Simulated annealing

From Wikipedia, the free encyclopedia

Simulated annealing (SA) is a generic probabilistic metaheuristic for the global optimization problem of locating a good approximation to the global optimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For certain problems, simulated annealing may be more efficient than exhaustive enumeration — provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects, both are attributes of the material that depend on its thermodynamic free energy. Heating and cooling the material affects both the temperature and the thermodynamic free energy. While the same amount of cooling brings the same amount of decrease in temperature it will bring a bigger or smaller decrease in the thermodynamic free energy depending to the rate that it occurs, with a slower rate producing a bigger decrease.

This notion of slow cooling is implemented in the Simulated Annealing algorithm as a slow decrease in the probability of accepting worse solutions as it explores the solution space. Accepting worse solutions is a fundamental property of metaheuristics because it allows for a more extensive search for the optimal solution.

The method was independently described by Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi in 1983,[1] and by Vlado Černý in 1985.[2] The method is an adaptation of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system, invented by M.N. Rosenbluth and published in a paper by N. Metropolis et al. in 1953.[3]

Contents

Overview

In the simulated annealing (SA) method, each point *s* of the search space is analogous to a state of some physical system, and the function *E*(*s*) to be minimized is analogous to the internal energy of the system in that state. The goal is to bring the system, from an arbitrary *initial state*, to a state with the minimum possible energy.

The basic iteration

At each step, the SA heuristic considers some neighbouring state *s'* of the current state *s*, and probabilistically decides between moving the system to state *s'* or staying in state *s*. These probabilities ultimately lead the system to move to states of lower energy. Typically this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted.

The neighbours of a state

The neighbours of a state are new states of the problem that are produced after altering a given state in some particular way. For example, in the traveling salesman problem, each state is typically defined as a particular permutation of the cities to be visited. The neighbours of a permutation are the permutations that are produced for example by interchanging a pair of adjacent cities. The action taken to alter the solution in order to find neighbouring solutions is called a "move" and different moves give different neighbours. These moves usually result in minimal alterations of the solution, as the previous example depicts, in order to help an algorithm optimize the solution to the maximum extent while retaining the already optimum parts of the solution and affecting only the suboptimum parts. In the previous example, the parts of the solution are the city connections.

Searching for neighbours of a state is fundamental to optimization because the final solution will come after a tour of successive neighbours. Simple heuristics move by finding best neighbour after best neighbour and stop when they have reached a solution which has no neighbours that are better solutions. The problem with this approach is that the neighbours of a state are not guaranteed to contain any of the existing better solutions which means that failure to find a better solution among them does not guarantee that no better solution exists. This is why the best solution found by such algorithms is called a local optimum in contrast with the actual best solution which is called a global optimum. Metaheuristics use the neighbours of a state as a way to explore the solutions space and can accept worse solutions in their search in order to accomplish that. This means that the search will not get stuck to a local optimum and if the algorithm is run for an infinite amount of time, the global optimum will be found.[*citation needed*]

Acceptance probabilities

The probability of making the transition from the current state s to a candidate new state s' is specified by an *acceptance probability function* $P(e, e', T)$, that depends on the energies $e = E(s)$ and $e' = E(s')$ of the two states, and on a global time-varying parameter T called the *temperature*. States with a smaller energy are better than those with a greater energy. The probability function P must be positive even when e' is greater than e. This feature prevents the method from becoming stuck at a local minimum that is worse than the global one.

When T tends to zero, the probability $P(e, e', T)$ must tend to zero if $e' > e$ and to a positive value otherwise. For sufficiently small values of T, the system will then increasingly favor moves that go "downhill" (i.e., to lower energy values), and avoid those that go "uphill." With $T = 0$ the procedure reduces to the greedy algorithm, which makes only the downhill transitions.

