European Journal of Operational Research 46 (1990) 271–281
North-Holland

Invited Review

Simulated Annealing: A tool for Operational Research

R.W. EGLESE
Department of Operational Research and Operations Management, The Management School, Lancaster University, Lancaster LA1 4YX, UK

Abstract: This paper describes the Simulated Annealing algorithm and the physical analogy on which it is based. Some significant theoretical results are presented before describing how the algorithm may be implemented and some of the choices facing the user of this method. An overview is given of the experience of experimenters with SA and some suggestions are made for ways to improve the performance of the algorithm by modifying the 'pure' SA approach.

Keywords: Simulated annealing, heuristics, optimisation

1. Introduction

Simulated Annealing (SA) is a method for obtaining good solutions to difficult optimisation problems which has received much attention over the last few years. The recent interest began with the work of Kirkpatrick et al. (1983), and Cerny (1985). They showed how a model for simulating the annealing of solids, as proposed by Metropolis et al. (1953) could be used for optimisation problems, where the objective function to be minimised corresponds to the energy of the states of the solid.

Since then, SA has been applied to many optimisation problems occurring in areas such as computer (VLSI) design, image processing, molecular physics and chemistry, and job shop scheduling. There has also been progress on theoretical results from a mathematical analysis of the method, as well as many computational experiments comparing the performance of SA with other methods for a range of problems. The aim of the paper is to provide some understanding and guidance for those interested in using SA, particularly those involved in Operational Research.

SA is not the only heuristic search method to have received attention recently. Glover and Greenberg (1989) summarise the approaches offered by genetic algorithms, neural networks, tabu search, target analysis, as well as SA. Maffioli (1987) shows how SA can be considered as one type of randomized heuristic for combinatorial optimisation problems.

Received November 1989

2. Description of SA

The SA algorithm will be described as applied to a combinatorial optimisation problem. Many combinatorial optimisation problems have been shown to be NP-hard, which means that the running time for any algorithms currently known to guarantee an optimal solution is an exponential function of the size of the problem. Examples include the travelling salesman problem. SA is one of many heuristic approaches designed to give a good though not necessarily optimal solution, within a reasonable computing time. SA has also been extended to optimisation problems for continuous variables; a summary of these approaches is given in van Laarhoven and Aarts (1987).

Suppose that the solution space S is the finite set of all solutions and the cost function, f, is a real valued function defined on members of S. The problem is to find a solution or state, $i \in S$, which minimises f over S.

SA is a type of local search algorithm. A simple form of local search (a descent algorithm) starts with an initial solution perhaps chosen at random. A neighbour of this solution is then generated by some suitable mechanism and the change in cost is calculated. If a reduction in cost is found, the current solution is replaced by the generated neighbour, otherwise the current solution is retained. The process is repeated until no further improvement can be found in the neighbourhood of the current solution and so the descent algorithm terminates at a local minimum. Table 1 describes the descent algorithm in pseudo-code.

Although a descent algorithm is simple and quick to execute, the disadvantage of the method is that the local minimum found may be far from any global minimum. One way of improving the solution is to run the descent algorithm several times starting from different initial solutions, and take the best of the local minima found. In SA, instead of this strategy, the algorithm attempts to avoid becoming trapped in a local optimum by sometimes accepting a neighbourhood move which increases the value of f. The acceptance or rejection of an uphill move is determined by a sequence of random numbers, but with a controlled probability. The probability of accepting a move which causes an increase δ in f is called the acceptance function and is normally set to $\exp(-\delta/T)$ where T is a control parameter which corresponds to temperature in the analogy with physical annealing. This acceptance function implies that small increases in f are more likely to be accepted than large increases, and that when T is high most moves will be accepted, but as T approaches zero most uphill moves will be rejected. So in SA, the algorithm is started with a relatively high value of T, to avoid being prematurely trapped in a local optimum. The algorithm proceeds by attempting a certain number of neighbourhood moves at each temperature, while the temperature parameter is gradually dropped. The SA algorithm is illustrated in pseudo-code in Table 2. The single loop of the descent algorithm has been replaced by a double loop: in the outer loop the temperature is changed and the inner loop determines how many neighbourhood moves are to be attempted at each temperature.

