Simulated annealing overview

Franco Busetti

1 Introduction and background

Note: Terminology will be developed within the text by means of *italics*.

Simulated annealing (SA) is a random-search technique which exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system; it forms the basis of an optimisation technique for combinatorial and other problems.

Simulated annealing was developed in 1983 to deal with highly nonlinear problems. SA approaches the global maximisation problem similarly to using a bouncing ball that can bounce over mountains from valley to valley. It begins at a high "*temperature*" which enables the ball to make very high bounces, which enables it to bounce over any mountain to access any valley, given enough bounces. As the temperature declines the ball cannot bounce so high, and it can also settle to become trapped in relatively small ranges of valleys. A *generating distribution* generates possible valleys or states to be explored. An *acceptance distribution* is also defined, which depends on the difference between the function value of the present generated valley to be explored and the last saved lowest valley. The acceptance distribution decides probabilistically whether to stay in a new lower valley or to bounce out of it. All the generating and acceptance distributions depend on the temperature.

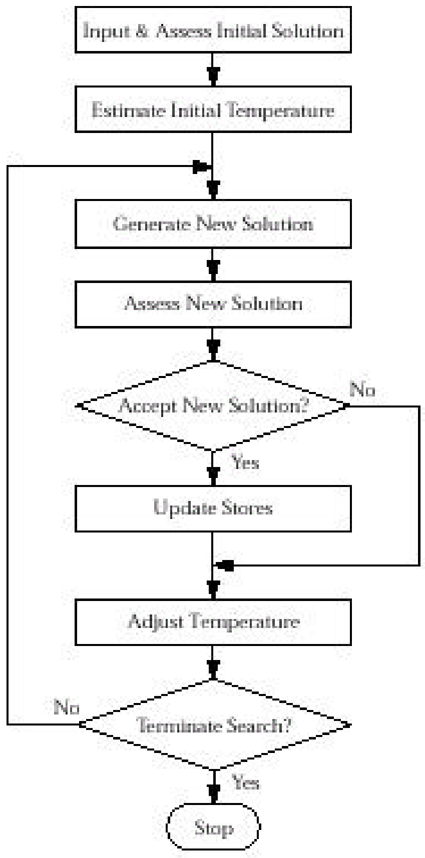
It has been proved that by carefully controlling the rate of cooling of the temperature, SA can find the global optimum. However, this requires infinite time. *Fast annealing* and *very fast simulated re-annealing (VFSR)* or *adaptive simulated annealing (ASA)* are each in turn exponentially faster and overcome this problem.

2 The method

SA's major advantage over other methods is an ability to avoid becoming trapped in local minima. The algorithm employs a random search which not only accepts changes that decrease the objective function f (assuming a minimisation problem), but also some changes that increase it. The latter are accepted with a probability

$$p = \exp(-\delta f / T) \qquad (1)$$

where δf is the increase in f and T is a control parameter, which by analogy with the original application is known as the system *"temperature*" irrespective of the objective function involved. The implementation of the basic SA algorithm is straightforward. The following figure shows its structure:



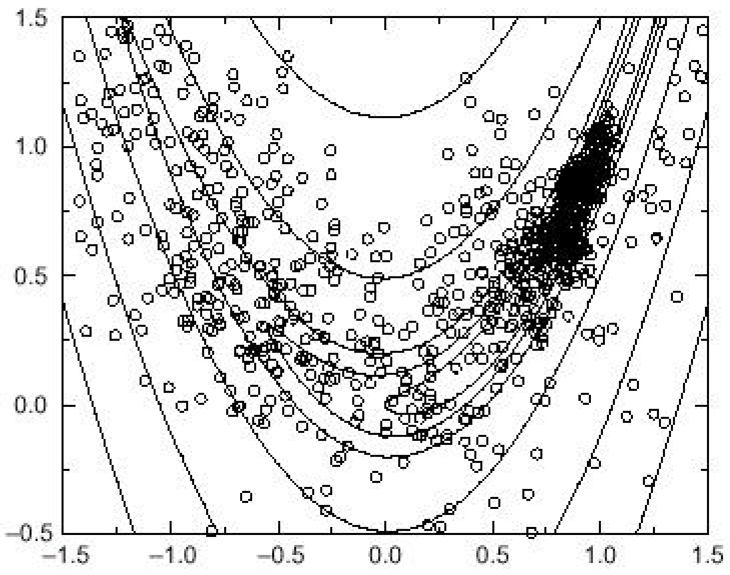
The structure of the simulated annealing algorithm

The following elements must be provided:

- a representation of possible solutions
- a generator of random changes in solutions
- a means of evaluating the problem functions and
- an *annealing schedule* - an initial temperature and rules for lowering it as the search progresses.

