the literature for improving the efficiency of the basic SA algorithm. For more detail and evaluation on the graph partitioning problem, see Johnson et al.(1989).

Cutoffs: Stop the series of trials of a plateau (of the temperature schedule) as soon as a fixed proportion of moves have been accepted. This may result in time savings at high temperature.

Approximate exponentiation: Reading approximate values of the exponential function in tables instead of using the usual formula for the Boltzmann distribution, may lead to non-negligible time savings (up to $\frac{1}{3}$ according to Johnson et al.(1989)).

Starting from better than random solutions: There may be some advantage to start with good initial solutions (e.g. obtained by a conventional fast heuristic). This is the case when good solutions are characterized by a very particular structure which is difficult to find through a guided random search such as SA (see Johnson ct al.(1989) who experimented with the graph partitioning problem on special geometric graphs). In case better than random initial solutions are used, the starting temperature should not be too high in order not to destroy the structure of the initial solution. This however may prevent sufficient exploration of the search space and multiple starts (with *different* better than random initial solution) may be advisable.

More efficient choice of the moves: Much time is lost in SA due to the random choice of a move. Here are several suggestions.

- In the special case of graph partitioning, select a random permutation of the set V of vertices and, in a series of $|V|$ consecutive trials, consider each vertex exactly once for exchanging. In the case there are few solutions in the neighborhood which improve the current one, this may avoid missing them.
- **Rejection free annealing** (Green and Supowit (1986)) is an appealing idea for a more efficient search at low temperature. Implementation of this idea is problem– dependent and it will often be difficult to apply it in practice.
- **Locally optimized SA**. Instead of selecting a solution at random in the neighborhood of the current one, it seems more efficient to take the *best* solution in a *sub*neighborhood. In the example of nuclear fuel cans grouping described in section 4.1, we implemented such a variant of SA. The subneighborhood is a random sample of solutions obtained by exchanging pairs of cans. At each step, we generate and evaluate such a subneighborhood and we apply the usual decision rule to the best solution found in the subneighborhood. This leads to substantial improvement both in computer time and solution quality as can be seen by comparing Table 5 with Table 6 below.

Alg.1' Locally optimized S.A.										
	T_0	α	L	C_{min}	C_{max}	C_{mean}	*std*	*time*	*tot_it*	*opt_it*
a	50	0.90	100	97.82	143.96	113.91	8.13	11	71	71
b	50	0.90	200	96.73	162.05	109.59	8.05	23	70	69

Table 6. Best empirical values of the parameters and results for Locally optimized SA

5.3. Mixed approaches

The experiments briefly reported above show that the reasons for *strictly* applying the original principles of SA are rather weak. Indeed neither theoretical considerations nor the analogy with the cooling process of a physical system are able to explain the successes and failures of SA. In addition, it is pretty clear that importing other heuristic ideas may prove useful for improving the results. This trend is much in the spirit of "*Tabu Search*" which is described by F. Glover et al. (1993) as a technique based on "*selected concepts of artificial intelligence*". So, in modern heuristics, there are no reasons for *a priori* rejecting any idea. Conversely, legitimising an idea or an approach requires careful, honest and extensive experimentation. In particular, it is advisable to examine whether some of the ideas implemented in Tabu Search and Genetic Algorithms could not be fruitfully integrated in a SA algorithm. This path is currently explored by a growing number of researchers among which we point out for example, Osman and Christofides (1994), Moscato (1993), Andersen et al.(1993).

6. Applications

SA has been applied to such a rich variety of problems that it would be too long to mention them all. The reader is referred to the papers and books cited in the introduction and in particular to van Laarhoven and Aarts (1987), Collins et al.(1988), Vidal (1993) and Koulamas et al.(1994). Among the most important applications let us mention image processing (pattern recognition) and computer–aided circuit design (VLSI). Note that for large scale problems, a good parallel implementation is usually crucial but we will not treat the issue of parallelism in this introduction (see e.g. Aarts & Korst, 1989). Note also that SA has not only been applied to solve optimization problems on a discrete set but also on a continuous space of solutions. This is however done through discretization.

Our goal in this section is to outline applications of simulated annealing to a few combinatorial optimization problems. These were selected in order to further illustrate some points already mentioned in the previous chapters and give examples of more subtle implementations.

