Simulated Annealing

1. What is Simulated Annealing?

Simulated Annealing (SA) is motivated by an analogy to annealing in solids. The idea of SA comes from a paper published by Metropolis etc al in 1953 [Metropolis, 1953). The algorithm in this paper simulated the cooling of material in a heat bath. This is a process known as annealing.
If you heat a solid past melting point and then cool it, the structural properties of the solid depend on the rate of cooling. If the liquid is cooled slowly enough, large crystals will be formed. However, if the liquid is cooled quickly (quenched) the crystals will contain imperfections.
Metropolis's algorithm simulated the material as a system of particles. The algorithm simulates the cooling process by gradually lowering the temperature of the system until it converges to a steady, *frozen* state.

In 1982, Kirkpatrick et al (Kirkpatrick, 1983) took the idea of the Metropolis algorithm and applied it to optimisation problems. The idea is to use simulated annealing to search for feasible solutions and converge to an optimal solution.
Simulated annealing is described in many textbooks. If you want an easy to follow description, I would recommend (Dowsland, 1995). Not only is the description good, but it contains many references for the interested student. Much of this text is based on (Dowsland, 1995).

2. Simulated Annealing versus Hill Climbing

As we have seen in previous lectures, hill climbing suffers from problems in getting stuck at local minima (or maxima). We could try to overcome these problems by trying various techniques.

- We could try a hill climbing algorithm using different starting points.
- We could increase the size of the neighbourhood so that we consider more of the search space at each move. For example, we could try 3-opt, rather than a 2-opt move when implementing the TSP.

Unfortunately, neither of these have proved satisfactory in practice when using a simple hill climbing algorithm.
Simulated annealing solves this problem by allowing worse moves (lesser quality) to be taken some of the time. That is, it allows some uphill steps so that it can escape from local minima.

Unlike hill climbing, simulated annealing chooses a random move from the neighbourhood (recall that hill climbing chooses the best move from all those available – at least when using steepest descent (or ascent)). If the move is better than its current position then simulated annealing will ***always*** take it. If the move is worse (i.e. lesser quality) then it will be accepted based on some probability. This is discussed below.

3. Acceptance Criteria

The law of thermodynamics state that at temperature, *t*, the probability of an increase in energy of magnitude, δE, is given by

$$P(\delta E) = \exp(-\delta E / kt) \qquad (1)$$

Where *k* is a constant known as Boltzmann's constant.

The simulation in the Metropolis algorithm calculates the new energy of the system. If the energy has decreased then the system moves to this state. If the energy has increased then the new state is accepted using the probability returned by the above formula.
A certain number of iterations are carried out at each temperature and then the temperature is decreased. This is repeated until the system freezes into a steady state.

This equation is directly used in simulated annealing, although it is usual to drop the Boltzmann constant as this was only introduced into the equation to cope with different materials. Therefore, the probability of accepting a worse state is given by the equation

$$P = \exp(-c/t) > r \qquad (2)$$

Where

c	=	the change in the evaluation function
t	=	the current temperature
r	=	a random number between 0 and 1

The probability of accepting a worse move is a function of both the temperature of the system and of the change in the cost function. This was shown in the lectures and a spreadsheet is available from the web site for this course which shows the same example that was presented in the lectures.

It can be appreciated that as the temperature of the system decreases the probability of accepting a worse move is decreased. This is the same as gradually moving to a frozen state in physical annealing.
Also note, that if the temperature is zero then only better moves will be accepted which effectively makes simulated annealing act like hill climbing.

4. Relationship between Physical Annealing and Simulated Annealing

In (Dowsland, 1995) a table is presented which shows how physical annealing can be mapped to simulated annealing. It is repeated here

Thermodynamic Simulation	Combinatorial Optimisation
System States	Feasible Solutions
Energy	Cost
Change of State	Neighbouring Solutions
Temperature	Control Parameter
Frozen State	Heuristic Solution

Using these mappings any combinatorial optimisation problem can be converted into an annealing algorithm [(Kirkpatrick, 1983), (Černy, 1985)] by sampling the neighbourhood randomly and accepting worse solutions using equation 2.

5. Implementation of Simulated Annealing

The following algorithm is taken from (Russell, 1995), although you will be able to find similar algorithms in many of the other text books mentioned in the course introduction, as well as in the references at the end of this handout.