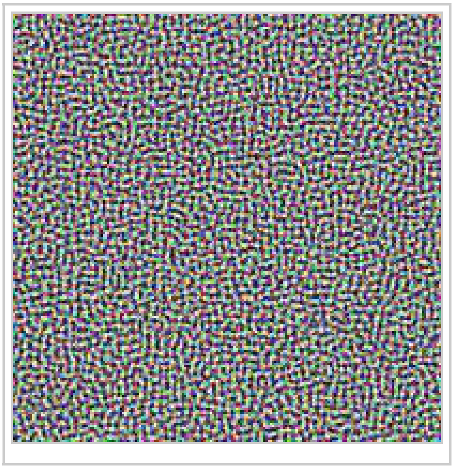
In the original description of SA, the probability $P(e, e', T)$ was equal to 1 when $e' < e$ — i.e., the procedure always moved downhill when it found a way to do so, irrespective of the temperature. Many descriptions and implementations of SA still take this condition as part of the method's definition. However, this condition is not essential for the method to work, and one may argue that it is both counterproductive and contrary to the method's principle.

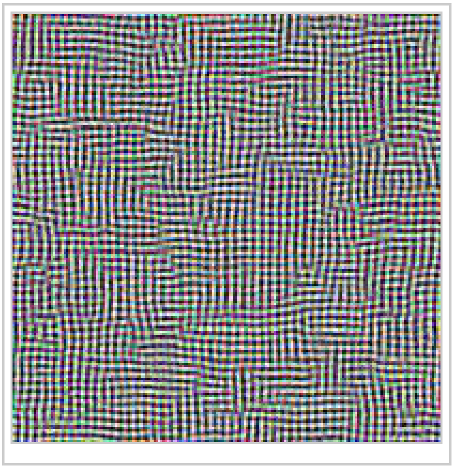
The P function is usually chosen so that the probability of accepting a move decreases when the difference $e' - e$ increases—that is, small uphill moves are more likely than large ones. However, this requirement is not strictly necessary, provided that the above requirements are met.

Given these properties, the temperature T plays a crucial role in controlling the evolution of the state s of the system vis-a-vis its sensitivity to the variations of system energies. To be precise, for a large T, the evolution of s is sensitive to coarser energy variations, while it is sensitive to finer energy variations when T is small.

The annealing schedule

The name and inspiration of the algorithm demand an interesting feature related to the temperature variation to be embedded in the operational characteristics of the algorithm. This necessitates a gradual reduction of the temperature as the simulation proceeds. The algorithm starts initially with T set to a high value (or infinity), and then it is decreased at each step following some *annealing schedule*—which may be specified by the user, but must end with $T = 0$ towards the end of the allotted time budget. In this way, the system is expected to wander initially towards a broad region of the search space containing good solutions, ignoring small features of the energy function; then drift towards low-energy regions that become narrower and narrower; and finally move downhill according to the steepest descent heuristic.





Example illustrating the effect of cooling schedule on the performance of simulated annealing. The problem is to rearrange the pixels of an image so as to minimize a certain potential energy function, which causes similar colours to attract at short range and repel at a slightly larger distance. The elementary moves swap two adjacent pixels. These images were obtained with a fast cooling schedule (left) and a slow cooling schedule (right), producing results similar to amorphous and crystalline solids, respectively.

For any given finite problem, the probability that the simulated annealing algorithm terminates with a global optimal solution approaches 1 as the annealing schedule is extended.[4] This theoretical result, however, is not particularly helpful, since the time required to ensure a significant probability of success will usually exceed the time required for a complete search of the solution space.[*citation needed*]

Pseudocode

The following pseudocode presents the simulated annealing heuristic as described above. It starts from a state `s0` and continues to either a maximum of `kmax` steps or until a state with an energy of `emax` or less is found. In the process, the call `neighbour(s)` should generate a randomly chosen neighbour of a given state s; the call `random()` should return a random value in the range $[0, 1]$. The annealing schedule is defined by the call `temperature(r)`, which should yield the temperature to use, given the fraction `r` of the time budget that has been expended so far.

```
s ← s0; e ← E(s)                                  // Initial state, energy.
sbest ← s; ebest ← e                              // Initial "best" solution
k ← 0                                             // Energy evaluation count.
while k < kmax and e > emax                       // While time left & not good enough:
  T ← temperature(k/kmax)                         // Temperature calculation.
  snew ← neighbour(s)                             // Pick some neighbour.
  enew ← E(snew)                                  // Compute its energy.
  if P(e, enew, T) > random() then                // Should we move to it?
    s ← snew; e ← enew                            // Yes, change state.
  if enew < ebest then                               // Is this a new best?
    sbest ← snew; ebest ← enew                    // Save 'new neighbour' to 'best found'.
  k ← k + 1                                       // One more evaluation done
return sbest                                      // Return the best solution found.
```

Pedantically speaking, the "pure" SA algorithm does not keep track of the best solution found so far: it does not use the variables `sbest` and `ebest`, it lacks the second **`if`** inside the loop, and, at the end, it returns the current state `s` instead of `sbest`. While remembering the best state is a standard technique in optimization that can be used in any metaheuristic, it does not have an analogy with physical annealing — since a physical system can "store" a single state only.