These steps are discussed in section 5.

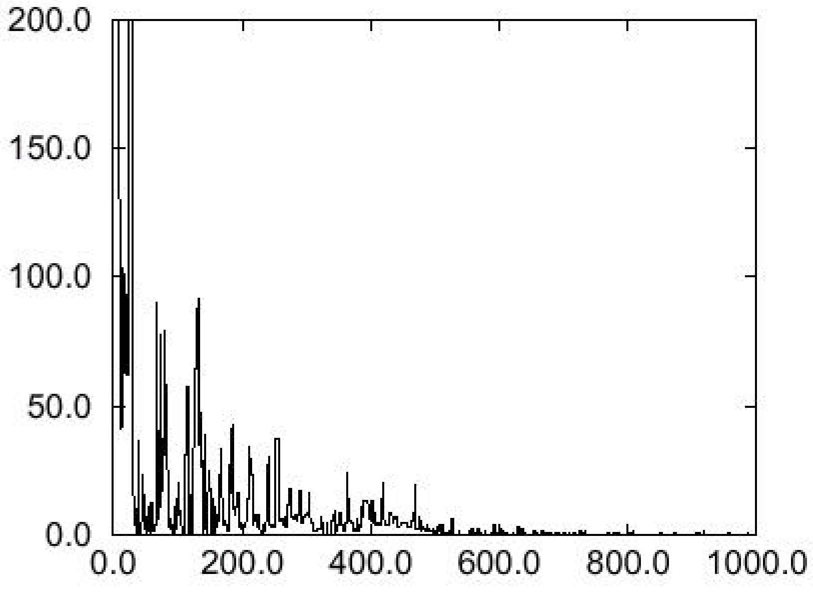
The following figure shows the progress of a SA search on the two-dimensional Rosenbrock function, $f = [(1-x_1)^2 + 100\ (x_2 - x_1^2)]^2$:



Search pattern

Although this function is amenable to solution by more efficient methods, it is useful for purposes of comparison. Each of the solutions accepted in a 1000-trial search is shown (marked by symbols). The algorithm employed the adaptive step size selection scheme of equations (5) and (6) shown later in section 5.1. It is clear that the search is wide-ranging but ultimately concentrates in the neighbourhood of the optimum.

The following figure shows the progress in reducing the objective function for the same search:



Objective function reduction

Initially, when the annealing temperature is high, some large increases in f are accepted and some areas far from the optimum are explored. As execution continues and T falls, fewer uphill excursions are tolerated (and those that are tolerated are of smaller magnitude). The last 40% of the run is spent searching around the optimum. This performance is typical of the SA algorithm.

3 Strengths

Simulated annealing can deal with highly nonlinear models, chaotic and noisy data and many constraints. It is a robust and general technique.

Its main advantages over other local search methods are its flexibility and its ability to approach global optimality.

The algorithm is quite versatile since it does not rely on any restrictive properties of the model.

SA methods are easily "*tuned*". For any reasonably difficult nonlinear or stochastic system, a given optimisation algorithm can be tuned to enhance its performance and since it takes time and effort to become familiar with a given code, the ability to tune a given algorithm for use in more than one problem should be considered an important feature of an algorithm.

4 Weaknesses

Since SA is a metaheuristic, a lot of choices are required to turn it into an actual algorithm.

There is a clear tradeoff between the quality of the solutions and the time required to compute them.

The tailoring work required to account for different classes of constraints and to fine-tune the parameters of the algorithm can be rather delicate.

The precision of the numbers used in implementation is of SA can have a significant effect upon the quality of the outcome.

5 Comparison with other methods

Any efficient optimisation algorithm must use two techniques to find a global maximum: *exploration* to investigate new and unknown areas in the search space, and *exploitation* to make use of knowledge found at points previously visited to help find better points. These two requirements are contradictory, and a good search algorithm must find a tradeoff between the two.

Neural nets

The main difference compared with neural nets is that neural nets *learn* (how to approximate a function) while simulated annealing *searches* for a global optimum. Neural nets are flexible function approximators while SA is an intelligent random search method. The adaptive characteristics of neural nets are a huge advantage in modelling changing environments. However, the power-hungriness of SA limits its use as a real-time application.