Function SIMULATED-ANNEALING(*Problem*, *Schedule*) **returns** a solution state

Inputs : *Problem*, a problem
Schedule, a mapping from time to temperature
Local Variables : *Current*, a node
Next, a node
T, a "temperature" controlling the probability of downward steps

Current = MAKE-NODE(INITIAL-STATE[*Problem*])
For t = 1 **to** ∞ **do**
T = *Schedule*[t]
If T = 0 **then return** C*urrent*
Next = a randomly selected successor of *Current*
ΛE = VALUE[*Next*] – VALUE[*Current*]
if ΛE > 0 **then** *Current* = *Next*
else *Current* = *Next* only with probability exp(-ΛE/T)

We can make several observations about the algorithm.

One of the parameters to the algorithm is the ***schedule***. This is the cooling schedule (see below)
This algorithm assumes that the annealing process will continue until the temperature reaches zero. Some implementations keep decreasing the temperature until some other condition is met. For example, no change in the best state for a certain period of time.
The way this algorithm is presented may hide another aspect of the algorithm that is shown more directly in some other presentations.
That is, a particular phase of the search normally continues at a certain temperature until some sort of equilibrium is reached. This might be a certain number of iterations or it could be until there has been no change in state for a certain number of iterations.
This is all part of the cooling schedule which, in the above algorithm, hides some of these details.

6. The Cooling Schedule

The cooling schedule of a simulated annealing algorithm consists of four components.

- Starting Temperature
- Final Temperature
- Temperature Decrement
- Iterations at each temperature

We will consider these further below

6.1 Starting Temperature

The starting temperature must be hot enough to allow a move to almost any neighbourhood state. If this is not done then the ending solution will be the same (or very close) to the starting solution. Alternatively, we will simply implement a hill climbing algorithm.
However, if the temperature starts at too high a value then the search can move to any neighbour and thus transform the search (at least in the early stages) into a random search. Effectively, the search will be random until the temperature is cool enough to start acting as a simulated annealing algorithm.
The problem is finding the correct starting temperature. At present, there is no known method for finding a suitable starting temperature for a whole range of problems. Therefore, we need to consider other ways.

If we know the maximum distance (cost function difference) between one neighbour and another then we can use this information to calculate a starting temperature.
Another method, suggested in (Rayward-Smith, 1996), is to start with a very high temperature and cool it rapidly until about 60% of worst solutions are being accepted. This forms the real starting temperature and it can now be cooled more slowly.
A similar idea, suggested in (Dowsland, 1995), is to rapidly heat the system until a certain proportion of worse solutions are accepted and then slow cooling can start. This can be seen to be similar to how physical annealing works in that the material is heated until it is liquid and then cooling begins (i.e. once the material is a liquid it is pointless carrying on heating it).

6.2 Final Temperature

It is usual to let the temperature decrease until it reaches zero. However, this can make the algorithm run for a lot longer, especially when a geometric cooling schedule is being used (see below).
In practise, it is not necessary to let the temperature reach zero because as it approaches zero the chances of accepting a worse move are almost the same as the temperature being equal to zero.
Therefore, the stopping criteria can either be a suitably low temperature or when the system is “frozen” at the current temperature (i.e. no better or worse moves are being accepted).

6.3 Temperature Decrement

Once we have our starting and stopping temperature we need to get from one to the other. That is, we need to decrement our temperature so that we eventually arrive at the stopping criterion.
The way in which we decrement our temperature is critical to the success of the algorithm. Theory states that we should allow enough iterations at each temperature so that the system stabilises at that temperature. Unfortunately, theory also states that the number of iterations at each temperature to achieve this might be exponential to the problem size. As this is impractical we need to compromise. We can either do this by doing a large number of iterations at a few temperatures, a small number of iterations at many temperatures or a balance between the two.

One way to decrement the temperature is a simple linear method.
An alternative is a geometric decrement where

$$t = t\alpha$$

where $\alpha < 1$.

Experience has shown that α should be between 0.8 and 0.99, with better results being found in the higher end of the range. Of course, the higher the value of α, the longer it will take to decrement the temperature to the stopping criterion.