6.1. The Traveling Salesman Problem (TSP)

This is the problem to which SA was first applied. In the euclidean version of the problem, one has to find the shortest circuit which passes once and only once in each point of a set of points spread in the unit square of the plane. Although the results were encouraging at first glance, it soon became evident that the basic version of SA was outperformed by specialized heuristics like the well-known Lin-Kernighan algorithm. The basic version of SA for the TSP is based on the 2-exchange neighborhood: a tour can be described as an ordered list of the cities (= points) and the neighbors of the current tour can be obtained by selecting 2 cities in the tour and reversing the order in which all the cities in between are visited.

Bonomi and Lutton (1984) built a specialized version of SA for the euclidean TSP on n cities drawn uniformly at random in the unit square. Probabilistic analysis shows, when n is large, that the distance between consecutive cities in an optimal tour are small as compared to 1, the unit square side length. Hence the square is divided in disjoint sub-regions whose size has the same order of magnitude as the average step of the optimal tour. The initial tour crosses each subregion only once and all cities of a subregion are visited consecutively in a certain order; 2-exchanges are performed only on pairs of points belonging to the same subregion or neighboring ones. In this way crucial structural properties of good tours are maintained throughout (by using relatively moderate initial temperature) and no time is lost with aberrant tours. For a number of cities above 400, it seems that the approach performs well as the length of the best tour found is within a few percents of the theoretical optimal value (which can be predicted by probabilistic considerations). This is an example where theoretical knowledge about a problem has been used to reinforce the efficiency of an algorithm.

6.2. Graph coloring

Hertz and de Werra (1987) proposed a combined Tabu Search algorithm for coloring large graphs. Their proposal is equally valid in a SA approach. First use SA for finding a large independent set (i.e. a set of nodes, no pair of which is linked by an arc of the graph). This can be done by an "*annealed*" version of an improvement procedure which aims at enlarging the size of the set and diminishing the number of "interior" arcs. Once a large independent set has been found, all its nodes are colored with a single color and one tries to find a large independent set in the set of not yet colored nodes. The procedure continues in this way until the number of not yet colored nodes falls below some threshold. Then the remaining nodes are colored using the SA coloring algorithm described in section 4. The Tabu Search version of the approach yields good results (near to the theoretically estimated optimal value) for randomly generated graphs. This illustrates the potentialities of combined approaches.

6.3. Jobshop scheduling

In a Jobshop problem each job consists in several operation to be performed in a prescribed order on a specific machine in the shop. The goal is e.g. to minimize the makespan i.e. the earliest time on which all operations of all jobs can be accomplished. Scheduling a jobshop amounts to specify for each machine the order of succession (sequencing) of the operations on this machine as well as the beginning of the processing period (scheduling) of each operation. Jobshop scheduling is a particularly (NP-) hard optimization problem; there are many test examples of moderate size (less than 50 jobs) for which an optimal schedule is not known despite the efforts devoted to their solving.

Van Laarhoven and Aarts (1987) have proposed and tested an approach based on the *disjunctive graph* model due to Roy and Sussmann (1964). In the disjunctive graph, each operation is represented by a vertex with two additional vertices representing the starting and finishing times of the schedule. Directed arcs link consecutive operations in a job; undirected edges link all pairs of operations to be done on the same machine. A feasible schedule is associated to any orientation without cycle of the non directed edges. Each *vertex* is assigned a weight equal to the duration of the corresponding operation. Solving the scheduling problem amounts to find an acyclic orientation of the non directed edges in which the length of the critical paths (i.e. the paths of maximal weight from the initial to the final vertex) is minimal. Van Laarhoven and Aarts have shown that reversing the orientation of one arc of a critical path results in a new feasible solution and a sequence of such transformations allows to reach a global optimum. So, the neighborhood of a solution is defined as the set of all solutions that can be obtained by reversing the orientation of an arc of a critical path in the oriented graph.