Genetic algorithms

Direct comparisons have been made between ASA/VFSR and publicly-available genetic algorithm (GA) codes, using a test suite already adapted and adopted for GA. In each case, ASA outperformed the GA problem. GA is a class of algorithms that are interesting in their own right; GA was not originally developed as an optimisation algorithm, and basic GA does not offer any statistical guarantee of global convergence to an optimal point. Nevertheless, it should be expected that GA may be better suited for some problems than SA.

Random Search
The brute force approach for difficult functions is a random, or an enumerated search. Points in the search space are selected randomly, or in some systematic way, and their fitness evaluated. This is an unintelligent strategy, and is rarely used by itself.

Gradient methods
A number of different methods for optimising well-behaved continuous functions have been developed which rely on using information about the gradient of the function to guide the direction of search. If the derivative of the function cannot be computed, because it is discontinuous, for example, these methods often fail. Such methods are generally referred to as *hillclimbing*. They can perform well on functions with only one peak (*unimodal* functions). But on functions with many peaks, (multimodal functions), they suffer from the problem that the first peak found will be climbed, and this may not be the highest peak. Having reached the top of a local maximum, no further progress can be made.

Iterated Search
Random search and gradient search may be combined to give an *iterated hillclimbing* search. Once one peak has been located, the hillclimb is started again, but with another, randomly chosen, starting point. This technique has the advantage of simplicity, and can perform well *if the function does not have too many local maxima*. However, since each random trial is carried out in isolation, *no overall picture of the "shape" of the domain is obtained*. As the random search progresses, it continues to allocate its trials evenly over the search space. This means that it will still evaluate just as many points in regions found to be of low fitness as in regions found to be of high fitness.

Both SA and GAs, by comparison, start with an initial random population, *and allocate increasing trials to regions of the search space found to have high fitness*. This is a disadvantage if the maximum is in a small region, surrounded on all sides by regions of low fitness. This kind of function is difficult to optimise by any method, and here the simplicity of the iterated search usually wins.

6 Suitability

SA appears rapidly to be becoming an algorithm of choice when dealing with financial instruments [1]. Standard nested regression and local-search methods usually are applied to develop hybrid securities, e.g. combining markets in interest rates, foreign exchange, equities and commodities by linking them via options, futures, forwards, and swaps, to increase profits and reduce risks in investments as well as in trading [2].

However, the complexity and nonlinearity of these multivariate systems, and the increasing interest of including information from more sophisticated modelling into trading rules, have called for more sophisticated numerical algorithms. As such, the parameters are a mix of continuous and discrete sets, but these seem to be able to be processed quite smoothly by adaptive simulated annealing (ASA). One of the several strong features of these algorithms is their flexibility in accommodating many *ad hoc* constraints, rules, etc., as well as algebraic models.

The potential uses for SA in stock price modelling may be limited. However, simulated annealing has been reasonably successfully used in the solution of a complex portfolio selection model [3]. The algorithm was able to handle more classes of constraints than most other techniques. Trading constraints were, however, difficult to handle because of the discontinuities they introduced in the space of feasible portfolios.

An example of the power of SA (ASA), coupled with new statistical mechanical modelling techniques, is that interest rates can be fitted to the data much better than in previously published studies [4]. One study has used SA on a set of several econometric problems [5], including cost functions arising in the monetary theory of exchange rate determination, a study of firm production efficiency, and a neural net model which generates chaos reputed to mirror some economic and financial series. The authors demonstrated that their SA algorithm performed better, e.g. at least more reliably finding more optima, than other numerical techniques such as a genetic algorithms and a quasi-Newton algorithm.

The technique appears well suited to asset allocation problems with constraints.

7. Practical implementation

7.1 Solution representation and generation

When attempting to solve an optimisation problem using the SA algorithm, the most obvious representation of the control variables is usually appropriate. However, the way in which new solutions are generated may need some thought. The solution generator should

- introduce small random changes, and
- allow all possible solutions to be reached.

For problems with continuous control values new *trial solutions* can be generated according to the formula:

$$x_{i+1} = x_i + \mathbf{Qu} \qquad (2)$$

where **u** is a vector of random numbers in the range $(-\sqrt{3},+\sqrt{3})$ so that each has zero mean and unit variance, and **Q** is a matrix that controls the step size distribution. In order to generate random steps with a covariance matrix **S**, **Q** is found by solving

$$\mathbf{S} = \mathbf{QQ}^T \qquad (3)$$

by Cholesky decomposition, for example. **S** should be updated as the search progresses to include information about the local topography:

$$\mathbf{S}_{i+1} = (1-\alpha)\,\mathbf{S}_i + \alpha\omega\mathbf{X} \qquad (4)$$

where matrix **X** measures the covariance of the path actually followed and the damping constant α controls the rate at which information from **X** is folded into **S** with weighting ω. One drawback of this scheme is that the solution of equation (3), which must be done every time **S** is updated, can represent a substantial computational overhead for problems with high dimensionality. In addition, because the probability of an increase in the objective function being accepted, given by equation (1), does not reflect the size of the step taken, **S** must be estimated afresh every time the system temperature is changed.