6.4 Iterations at each Temperature

The final decision we have to make is how many iterations we make at each temperature.
A constant number of iterations at each temperature is an obvious scheme.

Another method, first suggested by (Lundy, 1986) is to only do one iteration at each temperature, but to decrease the temperature *very* slowly. The formula they use is

$$t = t/(1 + \beta t)$$

where β is a suitably small value. I have entered this formula on a spreadsheet (available from the web site) so that you can play around with the parameters, if you are interested.

An alternative is to dynamically change the number of iterations as the algorithm progresses. At lower temperatures it is important that a large number of iterations are done so that the local optimum can be fully explored. At higher temperatures, the number of iterations can be less.

7. Problem Specific Decisions

The cooling schedule (discussed above) is really having to make decisions about the simulated annealing algorithm. There is another set of decision we have to make, those that are specific to the problem we are trying to solve.

7.1 Cost Function

Presented with a solution to a problem, there must be some way of measuring the quality of the solution. In defining this cost function we obviously need to ensure that it represents the problem we are trying to solve.
It is also important that the cost function can be calculated as efficiently as possible, as it will be calculated at every iteration of the algorithm. However, the cost function is often a bottleneck and it may sometimes be necessary to use

Delta Evaluation : the difference between the current solution and the neighbourhood solution is evaluated.

Partial Evaluation : a simplified evaluation function is used that does not give an exact result but gives a good indication as to the quality of the solution.

If possible, the cost function should also be designed so that it can lead the search. One way of achieving this is to avoid cost functions where many states return the same value. This can be seen as representing a plateau in the search space which the search has no knowledge about which way it should proceed.
In the lecture an example was given that described two possible cost functions for a bin packing problem.

Many cost functions cater for the fact that some solutions are illegal. This is typically achieved using constraints. Two types of constraints are often used.

Hard Constraints : these constraints cannot be violated in a feasible solution. For example, in designing the Jubilee Campus the space allocation cost function could define a hard constraint as not allowing a professor to occupy an office that is smaller than a certain size.

Soft Constraints : these constraints should, ideally, not be violated but, if they are, the solution is still feasible.

When defining constraints they are usually weighted. Hard constraints maybe given a large weighting so that those solutions which violate those constraints have a high cost function. Soft constraints are weighted depending on their importance.
Sometimes the weightings are dynamically changed as the algorithm progresses. This allows, for example, hard constraints to be accepted more readily at the start of the algorithm but later on these solutions would be rejected.

7.2 Neighbourhood Structure

When thinking about your problem one of the first considerations will be how you move from one state to another. This means that you have to define a neighbourhood. That is, when you are in a certain state, what other states are reachable.
In a problem such as space allocation, the neighbourhood function could be defined as swapping a person from one room to another. In a timetabling problem, the neighbourhood function could be defined as moving a lecture from one room to another.

Some results have shown that the neighbourhood structure should be symmetric. That is, if you move from state *i* to state *j* then it must be possible to move from state *j* to state *i*.
It has been found, however, that a weaker condition can hold in order to ensure convergence. That is, that every state must be *reachable* from every other. Therefore, it is important, when thinking about your problem to ensure that this condition is met.

7.3 The solution Space

Common sense tells us that if the search space is as small as possible then the search process will be easier as there are not as many states to explore. Results have shown, not surprisingly, that the number of iterations required to converge to an optimal solution is less for a smaller solution space.

However, if we have defined our cost function such that we allow infeasible solutions (e.g. by violating hard constraints) this obviously increases the search of the search space.

As well as trying to keep the solution space as small as possible, it is also advisable to keep the neighbourhood as small as possible. This allows it to be searched faster but, on the downside, it does cut down the possibility of dramatic improvements.

7.4 Summary

In defining our problem, we have number of conflicting interests. We need a cost function that models our problem but which is easy and fast to calculate. We need a cost function that does not allow infeasible solutions but we sometimes need to explore infeasible areas of the search space to allow us to find a good solution.

We want the solution space to be as small as possible, but we do not want to restrict the search too much. We also need the neighbourhood to be as small as possible but, again, not at the detriment of solution quality.

In summary, our aim is to make the most effective use of each iteration, whilst trying to ensure that we arrive at a good quality solution.

You can appreciate that many of the comments made above apply to other search algorithms as well as to simulated annealing.