With this approach, using very carefully tuned cooling schedules, the authors were able to solve difficult test problems to optimality (among others the famous Fisher and Thompson's problem with 10 jobs of 10 operations each). It should be noticed that it is not very easy to reproduce the claimed results; the frequency with which we find the optimal solution when starting with different initial solutions or different initializations of the random sequence is not very high. This means that there could be a certain lack of robustness of the suggested cooling schedules. But the most important drawback is not there. Quite often, in practical situations, one is not dealing with a purely classical jobshop problem. Almost always, there are additional constraints specific to the treated case. It is then essential that the algorithm for the classical situation could be flexible enough to take into account some kinds of additional constraints. This is not the case with the disjunctive graph approach in a jobshop scheduling problem with additional due date constraints (each job should be completed before a specified "due" date).

For the jobshop problem, another algorithm can be implemented which allows for more flexibility. Associated with each machine, a *priority list* of the operations to be performed on the machine is held. This list does not specify the exact order in which the operations will be executed but an order of priority. The algorithm roughly runs as follows. One looks at the first time a machine becomes idle and the first operation in the priority list associated to the machine is considered for being loaded on the machine. If the operation may not be executed immediately due to anteriority constraints, one looks at the second operation in the priority list and so on. SA is used for modifying (through 2-exchanges) the order of operations in each of the priority lists. This procedure of course does not give as good results as the previous one but it is easy to take into account additional constraints. In particular due dates can be managed by adding to the objective function a penalty term proportional to the degree of violation of the date. This example illustrates that some approaches (and it is usually the case with SA) allow to take additional constraints into account while other do not and this is often of crucial importance in applications.

7. Conclusions

In the present paper we tried to provide some guidelines for implementing a metaheuristic approach, i.e. to adapt the general ideas to a particular optimization context. A traditional warning has been exposed by Johnson et al.(1991, p. 405): "*SA is not a panacea but a potentially useful tool*". It is clear that either exact algorithms or specialized heuristics or either other metaheuristics, like Tabu Search or Genetic Algorithms, may be better suited in specific situations. However SA has now its place as a tool in the arsenal of optimization techniques.

The main advantages of SA are a relative ease of implementation and a good flexibility. The strategic and tactical choices (choice of a neighborhood structure, parameter, ...) to be made for adapting the approach to a particular problem are rather few and one can imagine to implement SA for quickly obtaining the first estimation of a good solution. If better solutions are needed, the initial approach can be refined or one may turn to other approaches. Also, constraints specific to the problem, often additional constraints not known from the beginning, can be implemented without fundamental change either in a hard or a soft manner (either by restricting the solution space or adding penalties in the objective function).

The main drawback of SA is its inefficiency (long computing times, or inapplicability in very large size problems). We have seen that this inefficiency can be corrected by several types of change in the basic strategy (strategic oscillation, selection of the best solution in a subneighborhood; nested approach like in the coloring problem, restricted neighborhoods in the TSP, ...).

For summarizing, we believe that SA has become and remains a useful tool. It is especially the case in practical problems where ease of implementation and flexibility are more important than absolute performance either in computing times or solution quality. This is the case for decision support systems where the users are usually non-specialists in optimization but are experts on the concrete problem to be solved.

References

AARTS, E. and KORST, J. (1989) *Simulated Annealing and Boltzman Machines: a stochastic approach to combinatorial optimization and neural computing.* Wiley.

ANDERSEN, K. et al. (1993) Design of a teleprocessing communication network using simulated annealing. In: Vidal, R.V.V. (ed.), *Applied Simulated Annealing*, Lecture Notes in Economics and Mathematical Systems, Springer, Berlin, 201-216.

AZENCOTT, R. (ed.) (1992) *Simulated Annealing. Parallelization Techniques.* Wiley.

BONOMI, E. and LUTTON, J.L. (1984) The N-city travelling salesman problem: Statistical mechanics and the Metropolis algorithm. *SIAM Review*, **26**, 551-568.

ČERNY, V. (1985) A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, **45**, 41-55.

CHAMS, M., HERTZ, A. and DE WERRA, D. (1987) Some experiments with simulated annealing for coloring graphs. *Eur.J.Op.Res.*, **32**, 260-266.

COLLINS, N.E. et al. (1988) Simulated Annealing - an annotated bibliography. *AJMMS*, **8**, 209-507.

DEKKERS, A. and AARTS, E. (1991) Global optimization and simulated annealing. *Math. Progr.*, **50**, 367-393.

DOWSLAND, K.A. (1993) Simulated Annealing. In Reeves, C.R. (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell, Oxford, 20-65.

