



Simulated Annealing

Shyamji Gupta	9915103028
Mohammad Naeem Khan	9915103049
Saksham Kaushal	9915103061
Mohit	9915103088



Introduction

Simulated Annealing (SA) is a local search algorithm motivated by an analogy to annealing in solids.

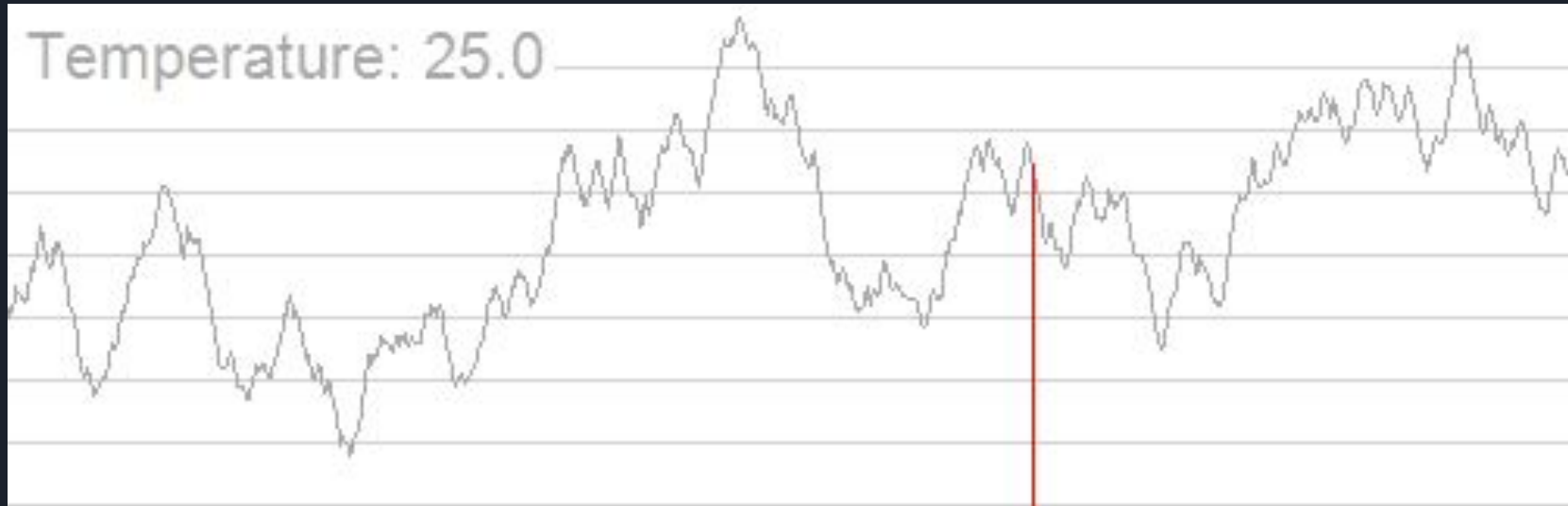
The idea of SA comes from a paper published by Metropolis et. al. in 1953 . The algorithm in this paper simulated the cooling of material in a heat bath. This is a process known as annealing.

In 1982, Kirkpatrick et. al. 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.



Visualization -

Sample Execution of Simulated Annealing



**Adapted from https://en.wikipedia.org/wiki/Simulated_annealing*



Comparison With Hill Climbing

Unlike hill climbing, simulated annealing chooses a random move from the neighbourhood.

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 known as acceptance probability.

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.

If the temperature is zero then only better moves will be accepted which effectively makes simulated annealing act like hill climbing.



Acceptance Probability

Acceptance probability of a move is given by:

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

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.



The Algorithm

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

Inputs : Problem and Schedule - a mapping from time to temperature

Local Variables : Current and Next - denoting corresponding nodes
 T - temperature controlling the probability of downward steps

Current = MAKE-NODE(INITIAL-STATE[Problem])

For t = 1 to infinity do

 T = Schedule[t]

 If T = 0 then return Current

 Next = a randomly selected successor of Current

 E = VALUE[Next] - VALUE[Current]

 if E > 0 then Current = Next

 else Current = Next only with probability $\exp(-E/T)$

**Adapted from Russell, 1995*



Schedule - The Cooling Schedule

This algorithm assumes that the annealing process will continue until either the temperature reaches zero or some equilibrium condition is met.

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, which constitutes 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

Components of Cooling Schedule





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 procedure will simply implement a hill climbing algorithm.

However, if the temperature starts at a very high value then the search can move to any neighbour and act like 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 based on its application and constraints.

An idea suggested by Dowsland, 1995 is to rapidly heat the system until a certain proportion of worse solutions are accepted and then start cooling slowly. This is similar to how physical annealing works; the material is heated until it is liquid and then cooling begins.



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.

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 those at zero temperature.

Therefore, the stopping criteria can either be a suitably low temperature or when the system is “frozen” (at equilibrium) at the current temperature.



Temperature Decrement

We need to decrement our temperature so that we eventually arrive at the stopping criterion.

Enough iterations should be allowed at each temperature so that the system stabilises at that temperature, but theoretically, this might be exponential to problem size.

This can either be done 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.

Another alternative is geometric decrement where,

$$t = t\alpha,$$

where $\alpha < 1$, preferably between 0.8 and 0.99.



Iterations at each Temperature

A constant number of iterations at each temperature is an one possible scheme.

Another method, is to only do one iteration at each temperature, but to decrease the temperature very slowly, i.e.

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

where, β is a suitably small value.

At lower temperatures it is important to do a large number of iterations so that the local exploitation can occur.

At higher temperatures, the number of iterations can be less, which affects exploration.

Problem Specific Decisions





Cost Function

Cost function is used to measure the quality of the solution. Calculation it is often the bottleneck of the program.