You might also like to read about Gray coding representation which is discussed in the Genetic Algorithm handout.

8. Improving Performance

Even though we might find a good set of parameters for our simulated annealing algorithm it is still worthwhile trying to improve the performance (where performance could mean the quality of the solution returned, the time taken by the algorithm etc.) of the algorithm by incorporating other techniques.

8.1 Initialisation

When the simulated annealing algorithm starts it is common to start with a random solution and let the annealing process improve on that. However, it might be better to start with a solution that has been heuristically built. For example, when trying to produce a solution to the TSP problem it could be worthwhile starting with a solution that is built using a greedy search.

8.2 Hybridisation

In recent years many researchers have been looking at ***hybridisation***. You might also hear these techniques referred to as ***memetic algorithms***. What this means is that you combine two (or more – but typically two) search algorithms together. This is a relatively new research area (much of the work being pioneered by researchers at Nottingham) but often a population based search strategy (such as genetic algorithms) are used as the primary search mechanism. As each member of the population is created a local search mechanism is applied to move the individual to a local optimum.
Simulated annealing is just one such search method that can be used as the local search. Other possibilities include hill climbing and tabu search.

An alternative, is to apply a search technique to each solution produced by each iteration of the simulated annealing cycle. For example, it may be possible to apply some heuristic to a solution in order to improve it.

9. Modifications

The algorithm described above is the *classic* algorithm but there are some changes we could consider making.

9.1 Acceptance Probability

In 3 above, the probability of accepting a worse move is based on the physical analogy (based on the Boltzmann distribution). But, could we use another formula? And is there any reason why a different one will not perform better for all, or at least certain, problems?
Firstly, let's consider why we might want to use a different acceptance criteria?
The most obvious reason is that the one proposed does not work. Or at least, it produces results which we suspect might be able to be bettered.
A second, not so obvious reason, is that the exponential calculation is computationally expensive. (Johnson, 1991) found that the acceptance calculation took about one third of the computation time. They experimented with two other acceptance criteria. The first

$P(\delta) = 1 - \delta/t$

approximates the exponential. This formula (as well as the classic acceptance criteria) has been implemented on the spreadsheet, for the interested student.
A better approach was found by building a look-up table of a set of values over the range δ/t. During the course of the algorithm δ/t was rounded to the nearest integer and this value was used to access the look-up table. This method was found to speed up the algorithm by about a third with no significant effect on solution quality.

9.2 Cooling

If you plot a typical cooling schedule you are likely to find that at high temperatures many solutions are accepted. If you start at too high a temperature a random search is emulated and until the temperature cools sufficiently any solution can be reached and could have been used as a starting position.
At lower temperatures, the plot of the cooling schedule, is likely to show that very few worse moves are accepted; almost making simulated annealing emulate hill climbing.
None of this should come as a surprise as this is precisely what we would expect from simulated annealing.

Taking this one stage further, we can say that simulated annealing does most of its work during the middle stages of the cooling schedule. (Connolly, 1990) took this argument one stage further and suggested annealing at a constant temperature. The problem is (again) what is the best temperature. It must be high enough to allow movement but not so low that the system is frozen. But the problem goes even further than that. The optimum temperature will vary from one type of problem to another and also from one instance of a problem to another instance of the same problem.
One solution to this problem is to spend some time searching for the optimum temperature and than stay at that temperature for the remainder of the algorithm. The final temperature is chosen as the temperature that returns the best cost function during the search phase.

9.3 Neighbourhood

The neighbourhood of any move is normally the same throughout the algorithm, but these need not be the case. The neighbourhood could be changed as the algorithm progresses. For example, a cost function based on penalty values can be used to restrict the neighbourhood if the weights associated with the penalties are adjusted as the algorithm progresses.