Table 1
Descent algorithm in pseudo-code

```
Select an initial state $i \in S$;
Repeat
  Generate state $j$, a neighbour of $i$;
  Calculate $\delta = f(j) - f(i)$
  If $\delta \leqslant 0$ then $i := j$;
until $f(j) \geqslant f(i)$ for all $j$ in the neighbourhood of $i$;
```

Table 2
Simulated Annealing algorithm in pseudo-code

```
Select an initial state $i \in S$;
Select an initial temperature $T > 0$;
Set temperature change counter $t = 0$;
Repeat
Set repetition counter $n = 0$;
  Repeat
  Generate state $j$, a neighbour of $i$;
  Calculate $\delta = f(j) - f(i)$;
  If $\delta < 0$ then $i := j$
    else if random(0, 1) $< \exp(-\delta/T)$ then $i := j$;
  $n := n + 1$;
  until $n = N(t)$;
$t := t + 1$;
$T := T(t)$;
until stopping criterion true.
```

3. The physical analogy

The motivation for the SA algorithm comes from an analogy between the physical annealing of solids and combinatorial optimisation problems. Physical annealing refers to the process of finding low energy states of a solid by initially melting the substance, and then lowering the temperature slowly, spending a long time at temperatures close to the freezing point. An example would be producing a crystal from the molten substance. In a liquid, the particles are arranged randomly. But the ground state of the solid, which corresponds to the minimum energy configuration, will have a particular structure, such as seen in a crystal. If the cooling is not done slowly, the resulting solid will not attain the ground state, but will be frozen into a metastable, locally optimal structure, such as a glass or a crystal with several defects in the structure.

In the analogy, the different states of the substance correspond to the different feasible solutions to the combinatorial optimisation problem, and the energy of the system corresponds to the function to be minimised.

Physical annealing has been successfully modelled as a Monte Carlo simulation. Back in the fifties, Metropolis et al. (1953) introduced a simple algorithm to simulate a collection of atoms at a given temperature. At each iteration, an atom is given a small random displacement and the resulting change, δ, in the energy of the system is calculated. If $\delta < 0$ then the resulting change is accepted, but if $\delta > 0$, the change is accepted with probability $\exp(-\delta/k_B T)$, where T is the temperature and k_B a physical constant called the Boltzmann constant. If a large number of iterations are carried out at each temperature, the system attains thermal equilibrium at each temperature. At thermal equilibrium, the probability distribution of the system states follows a Boltzmann distribution where the probability of the system being in state i at temperature T is $(1/Z(T)) \cdot \exp(-E_i/k_B T)$ where E_i is the energy of state i and $Z(T)$ is the partition function required for normalisation. The acceptance function introduced by Metropolis et al. ensures that the system evolves into the required Boltzmann distribution.

So in the analogy, the different states of the substance correspond to different feasible solutions to the combinatorial optimisation problem, and the energy of the system corresponds to the function to be minimised. The descent algorithm corresponds to 'rapid quenching' where the temperature is reduced quickly, so that only moves which result in a reduction of the energy of the system are accepted. The determination of the initial temperature, the rate at which the temperature is reduced, the number of iterations at each temperature and the criterion used for stopping is known as the annealing or cooling schedule. The choice of annealing schedule has an important bearing on the performance of the algorithm and is discussed in a later section.

The physical analogy has led to ideas from statistical physics being suggested as ways of determining the annealing schedule. For example, specific heat and entropy might be measured and used to determine a suitable temperature.

4. Theoretical results

The SA algorithm can be modelled using the theory of Markov chains. If the temperature parameter, T, is kept constant, then the transition matrix, P_{ij}, representing the probability of moving from state i to state j is independent of the iteration number, which corresponds to a homogeneous Markov chain. It may be shown that providing it is possible to get from any state i to any other state j in a finite number of moves with non-zero probability, then the Markov chain corresponding to the SA algorithm has a unique stationary distribution $q(i)$, which does not depend on the initial state. The stationary distribution corresponds to the Boltzmann distribution at thermal equilibrium in the physical analogy. As a corollary of this result, the limit as $T \to 0$ of this stationary distribution is a uniform distribution over the set of optimal solutions, i.e. the SA algorithm converges asymptotically to the set of globally optimal solutions. Asymptotic convergence to the set of globally optimal solutions is an encouraging result (which also holds for more general acceptance functions and transition mechanisms) but this result does not tell us anything about the number of iterations required to achieve convergence. Aarts and van Laarhoven (1985) show that the stationary distribution is approximated arbitarily closely, only if the number of iterations is at least quadratic in the size of the solution space. Since the size of the solution space is usually exponential in the size of the problem itself, this means that approximating the stationary distribution arbitarily closely results in exponential running time for SA.