Even more pedantically speaking, saving the best state is not necessarily an improvement, since one may have to specify a smaller `kmax` in order to compensate for the higher cost per iteration and since there is a good probability that `sbest` equals `s` in the final iteration anyway. However, the step `sbest ← snew` happens only on a small fraction of the moves. Therefore, the optimization is usually worthwhile, even when state-copying is an expensive operation.[*citation needed*]

Selecting the parameters

In order to apply the SA method to a specific problem, one must specify the following parameters: the state space, the energy (goal) function `E()`, the candidate generator procedure `neighbour()`, the acceptance probability function `P()`, and the annealing schedule `temperature()` AND initial temperature <init temp>. These choices can have a significant impact on the method's effectiveness. Unfortunately, there are no choices of these parameters that will be good for all problems, and there is no general way to find the best choices for a given problem. The following sections give some general guidelines.

Diameter of the search graph

Simulated annealing may be modeled as a random walk on a *search graph*, whose vertices are all possible states, and whose edges are the candidate moves. An essential requirement for the `neighbour()` function is that it must provide a sufficiently short path on this graph from the initial state to any state which may be the global optimum. (In other words, the diameter of the search graph must be small.) In the traveling salesman example above, for instance, the search space for $n = 20$ cities has $n!$ = 2,432,902,008,176,640,000 (2.4 quintillion) states; yet the neighbour generator function that swaps two consecutive cities can get from any state (tour) to any other state in at most $n(n-1)/2 = 190$ steps.

Transition probabilities

For each edge (s, s') of the search graph, one defines a *transition probability*, which is the probability that the SA algorithm will move to state s' when its current state is s. This probability depends on the current temperature as specified by `temp()`, by the order in which the candidate moves are generated by the `neighbour()` function, and by the acceptance probability function `P()`. (Note that the transition probability is **not** simply $P(e, e', T)$, because the candidates are tested serially.)

Acceptance probabilities

The specification of `neighbour()`, `P()`, and `temperature()` is partially redundant. In practice, it's common to use the same acceptance function `P()` for many problems, and adjust the other two functions according to the specific problem.

In the formulation of the method by Kirkpatrick et al., the acceptance probability function $P(e, e', T)$ was defined as 1 if $e' < e$, and $\exp((e - e')/T)$ otherwise. This formula was superficially justified by analogy with the transitions of a physical system; it corresponds to the Metropolis-Hastings algorithm, in the case where the proposal distribution of Metropolis-Hastings is symmetric. However, this acceptance probability is often used for simulated annealing even when the `neighbour()` function, which is analogous to the proposal distribution in Metropolis-Hastings, is not symmetric, or not probabilistic at all. As a result, the transition probabilities of the simulated annealing algorithm do not correspond to the transitions of the analogous physical system, and the long-term distribution of states at a constant temperature T need not bear any resemblance to the thermodynamic equilibrium distribution over states of that physical system, at any temperature. Nevertheless, most descriptions of SA assume the original acceptance function, which is probably hard-coded in many implementations of SA.

Efficient candidate generation

When choosing the candidate generator `neighbour()`, one must consider that after a few iterations of the SA algorithm, the current state is expected to have much lower energy than a random state. Therefore, as a general rule, one should skew the generator towards candidate moves where the energy of the destination state s' is likely to be similar to that of the current state. This heuristic (which is the main principle of the Metropolis-Hastings algorithm) tends to exclude "very good" candidate moves as well as "very bad" ones; however, the former are usually much more common than the latter, so the heuristic is generally quite effective.

In the traveling salesman problem above, for example, swapping two *consecutive* cities in a low-energy tour is expected to have a modest effect on its energy (length); whereas swapping two *arbitrary* cities is far more likely to increase its length than to decrease it. Thus, the consecutive-swap neighbour generator is expected to perform better than the arbitrary-swap one, even though the latter could provide a somewhat shorter path to the optimum (with $n-1$ swaps, instead of $n(n-1)/2$).