Following optimizations can be employed in certain cases :

- 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 of the quality of the solution.

Many cost functions consider rejecting illegal solutions at early stages of algorithm. This is typically achieved using constraints. Carefully implementing hard and soft constraints can provide considerable speedups.



Neighbourhood Structure

Ergodicity is expected in this algorithm in order to ensure convergence, i.e. every state of the system should be reachable from every other state in finite number of steps.

Also, 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 .



Solution Space

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.

However, if cost function is defined such that infeasible solutions can also be explored and evaluated before getting rejected, the search space increases manifold. Thus, constraints need to be taken care of.

A smaller search space can be searched faster but, on the downside, it does cut down the possibility of dramatic improvements.



Ideal Requirements

We need a cost function that models our problem but should be 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.

Applications





Applications


SA has been applied to traditional optimization problems like single machine, flowshop and jobshop scheduling, lot sizing, etc.

It has been applied to non-traditional problems in Operation Research like graph colouring, number partitioning, etc.

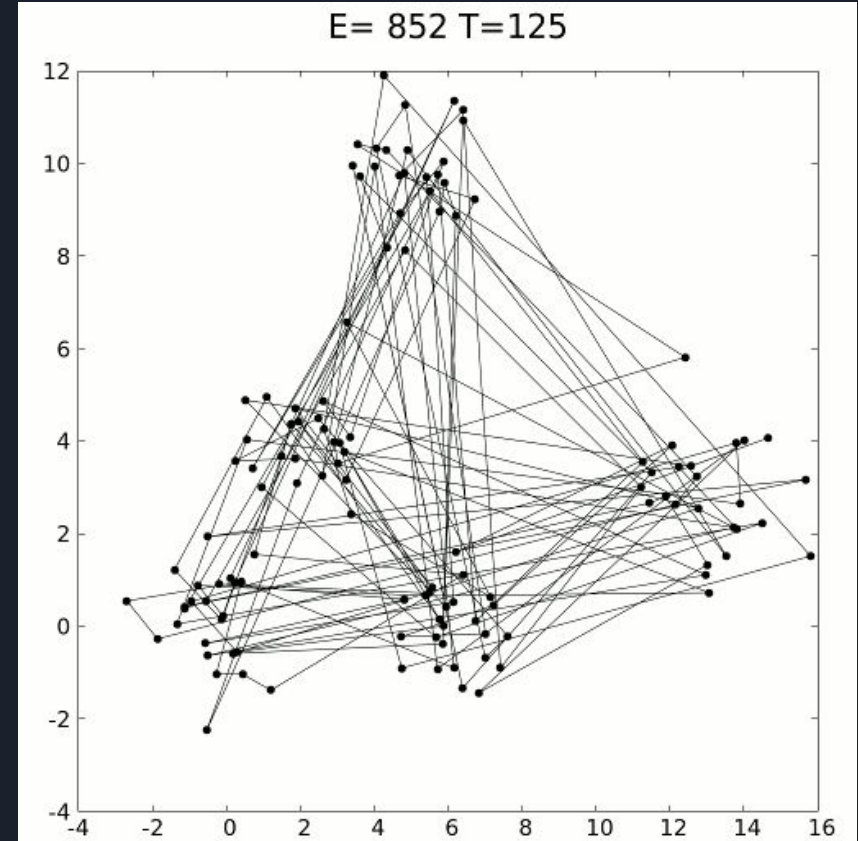
SA has been profoundly used in Big Data Analysis.

SA has been used to solve combinatorial problems like travelling salesman problem, n-queens problem, capacitated vehicle routing problems, knapsack problem etc.

It has been implemented in simulations of population genetics and other biological researches to find approximate global optima.



Visualization - Solving TSP using Simulated Annealing



**Adapted from https://en.wikipedia.org/wiki/Simulated_annealing*



Applications

Most extensive use of SA has probably been done in computational physics and computational neuroscience.

The approximate ground states of simulated large physical systems implementing Ising model, RFIM, spin glass models, etc. can be found conveniently.

SA has found great use in statistical physics and condensed matter physics.

A new method of training neural networks using SA has been recently proposed as an alternative to backpropagation method.

SA has been used in finding efficient solutions to VLSI problems.



Applications

Any optimization problem can be solved using SA using the following analogy

Physical System	Optimization Problem
State (configuration)	Solution
Energy	Cost function
Ground State	Optimal solution
Rapid Quenching	Iteration improvement
Careful Annealing	Simulated annealing

Improving Performance





Initialization

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 beneficial starting with a solution that is built using a greedy search, or any other similar algorithm.



Hybridization

In hybridization, two or more algorithms are combined together.

Often a population based search strategy (such as genetic algorithms) is 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.

Here, SA can be used to perform local search.



Using Follow-Up Algorithms

Since, SA gives us approximate global optimum, there is scope for improvement

Max-min cut theorems and branch and bound algorithms can be used after finding approximate optima using SA, to get true global optima.

Although execution time will be increased, but true global optimate, such as true ground states of crystals, are guaranteed to be achieved based on problem size.

Another alternative would be to use Parallel Tempering Algorithm, which is quite similar to simulated annealing but can be easily and efficiently parallelized.



References

1. AI Methods, Simulated Annealing - by Graham Kendall at Nottingham University
Link to notes : www.cs.nott.ac.uk/~gxx/aim/notes/simulatedannealing.doc
2. <http://katrinaeg.com/simulated-annealing.html>
3. <http://www.cs.cmu.edu/afs/cs.cmu.edu/project/learn-43/lib/photoz/.g/web/glossary/anneal.html>
4. https://en.wikipedia.org/wiki/Simulated_annealing

Thank You!