However the SA algorithm can also be described as a sequence of homogeneous Markov chains of finite length, using decreasing values of the temperature parameter T. This can be considered as a single inhomogeneous Markov chain, as the transition probabilities are now not independent of the number of iterations. Several results have been derived giving sufficient conditions for asymptotic convergence to the set of global optima. Hajek (1988) gives the strongest result in that he provides necessary and sufficient conditions for this result. He shows that if $T(k) = c/\log(1 + k)$ where k is the number of iterations, the condition for asymptotic convergence is that c be greater than or equal to the depth of the deepest local minimum which is not a global minimum. This temperature function represents very slow cooling. It has also been shown (e.g. Mitra et al., 1986) that attempting to approximate arbitarily closely to the uniform distribution on the set of optimal solutions, typically leads to a number of iterations which is larger than the size of the solution space, and so results in exponential running time for most problems. Note that the theoretical results for the asymptotic convergence of SA modelled as an inhomogeneous Markov chain do not require the stationary distribution to be achieved at any non zero temperature.

It would be very useful to the user of SA to have some theoretical results on the average or expected performance of the algorithm. Unfortunately these are difficult to obtain. Sasaki and Hajek (1988) give some results for the maximum matching problem, but their proofs are lengthy and complex. Jerrum and Sinclair (1988) present an alternative proof of their results by analysing the Markov chains using the concept of 'rapid mixing' and suggest that their approach may lead to further theoretical performance measures for SA and related algorithms.

5. Implementing SA

In order to use SA for a particular combinatorial optimisation problem, there are a number of decisions to be made. These decisions can be divided into two groups following Johnson et al. (1987): problem specific choices and generic choices.

5.1. Problem specific choices

The problem must be clearly formulated, so that the set of feasible solutions is defined. The neighbourhood of any solution must also be defined as well as a way of determining the value of the objective to be minimised (or its change in value). An initial solution must also be generated.

Several researchers (e.g. Cerny, 1985; Matsuo et al., 1988) have shown that the efficiency of SA depends on the neighbourhood structure that is used. This is not surprising given Hajek's result in the previous section, which showed that the rate of cooling required for asymptotic convergence to the set of optimal solutions depends on the depth of the local minima, i.e. to the topology that is imposed by the neighbourhood structure. In general, a neighbourhood structure which imposes a 'smooth' topology, where the local optima are shallow is preferred to a 'bumpy' topology where there are many deep local minima.

Connolly (1987) has done some research on a general purpose SA algorithm which can be used for any problem expressed as a 0–1 integer program. Neighbourhood moves are generated automatically by changing an individual variable from 0 to 1 or vice versa, and then restoring feasibility. Experiments show that although this procedure is less efficient than a neighbourhood designed to take advantage of the problem structure, the method performed well compared to a commercial integer programming package.

If the problem is constrained, a choice must be made between restricting the solution space to solutions which conform to the constraints, or allowing solutions which break the constraints at the expense of a suitably defined penalty function. The second option is likely to lead to simpler neighbourhood moves, and a smoother topology, particularly if the penalty weights can be kept small.

5.2. Generic choices

These choices must be made for any implementation of SA and constitute the annealing or cooling schedule, viz.

(i) the initial value of the temperture parameter T,

(ii) a temperature function, $T(t)$, to determine how the temperature is to be changed,

(iii) the number of iterations, $N(t)$, to be performed at each temperature, and

(iv) a stopping criterion to terminate the algorithm.

A great variety of cooling schedules have now been suggested. These have been classified in Collins et al. (1988), which also indicates which variant different authors have tried.

The earliest proposed cooling schedules were based on the analogy with physical annealing. So, for example, Kirkpatrick et al. (1983) set their initial value of T to be large enough so that virtually all transitions are accepted, which corresponds to heating up a substance until all the particles are randomly arranged in a liquid. A proportional temperature function is used, i.e. $T(t+1) = \alpha T(t)$ where α is a constant. Typical values of α used in practice lie between 0.8 and 0.99. Such a function provides smaller decrements in T as zero temperature is approached. The number of iterations at each value of T, $N(t)$, is determined by a sufficient number of transitions being accepted subject to a constant upper bound. This is an attempt to bring the system to a state corresponding to thermal equilibrium in the physical analogy. Other simple schemes may keep $N(t)$ constant, or make it proportional to the size of the problem instance or the size of the neighbourhood defined. The SA algorithm is stopped when the solution obtained at each temperature change is unaltered for a number of consecutive temperature changes. The final state is then said to correspond to the frozen state.

Other cooling schedules make a more direct appeal to the theoretical results on asymptotic convergence described in Section 4. The number of iterations to carry out at each value of T, $N(t)$, is chosen so that the system is 'sufficiently close to' the stationary distribution at that value of T. Aarts and Korst (1989), and van Laarhoven and Aarts (1987) refer to this as 'quasi-equilibrium'. Different rules or heuristics for deciding when quasi-equilibrium has been reached lead to different cooling schedules.