A more precise statement of the heuristic is that one should try first candidate states s' for which $P(E(s), E(s'), T)$ is large. For the "standard" acceptance function P above, it means that $E(s') - E(s)$ is on the order of T or less. Thus, in the traveling salesman example above, one could use a `neighbour()` function that swaps two random cities, where the probability of choosing a city pair vanishes as their distance increases beyond T.

Barrier avoidance

When choosing the candidate generator `neighbour()` one must also try to reduce the number of "deep" local minima — states (or sets of connected states) that have much lower energy than all its neighbouring states. Such "closed catchment basins" of the energy function may trap the SA algorithm with high probability (roughly proportional to the number of states in the basin) and for a very long time (roughly exponential on the energy difference between the surrounding states and the bottom of the basin).

As a rule, it is impossible to design a candidate generator that will satisfy this goal and also prioritize candidates with similar energy. On the other hand, one can often vastly improve the efficiency of SA by relatively simple changes to the generator. In the traveling salesman problem, for instance, it is not hard to exhibit two tours A, B, with nearly equal lengths, such that (0) A is optimal, (1) every sequence of city-pair swaps that converts A to B goes through tours that are much longer than both, and (2) A can be transformed into B by flipping (reversing the order of) a set of consecutive cities. In this example, A and B lie in different "deep basins" if the generator performs only random pair-swaps; but they will be in the same basin if the generator performs random segment-flips.

Cooling schedule

The physical analogy that is used to justify SA assumes that the cooling rate is low enough for the probability distribution of the current state to be near thermodynamic equilibrium at all times. Unfortunately, the *relaxation time*—the time one must wait for the equilibrium to be restored after a change in temperature—strongly depends on the "topography" of the energy function and on the current temperature. In the SA algorithm, the relaxation time also depends on the candidate generator, in a very complicated way. Note that all these parameters are usually provided as black box functions to the SA algorithm. Therefore, the ideal cooling rate cannot be determined beforehand, and should be empirically adjusted for each problem. Adaptive simulated annealing algorithms address this problem by connecting the cooling schedule to the search progress.

Restarts

Sometimes it is better to move back to a solution that was significantly better rather than always moving from the current state. This process is called *restarting* of simulated annealing. To do this we set `s` and `e` to `sbest` and `ebest` and perhaps restart the annealing schedule. The decision to restart could be based on several criteria. Notable among these include restarting based a fixed number of steps, based on whether the current energy being too high from the best energy obtained so far, restarting randomly etc.

Related methods

- Quantum annealing uses "quantum fluctuations" instead of thermal fluctuations to get through high but thin barriers in the target function.
- Stochastic tunneling attempts to overcome the increasing difficulty simulated annealing runs have in escaping from local minima as the temperature decreases, by 'tunneling' through barriers.
- Tabu search normally moves to neighbouring states of lower energy, but will take uphill moves when it finds itself stuck in a local minimum; and avoids cycles by keeping a "taboo list" of solutions already seen.
- Reactive search optimization focuses on combining machine learning with optimization, by adding an internal feedback loop to self-tune the free parameters of an algorithm to the characteristics of the problem, of the instance, and of the local situation around the current solution.
- Stochastic gradient descent runs many greedy searches from random initial locations.
- Genetic algorithms maintain a pool of solutions rather than just one. New candidate solutions are generated not only by "mutation" (as in SA), but also by "recombination" of two solutions from the pool. Probabilistic criteria, similar to those used in SA, are used to select the candidates for mutation or combination, and for discarding excess solutions from the pool.
- Graduated optimization digressively "smooths" the target function while optimizing.
- Ant colony optimization (ACO) uses many ants (or agents) to traverse the solution space and find locally productive areas.
- The cross-entropy method (CE) generates candidates solutions via a parameterized probability distribution. The parameters are updated via cross-entropy minimization, so as to generate better samples in the next iteration.
- Harmony search mimics musicians in improvisation process where each musician plays a note for finding a best harmony all together.
- Stochastic optimization is an umbrella set of methods that includes simulated annealing and numerous other approaches.
- Particle swarm optimization is an algorithm modelled on swarm intelligence that finds a solution to an optimization problem in a search space, or model and predict social behavior in the presence of objectives.
- Intelligent Water Drops (IWD) which mimics the behavior of natural water drops to solve optimization problems

- Parallel tempering is a simulation of model copies at different temperatures (or Hamiltonians) to overcome the potential barriers.