EGLESE, R.W. (1990) Simulated Annealing: a tool for operational research, *EJOR*, **46**, 271-281.

FISHER, H. and THOMPSON, G.L. (1963) Probabilistic learning combinations of local job-shop scheduling rules. In Muth, J. and Thompson, G. L. (eds.), *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, N.J., 225-251.

GLOVER, F. and GREENBERG, H.J. (1989) New approaches for heuristic search: a bilateral link with artificial intelligence. *EJOR*, **39**, 119-130.

GLOVER, F., TAILLARD, E. and DE WERRA, D. (1993) A user's guide to tabu search. *Annals of Operations Research*, **41**, 3-28.

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

HERTZ, A. and DE WERRA, D. (1987) Using tabu search techniques for graph coloring. *Computing*, **29**, 345-351.

HUANG, M.D., ROMEO, F. and SANGIOVANNI-VINCENTELLI, A. (1986) An efficient general cooling schedule for simulated annealing. *Proc. IEEE Int. Conf. on CAD (ICCAD 86)*, 381-384, Santa Clara, Calif.

JOHNSON, D.S. et al. (1989) Optimization by Simulated Annealing: an experimental evaluation - Part I (Graph Partitioning). *Operations Research*, **37**, 865-892.

JOHNSON, D.S. et al. (1991) Optimization by Simulated Annealing: an experimental evaluation - Part II (Graph Coloring and Number Partitioning). *Operations Research*, **39**, 378-406.

KIRKPATRICK, S. et al. (1983) Optimization by Simulated Annealing. *Science*, **220**, 671-680.

KOULAMAS, C. et al. (1994) A survey of Simulated Annealing applications to Operations Research problems. *Omega*, **22**, 41-56.

LAARHOVEN VAN, P.J.M. and AARTS, E.H.L. (1987) *Simulated Annealing: Theory and Applications.* Reidel.

LAARHOVEN VAN, P.J.M. (1988) *Theoretical and Computational Aspects of Simulated Annealing.* Centre for Mathematics and Computer Science, Amsterdam.

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

LIÉGEOIS, B., PIRLOT, M., TEGHEM, J., TRAUWAERT, E., and TUYTTENS, D. (1992) Balanced grouping through simulated annealing. In Vidal, R.V.V. (ed.), *Applied Simulated Annealing*, Lecture Notes in Economics and Mathematical Systems, Springer, Berlin, 275-290.

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

METROPOLIS, N. et al. (1953) Equation of state. Calculation by fast computing machines. *Journal of Chem. Phys.*, **21**, 1087-1092.

MOSCATO, P. (1993) An introduction to population approaches for optimization and hierarchical objective functions: A discussion of the role of tabu search. *Annals of Operations Research*, **41**, 85-123.

OSMAN, I.H. (1993) Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, **41**, 421-453.

OSMAN, I. H. and CHRISTOFIDES, N. (1994) Capacitated clustering problems by hybrid simulated annealing and tabu search. *Int. Trans. Oper. Res.*, **1**, 317-337.

OTTEN, R.H.J.M. and VAN GINNEKEN, L.P.P.P. (1990) *Annealing Algorithm.* Kluwer.

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

PIRLOT, M. (1993) General Local Search Heuristics in Combinatorial Optimization: a tutorial. *Belgian Journal of Operations Research, Statistics and Computer Science*, **32**, 7-67.

ROY, B. and SUSSMANN, B. (1964) Les problèmes d'ordonnancement avec contraintes disjonctives, Note DS 9 bis. SEMA, Paris.

SHRAGOWITZ, E. and LIN, R.B. (1990) Combinatorial Optimization by Stochastic Automata, *Annals of Operations Research*, **22**, 293-324.

SIARRY, J. and DREYFUS, G. (1989) La méthode du Recuit Simulé (in French), Paris, IDSET.

TUYTTENS, D., PIRLOT, M., TEGHEM, J., TRAUWAERT, E. **and** LIÉGEOIS, B. (1994) Homogeneous grouping of nuclear fuel cans through simulated annealing and tabu search. *Annals of Operations Research*, **50**, 575-607.

VIDAL, R.V.V. (ed.) (1993) *Applied Simulated Annealing*, Lecture Notes in Economics and Mathematical Systems. Springer, Berlin.