However, as Hajek's (1988) result shows, for asymptotic convergence to the set of optimal solutions, the condition is for the cooling to be carried out sufficiently slowly, rather than there necessarily being any requirement to attain a state corresponding to thermal equilibrium at a succession of reducing temperatures. The result also suggests that if T is kept fixed for a number of iterations, there is a trade-off between large reductions in T and a small number of iterations at fixed T. So at one extreme there is, for example, the scheme proposed by Lundy and Mees (1986), where there is only a single iteration at each temperature. They use heuristic arguments to derive a temperature function of the form

$$T(t+1) = T(t)/(1 + BT(t))$$

where B is a constant, and which is equivalent to

$$T(t) = C_1/(1 + tC_2)$$

where C_1 and C_2 are constants. As the number of iterations increases, this will represent slower cooling than a proportional temperature reduction with fixed values of α and N. At the other extreme, there is the heuristic scheme based on SA proposed by Connolly (1988) which suggests that the majority of the iterations should be conducted at a suitably fixed temperature.

One broad distinction which can be made between cooling schedules is to contrast those with and without feedback for $T(t)$ and $N(t)$. Simple schedules without feedback determine $T(t)$ and $N(t)$ at the start of any run. Other schedules allow feedback from the progression of the algorithm to affect the current values of $T(t)$ and $N(t)$. The schedule described by Kirkpatrick et al. (1983) allows the number of accepted transitions to influence $N(t)$. Another example is the cooling schedule proposed by Aarts and van Laarhoven (1985) where $N(t)$ is kept constant, but the temperature function depends on an estimate of the standard deviation of the distribution of objective function values at the current temperature, which must be monitored during the execution of the algorithm. Aarts and van Laarhoven have shown that their cooling schedule normally leads to a worst case running time which is a polynomial function of the size of the problem. Other researchers have suggested that alternative measurements, such as monitoring the specific heat of the system, may be a useful guide to a suitable rate of cooling.

The relative performance of these cooling schedules will be discussed in a later section. However, many users of SA have found that implementing SA in the ways described in this section has led to either very long run times or relatively poor solutions when the run time is limited. Thus many users have modified the SA approach in ways which take the algorithm further away from the physical analogy on which it is based, but which lead to a more efficient algorithm. These are described in the next section.

6. Modifications to SA

6.1. Simple modifications

This section contains some simple suggestions which various authors have made to improve the effectiveness of the basic algorithm. They can all be implemented easily in a computer code.

6.1.1. Storing best solution so far

The SA algorithm sometimes accepts solutions which are worse than the current solution. It is therefore possible in any single SA run for the final solution to be worse than the best solution found during the run. It is not usually computationally expensive to store details of the best solution found so far, so better solutions may result from run times similar to those for the basic algorithm. In addition, Glover and Greenberg (1989) argue that with this modification, there is less need for the SA algorithm to rely on a strong stabilizing effect over time. This idea is supported by Connolly's (1988) modification of SA, where having found a suitable fixed temperature, all the remaining iterations are carried out that temperature. In the final phase, a descent algorithm can be employed to find the local optimum containing the best solution encountered in the earlier phases.

6.1.2. Sampling the neighbourhood without replacement

When the temperature parameter, T, gets low in the latter part of an SA run, the probability of accepting worse solutions than the current one becomes small. Thus in the region close to a local optimum, most of the computer time is spent rejecting worse solutions. If there are only one or two neighbourhood moves which give improved solutions, the basic SA algorithm may take a lot of time to find them. Therefore some researchers have suggested that neighbourhood moves should be generated in such a way that all possible moves in the neighbourhood of a solution are attempted before repeating a move, unless a new solution is accepted. Johnson et al. (1987) implement this idea by generating a new set of potential neighbourhood moves every n iterations, where n is the size of the neighbourhood. Within each block, each potential move is considered once and in a random order. Experiments with graph partioning problems showed that this modification gave significantly better results than simply choosing neighbourhood moves at random in a similar run time. Connolly (1988) also presents experimental evidence of the benefit of sampling the neighbourhood without replacement, though in this case it is implemented by simply searching through the neighbourhood in a sequential rather than random manner.

Another advantage of this approach is that when the algorithm has been terminated and there has been no improvement in the solution for at least the last n moves, where n is the size of the neighbourhood, then the final solution is guaranteed to be a local optimum.