See also

- Adaptive simulated annealing
- Markov chain
- Combinatorial optimization
- Automatic label placement
- Multidisciplinary optimization
- Place and route
- Molecular dynamics
- Traveling salesman problem
- Reactive search optimization
- Graph cuts in computer vision
- Particle swarm optimization
- Intelligent Water Drops

References

1. ^ Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. (1983). "Optimization by Simulated Annealing". *Science* **220** (4598): 671–680. doi:10.1126/science.220.4598.671 (http://dx.doi.org/10.1126%2Fscience.220.4598.671) . JSTOR 1690046 (http://www.jstor.org/stable/1690046) . PMID 17813860 (//www.ncbi.nlm.nih.gov/pubmed/17813860) .
2. ^ Černý, V. (1985). "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". *Journal of Optimization Theory and Applications* **45**: 41–51. doi:10.1007/BF00940812 (http://dx.doi.org/10.1007%2FBF00940812) .
3. ^ Metropolis, Nicholas; Rosenbluth, Arianna W.; Rosenbluth, Marshall N.; Teller, Augusta H.; Teller, Edward (1953). "Equation of State Calculations by Fast Computing Machines". *The Journal of Chemical Physics* **21** (6): 1087. doi:10.1063/1.1699114 (http://dx.doi.org/10.1063%2F1.1699114) .
4. ^ Granville, V.; Krivanek, M.; Rasson, J.-P. (1994). "Simulated annealing: A proof of convergence". *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16** (6): 652–656. doi:10.1109/34.295910 (http://dx.doi.org/10.1109%2F34.295910) .

Further reading

- A. Das and B. K. Chakrabarti (Eds.), *Quantum Annealing and Related Optimization Methods,* Lecture Note in Physics, Vol. 679, Springer, Heidelberg (2005)
- Weinberger, E. (1990). "Correlated and uncorrelated fitness landscapes and how to tell the difference". *Biological Cybernetics* **63** (5): 325–336. doi:10.1007/BF00202749 (http://dx.doi.org/10.1007%2FBF00202749) .
- De Vicente, Juan; Lanchares, Juan; Hermida, Román (2003). "Placement by thermodynamic simulated annealing". *Physics Letters A* **317** (5–6): 415–423. doi:10.1016/j.physleta.2003.08.070 (http://dx.doi.org/10.1016%2Fj.physleta.2003.08.070) .

- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 10.12. Simulated Annealing Methods" (http://apps.nrbook.com/empanel/index.html#pg=549) . *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8. http://apps.nrbook.com/empanel/index.html#pg=549

External links

- Simulated Annealing visualization (http://yuval.bar-or.org/index.php?item=9) A visualization of a simulated annealing solution to the N-Queens puzzle by Yuval Baror.
- Global optimization algorithms for MATLAB (http://biomath.ugent.be/~brecht/downloads.html)
- Simulated Annealing (http://www.heatonresearch.com/articles/64/page1.html) A Java applet that allows you to experiment with simulated annealing. Source code included.
- "General Simulated Annealing Algorithm" (http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=10548&objectType=file) An open-source MATLAB program for general simulated annealing exercises.
- Self-Guided Lesson on Simulated Annealing (http://en.wikiversity.org/wiki/Simulated_Annealing_Project) A Wikiversity project.
- Google in superposition of using, not using quantum computer (http://arstechnica.com/science/news/2009/12/uncertainty-hovers-over-claim-googles-using-quantum-computer.ars) Ars Technica discusses the possibility that the D-Wave computer being used by google may, in fact, be an efficient SA co-processor
- Minimizing Multimodal Functions of Continuous Variables with Simulated Annealing (http://www.netlib.org/opt/simann.f) A Fortran 77 simulated annealing code.

Retrieved from "http://en.wikipedia.org/w/index.php?title=Simulated_annealing&oldid=522900066"
Categories: Heuristic algorithms | Optimization algorithms and methods | Monte Carlo methods