9.4 Cost Function

Bearing in mind that the cost function is calculated at every iteration of the algorithm, various researchers (e.g. Burke,1999) have shown that the cost function can be responsible for a large proportion of the execution time of the algorithm.
Some techniques have been suggested which aim to alleviate this problem.
In (Rana, 1996) a warehouse scheduling problem (Coors Brewery) is solved using a genetic algorithm, but the same technique could be applied to simulated annealing. The evaluation function is sometimes approximated, which is faster than carrying out an exact evaluation of the given solution. In this case, the evaluation function is a list based simulation of orders progressing through the warehouse. An internal (detailed) simulator is used to verify solutions. This takes about three minutes. An external (coarse) simulator runs in about one tenth of a second and is used to identify potential solutions.
(Ross, 1994) uses delta evaluation on the timetabling problem. Instead of evaluating every timetable they show that, as only small changes are being made between one timetable and the next, it is possible to evaluate just the changes and update the previous cost function using the result of that calculation.
In (Burke, 1999) we use a cache to store solutions (partial and complete) that have already been evaluated so that we can retrieve the value of the cost function from the cache rather than having to go through the evaluation function again.

Concluding Remarks

Using simulated annealing it has been proved that it is possible to converge to the best solution. The problem is, it may take more time than an exhaustive search. So although it may not be practical to find the best solution using simulated annealing, simulated annealing does have this important property which is being used as the basis for future research

Many of the techniques we have looked at are not only applicable to simulated annealing. For example, the performance improvements with regards to the cost function can obviously be used in other evolutionary and meta-heuristic algorithms.

The interested student might be interested in the following up some of the references below, some of which have already been mentioned above.

References

1. Aarts, E.H.L., Korst, J.H.M. 1989. Simulated Annealing and Boltzmann Machines. Wiley, Chichester.
2. E.K. Burke and G. Kendall, "*Evaluation of Two Dimensional Bin Packing Problem using the No Fit Polygon*", Proceedings of the 26th International Conference on Computers and Industrial Engineering, Melbourne, Australia, 15-17 December 1999, pp 286-291
3. Cěrny, V. 1985. A Thermodynamical Approach to the Travelling Salesman Problem; An Efficient Simulation Algorithm. *J. of Optimization Theory and Applic.* 45, 41-55
4. Connolly, D.T. 1990. An Improved Annealing Scheme for the QAP. EJOR, 46, 93-100
5. Dowsland, K.A. 1995. Simulated Annealing. In Modern Heuristic Techniques for Combinatorial Problems (ed. Reeves, C.R.), McGraw-Hill, 1995
6. Hajek, B. 1988. *Cooling Schedules for Optimal Annealing*. Mathematics of Operations Research, vol 13, No. 2, pp311-329
7. Johnson, D.S., Aragon, C.R., McGeoch, L.A.M. and Schevon, C. 1991. Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. Operations Research, 39, 378-406
8. Kirkpatrick, S , Gelatt, C.D., Vecchi, M.P. 1983. *Optimization by Simulated Annealing*. Science, vol 220, No. 4598, pp671-680
9. Lundy, M., Mees, A. 1986. Convergence of an Annealing Algorithm. Math. Prog., 34, 111-124
10. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E. 1953. Equation of State Calculation by Fast Computing Machines. *J. of Chem. Phys.*, 21, 1087-1091.
11. Mitra, D., Romeo, F., Sangiovanni-Vincentelli, A. 1986. *Convergence and Finite Time Behavior of Simulated Annealing*. Advances in Applied Probability, vol 18, pp 747-771
12. A. Rana, A.E. Howe, L.D. Whitley and K. Mathias. 1996. Comparing Heuristic, Evolutionary and Local Search Approaches to Scheduling. Third Artificial Intelligence Plannings Systems Conference (AIPS-96)
13. Rayward-Smith, V.J., Osman, I.H., Reeves, C.R., Smith, G.D. 1996. Modern Heuristic Search Methods. John Wiley & Sons.
14. P. Ross, D. Corne and F. Hsiao-Lan. 1994. Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation. In Y. Davidor, H-P Schwefel and R. Manner (eds) Parallel Problem Solving in Nature, Vol 3, Springer-Verlag, Berlin
15. Russell, S., Norvig, P. 1995. *Artificial Intelligence A Modern Approach*. Prentice-Hall
16. Rutenbar, R.A. 1989. *Simulated Annealing Algorithms : An Overview*. IEEE Circuits and Devices Magazine, Vol 5, No. 1, pp 19-26
17. Van Laarhoven, P.J.M, Aarts, E.H.L. 1987. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing
18. White, S.R. 1984. *Concepts of Scale in Simulated Annealing*. Proceedings International Conference on Computers in Design, pp 646-665