6.1.3. Alternative acceptance probabilities

Some researchers have considered replacing the standard function, $\exp(-\delta/T)$ for the probability of accepting a solution which increases the objective by δ, with some alternative function. Indeed there has been some theoretical work to show that asymptotic convergence holds for more general acceptance functions (see e.g. Anily and Federgruen, 1987; Faigle and Schrader, 1988; Romeo and Sangiovanni-Vincentelli, 1985). Romeo and Sangiovanni-Vincentelli provide some experimental evidence that an alternative acceptance scheme does not significantly alter the quality of solutions found. Matsuo et al. (1988) find that when their algorithm is restricted to a small neighbourhood, using an acceptance function which is independent of δ, when δ is positive, yields results nearly as good as use of the standard function.

In applications of SA where the calculation of the value of $\exp(-\delta/T)$ is a significant factor in the run time, Johnson et al. (1987) propose that the exponential function is approximated by a simple table lookup scheme. Experiments on graph partioning problems saved one third of the running time with no apparent effect on the quality of the solutions.

6.2. More complex modifications

The modifications described in this section require rather more work to implement in a computer algorithm. The nature of some of these modifications depends more closely on the problem under consideration, and so although these ideas may result in a more powerful algorithm for a specific application, they cannot always be easily transferred to different applications.

6.2.1. Combination of SA with another method

There are two ways in which SA may be combined with an alternative method; either using the alternative method to provide a 'good' initial solution which SA attempts to improve, or by using SA to provide a 'good' initial solution as a starting point for the alternative method.

The first of these approaches is illustrated by Chams et al. (1987) for example, when considering graph colouring problems. Johnson et al. (1987) also provide some experimental results for the graph partitioning problem. They show that both quality of solution and running time may be improved by the use of a good starting solution. When a good initial solution is used, the initial temperature in the cooling schedule is reduced, otherwise the benefits of the good initial solution will be lost. They also show that starting solutions which take advantage of the special structure of the problem instance being considered seem preferable to those obtained by general heuristics.

The second approach is exemplified by using SA as a way of obtaining a good initial solution for a branch and bound algorithm or integer programming algorithm.

6.2.2. Problem specific modifications

Many users of SA adapt the basic approach to take advantage of special features of the problem being considerd. In many applications, the bulk of the run time is taken up calculating δ, the change in the objective value at each transition. Grover (1986) shows that significant speed-ups can be obtained by calculating δ approximately instead of exactly. These approximations are shown not to affect the quality of the solution significantly, provided the errors are less than a function of the temperature parameter, T. Tovey (1988) also suggests that the algorithm may be speeded up by using a quicker, approximate method to calculate δ. However, he incorporates this idea in a scheme he calls a 'poor man's swindle'. The faster, approximate method of calculating δ is not used every iteration, but only with a certain probability, which is updated as the algorithm proceeds so that the resulting algorithm will simulate the results of the basic SA approach. Tovey also suggests a 'Neighbourhood Prejudice Swindle'. The idea here is to use problem-specific information to identify a promising subset of the neighbourhood, and a generation mechanism that will give a greater probability of attempting moves into the promising subset. Although he shows that this can be advantageous compared to the basic SA approach, it is not clear whether it performs significantly better than the schemes suggested for sampling from a neighbourhood without replacement.

Greene and Supowit (1986) propose a rejectionless method. This scheme also aims to reduce the amount of time spent in the latter part of a SA run when the majority of the moves are rejected. In this scheme, a probability distribution is constructed over the set of all moves to show the relative probability of a move being accepted if it is chosen. A move is then selected at random according to this distribution and accepted automatically. It can be shown that the result of this procedure is equivalent to the basic SA algorithm. For the graph partitioning problem the probability distribution can be updated efficiently leading to quicker runs once the proportion of acceptances in the basic algorithm falls below some threshold. To apply this approach to other problems means finding an efficient way to update the probability distribution, and this depends on problem-specific information.

6.2.3. Parallel versions

Another approach to speeding up SA is to implement a parallel version of the algorithm. A good discussion of the issues involved is found in the book by Aarts and Korst (1989).

Two general strategies may first be distinguished: single-trial parallelism and multiple-trial parallelism. In the first strategy, the calculations to evaluate a single trial are divided between several processors. The implementation of this strategy and the speed-up that can be obtained are clearly problem dependent. In multiple-trial parallelism, however, trials are evaluated in parallel. Some approaches reported in the literature make use of problem-specific features to carry this out. For example, a placement problem may be subdivided so that different processors deal with local changes in different regions. However, Aarts and Korst describe three general approaches that use multiple-trial parallelism, viz. the division algorithm, the clustering algorithm and the error algorithm.

In the division algorithm, the number of iterations at each temperature is divided equally between the processors. After a change in temperature, each processor may simply start from the final solution obtained by that processor at the previous temperature. The best solution from all the processors is then taken to be the final solution. Another variant of this approach is to communicate the best solution from all the processors to each processor every time the temperature changes. Aarts and Korst found no significant differences in the performance of these two variants.

In the clustering algorithm, two or more processors are used to generate one sequence of accepted moves. Processors evaluate possible moves independently until one is accepted. This is then communicated to all the processors in the cluster, which abort their current calculations and resume with the accepted solution. This strategy is likely to be inefficient at the start of a run when the temperature parameter is high and many transitions are accepted, though it should become progressively more efficient as the temperature approches zero and few new solutions are accepted. A suggested strategy is therefore to start as the division algorithm already described, but as soon as the estimated efficiency of using two processors exceeds 50%, the processors are clustered into pairs. The clustering process continues until eventually all the processors form one cluster.

Finally, there is the error algorithm. Here all the processors are used to investigate potential neighbourhood moves in parallel. Any accepted move updates the current solution, but implementation on a computer using a global memory may need to be done sequentially. The name 'error algorithm' derives from the fact that some calculations made by a processor of the change in objective value for a potential move may be calculated wrongly if another processor has just accepted a move of which the processor is unaware.

Aarts and Korst compared the performances of these three parallel versions of SA on a computer which allowed up to 8 processors to be used in parallel. The algorithms were tested on a 100 city travelling salesman problem. The results show that both the division algorithm and the clustering algorithm could give results of similar quality to the sequential SA algorithm and with an efficiency close to 1. The results for the error algorithm were, on the other hand, quite poor, with no convergence being achieved with eight processors. This is in contrast to other implementations of types of error algorithms for placement problems, where high efficiencies have been reported (e.g. Casotto, Romeo and Sangiovanni-Vincentelli, 1987). The discrepancy may be explained by the fact that the neighbourhood structure usually employed for placement problems only allow local changes which seldom lead to errors if two such changes are accepted simultaneously. The 2-opt neighbourhood structure used for the travelling salesman problem involves reversing the order of several cities in the tour and so the chance of errors in calculating the change in objective from two potential moves which are accepted simulataneously is significant. Although some empirical results on convergence of error algorithms is available, there is not as yet, any precise analysis of the conditions required for convergence.

In their book, Aarts and Korst go on to describe how an efficient massively parallel implementation of SA could be carried out in a neural network, based on the model of Boltzmann machines. Encouraging results are obtained for some graph problems (e.g. max cut problem) where simulations show that the Boltzmann machine approach would give solutions of comparable quality to SA in considerably less computation time, when implemented on a suitable computer. However the Boltzmann machine approach was much less successful than sequential SA for travelling salesman problems. This is a growing area of research as Boltzmann machines and other neural network models are implemented on dedicated hardware. For example, a design for a VLSI chip has been presented by Alspector and Allen (1987) and proposed for optical implementations by Farhat (1987).

7. Computational results

What evidence is there that SA performs well in obtaining good solutions within reasonable run times? How does it compare with other approaches?

SA is a stochastic algorithm, requiring a neighbourhood structure to be specified as well as a number of parameters to provide a cooling schedule. Many variants of the basic algorithm have been proposed. It therefore requires considerable testing to come up with sound conclusions. Even then, the conclusions are likely to depend on the type of problem to which SA is applied, and in some cases, to the problem instance. Some studies making general claims about SA can only be applied to particular implementation schemes. However, some general remarks can be made.

7.1. Comparison with repeated descent algorithm

Studies of a number of different problems have shown that SA can give significantly better results than repeating a descent algorithm using different random starting positions (and the same neighbourhood structure) and taking the best solution in the same amount of computing time. This is shown very clearly for a graph partitioning problem by Johnson et al. (1987), and for a travelling salesman problem by van Laarhoven (1988). Lundy and Mees (1986) construct an exaple where the expected time for SA to find the optimum is linear, but where for the repeated descent algorithm the time is quadratic in the size of the problem.

Even though the superior performance of SA has been observed for several different problems, this result does depend on the problem and neighbourhood structure used. This is clear because if there are no local optima, but only a global optimum, a single run of the descent algorithm will find it more quickly than using SA.

7.2. Comparison with problem specific algorithms

There are many combinatorial problems for which special algorithms have already been developed. The performance of SA in comparison with these algorithms seems to depend on the problem under consideration. Some of the most thorough testing has been carried out by Johnson et al. (1987, 1988) who find that for the graph partitioning problem, SA outperforms the classicial Kernighan and Lin algorithms in both quality and speed for certain types of random graphs. For graph colouring problems, SA can find very good quality solutions, though at the expense of very large run times. For the travelling salesman problem, SA is outperformed in both quality and speed by the Lin–Kernighan algorithm which is based on k-opt neighbourhood moves, where the algorithm decides dynamically on the value of k. It is perhaps not surprising that an algorithm like SA which is very generally applicable does not always compete with algorithms specially designed to exploit features of the problem under consideration.

7.3. Comparison of different cooling schedules

Some computational experiments have been done to compare the performance of SA using different cooling schedules, particularly by Johnson et al. (1987) and van Laarhoven (1988). Similar performances were obtained from different types of cooling schedule, once the parameters had been chosen to produce good quality results.

On the problems tested, these researchers did not find that cooling schedules with feedback performed any better than those without. It seems likely that the advantages of cooling schedules with feedback are most evident when the problem has a structure which gives rise to 'phase transitions' as described by Kirkpatrick et al. (1983).

Whatever cooling schedule is chosen, it is important not to spend too long at high temperatures, where most neighbourhood moves are accepted, as this can waste running time. It is also important not to spend too long at very low temperatures, where most neighbourhood moves are rejected. At the end of the algorithm it may be worth checking that at least a local optimum has been obtained.

7.4. Comparison with other general heuristic methods

Some results have been published which compare SA with another general heuristic for specific problems. For example, Hertz and de Werra (1987) show that their tabu search technique for graph colouring problems gives better results than their implementation of SA. On the other hand, Bland and Dawson (1989) obtained better results with their implementation of SA for layout optimisation problems than an intitial version of tabu search. At this stage it is perhaps premature to make relative judgements about these and other general heuristics, since SA and other methods are relatively recent ideas, and research is still continuing to find the best way of implementing them for different types of problem.

7.5. *Practical and scientific problems*

SA has been applied to many problems where no problem-specific algorithms were available. These include problems of VLSI design and molecular structure. SA has in general been able to provide good quality solutions, in some cases improving the best known solutions (e.g. for the football pool problem, see Wille, 1987; van Laarhoven, Aarts, van Lint and Wille, 1988). However for some problems the reported running time is very long, which is why some of the variants of SA described in the last section may be needed in practice.

8. Conclusions

Considering SA as a heuristic algorithm for obtaining good, though not necessarily optimal solutions, to optimisation problems, it has several attractive features. First of all, it is very easy to implement. Once a neighbourhood structure has been devised, the SA algorithm only occupies a few lines of code. Secondly it can be generally applied to a wide range of problems. In principle any combinatorial optimisation problem can be tackled using SA provided a relevant neighbourhood structure is proposed. Thirdly SA can provide high quality solutions to many problems. However this may be at the expense of high run times, particularly when compared to a problem specific algorithm. This means that SA must not be regarded as a panacea, capable of simply solving any optimisation problem. Care is needed to devise an appropriate neighbourhood structure and cooling schedule to obtain an efficient algorithm.

For the OR practitioner, most models derived from real situations have special features which make it less likely that a problem specific algorithm will be available. Extra constraints can often be handled satisfactorily by the use of penalty functions; indeed the penalty function approach may be preferred if the constraints are soft rather than hard. In many practical contexts it is also acceptable to obtain several good solutions to an to an optimisation problem, rather than an optimal solution where that would require a much greater programming or computing effort. Eglese and Rand (1987), and Wright (1989) are examples of case-orientated papers involving SA.

Thus the features of SA which have been described make SA a very useful tool for the OR practitioner.

References

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

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

Alspector, J., and Allen, R.B. (1987), "A neuromorphic VLSI learning system", in: P. Losleben (ed.), *Advanced Research in VLSI*, MIT Press, Cambridge, MA, 313–349.

Anily, S., and Federgruen, A. (1987), "Simulated annealing methods with general acceptance probabilities", *Journal of Applied Probability* 24, 657–667.

Bland, J.A., and Dawson, G.P. (1989), "Tabu search applied to layout optimisation", Report, Department of Maths, Stats and O.R, Trent Polytechnic, UK, presented at CO89, Leeds, July 1989.

Casotto, A., Romeo, F., and Sangiovanni-Vincentelli, A.L. (1987), "A parallel simulated annealing algorithm for the placement of macro-cells", *IEEE Transactions on Computer-Aided Design* 6, 838–847.

Cerny, V. (1985), "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm", *Journal of Optimisation Theory and Applications* 45, 41–51.

Chams, M., Hertz, A., and de Werra, D. (1987), "Some experiments with simulated annealing for colouring graphs", *European Journal of Operational Research* 32, 260–266.

Collins, N.E., Eglese, R.W., and Golden, B.L. (1988), "Simulated annealing—An annotated bibliography", *American Journal of Mathematical and Management Sciences* 8, 209–307.

Connolly, D.T. (1987). "Combinatorial optimization using simulated annealing", Report, London School of Economics, London, WC2A 2AE, presented at the Martin Beale Memorial Symposium, London, July 1987.

Connolly, D.T. (1988), "An improved annealing scheme for the QAP", *European Journal of Operational Research* 46, 93–100.

Eglese, R.W., and Rand, G.K. (1987), "Conference seminar timetabling", *Journal of the Operational Research Society* 38, 591–598.

Faigle, U., and Schrader, R. (1988), "On the convergence of stationary distributions in simulated annealing algorithms", *Information Processing Letters* 27, 189–194.

Farhat, N.H. (1987), "Optoelectric analogs of self-programming neural nets: Architecture and methodologies for implementing fast stochastic learning by simulated annealing", *Applied Optics* 26, 5093–5103.

Glover, F., and Greenberg, H.J. (1989), "New approaches for heuristic search: A bilateral linkage with artificial intelligence", *European Journal of Operational Research* 39, 119–130.

Greene, J.W., and Supowit, K.J. (1986), "Simulated annealing without rejected moves", *IEEE Transactions on Computer-Aided Design* 5, 221–228.

Grover, L.K. (1986), "A new simulated annealing algorithm for standard cell placement", *Proc. IEEE International Conference on Computer-Aided Design*, Santa Clara, 378–380.

Hajek, B. (1988), "Cooling schedules for optimal annealing", *Mathematics of Operations Research* 13, 311–329.

Hertz, A., and de Werra, D. (1987), "Using tabu search techniques for graph colouring", *Computing* 39, 345–351.

Jerrum, M., and Sinclair, A. (1988), "Approximating the permanent", Internal report CSR-275-88, Department of Computer Science, University of Edinburgh.

Johnson, D.S., Aragon, C.R., McGeogh, L.A., and Schevon, C. (1987), "Optimization by simulated annealing: An experimental evaluation, Part I". AT&T Bell Laboratories, Murray Hill, NJ, preprint.

Johnson, D.S, Aragon, C.R., McGeogh, L.A., and Schevon, C. (1988), "Optimization by simulated annealing: An experimental evaluation, Part II", AT&T Bell Laboratories, Murray Hill, NJ, preprint.

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

Laarhoven, P.J.M. van (1988), "Theoretical and computational aspects of simulated annealing", PhD thesis, Erasmus University, Rotterdam.

Laarhoven, P.J.M. van, and Aarts, E.H.L. (1987), *Simulated Annealing: Theory and Applications*, Reidel, Dordrecht.

Laarhoven, P.J.M. van, Aarts, E.H.L., Lint, J.H. van, and Wille, L.T. (1988), "New upper bounds for the football pool problem for 6, 7 and 8 matches", *Journal of Combinatorial Theory A*, in press.

Lundy, M., and Mees, A. (1986), "Convergence of an annealing algorithm", *Mathematical Programming* 34, 111–124.

Maffioli, F. (1987), "Randomized heuristics for NP-hard problems", in: Andreatta, G., Mason, F., and Serafini, P. (eds.), *Advanced School on Stochastics in Combinatorial Optimisation*, World Scientific, Singapore, 76–93.

Matsuo, H., Suh, C.J., and Sullivan, R.S. (1988), "A controlled search simulated annealing method for the general jobshop scheduling problem", Working paper #03-04-88, Department of Management, The University of Texas at Austin, Austin.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953), "Equation of state calculations by fast computing machines", *Journal of Chemical Physics* 21, 1087–1092.

Mitra, D., Romeo, F., and Sangiovanni-Vincentelli, A.L. (1986), "Convergence of finite-time behaviour of simulated annealing", *Advances in Applied Probability* 18, 747–771.

Romeo, F., and Sangiovanni-Vincentelli, A.L. (1985), "Probabilistic hill climbing algorithms: Properties and applications", *Proc. Chapel Hill Conference on VLSI*, Chapel Hill, NC, 393–417.

Sasaki, G.H., and Hajek, B. (1988), "The time complexity of maximum matching by simulated annealing", *Journal of the ACM* 35, 387–403.

Tovey, C.A. (1988), "Simulated simulated annealing", *American Journal of Mathematical and Management Sciences* 8, 389–407.

Wille, L.T. (1987), "The football pool problem for 6 matches: A new upper bound obtained by simulated annealing", *Journal of Combinatorial Theory A* 45, 171–177.

Wright, M.B. (1989), "Applying stochastic algorithms to a locomotive scheduling problem", *Journal of the Operational Research Society* 40, 187–192.