An alternative strategy is to generate solutions according to the formula:

$x_{i+1} = x_i + \mathbf{Du}$ (5)

where **u** is now a vector of random numbers in the range (-1, 1) and **D** is a diagonal matrix which defines the maximum change allowed in each variable. After a successful trial, i.e. after an accepted change in the solution, **D** is updated:

$D_{i+1} = (1-\alpha)\,\mathbf{D}_i + \alpha\omega\mathbf{R}$ (6)

where α and ω perform similar roles and **R** is a diagonal matrix the elements of which consist of the magnitudes of the successful changes made to each control variable. This tunes the maximum step size associated with each control variable towards a value giving acceptable changes.

When using this strategy it is recommended that the probability of accepting an increase in f be changed from that given by equation (1) to:

$p = \exp(-\delta f / Td)$ (7)

where d is the average step size, so that $\delta f / d$ is a measure of the effectiveness of the change made. As the size of the step taken is considered in calculating p, **D** does not need to be adjusted when T is changed.

For problems with integer control variables, the simple strategy whereby new trial solutions are generated according to the formula:

$x_{i+1} = x_i + \mathbf{u}$ (8)

where **u** is a vector of random integers in the range (-1, 1) often suffices.

For combinatorial or permutation optimisation problems, the solution representation and generation mechanism(s) will necessarily be problem-specific.

7.2 Solution evaluation

The SA algorithm does not require or deduce derivative information, it merely needs to be supplied with an objective function for each trial solution it generates. Thus, the evaluation of the problem functions is essentially a "black box" operation as far as the optimisation algorithm is concerned. Obviously, in the interests of overall computational efficiency, it is important that the problem function evaluations should be performed efficiently, especially as in many applications these function evaluations are by far the most computationally intensive activity.

Some thought needs to be given to the handling of *constraints* when using the SA algorithm. In many cases the routine can simply be programmed to reject any proposed changes which result in constraint violation, so that a search of feasible space only is executed.

However, there are two important circumstances in which this approach cannot be followed:

- if there are any equality constraints defined on the system, or

- if the feasible space defined by the constraints is (suspected to be) *disjoint*, so that it is not possible to move between all feasible solutions without passing through *infeasible* space.

In either case the problem should be transformed into an *unconstrained* one by constructing an augmented objective function incorporating any violated constraints as *penalty functions:*

$$f_A(\mathbf{x}) = f(\mathbf{x}) + 1/T\ \mathbf{w}^T \mathbf{c}_V(\mathbf{x}) \qquad (9)$$

where **w** is a vector of nonnegative weighting coefficients and the vector $\mathbf{c}_V$ quantifies the magnitudes of any constraint violations. The inverse dependence of the penalty on temperature biases the search increasingly heavily towards feasible space as it progresses.

7.3 Annealing schedule

Through equation (1) or (7), the annealing schedule determines the degree of uphill movement permitted during the search and is thus critical to the algorithm's performance. The principle underlying the choice of a suitable annealing schedule is easily stated: the initial temperature should be high enough to *"melt"* the system completely and should be reduced towards its *"freezing point*" as the search progresses. Choosing an annealing schedule for practical purposes is something of an art.

The standard implementation of the SA algorithm is one in which homogeneous Markov chains of finite length are generated at decreasing temperatures. The following parameters should therefore be specified:

- an initial temperature T_0
- a final temperature T_f or a *stopping criterion*
- a length for the Markov chains and
- a rule for decrementing the temperature.

Initial Temperature

A suitable initial temperature T_0 is one that results in an average increase of acceptance probability p_0 of about 0.8. In other words, there is an 80% chance that a change which increases the objective function will be accepted. The value of T_0 will clearly depend on the scaling of f and, hence, be problem-specific. It can be estimated by conducting an initial search in which all increases are accepted and calculating the average objective increase observed δf^+. T_0 is then given by:

$$T_0 = -\delta f^+ / \ln(p_0) \qquad (10)$$

Final Temperature

In some simple implementations of the SA algorithm the final temperature is determined by fixing

- the number of temperature values to be used, or
- the total number of solutions to be generated.

Alternatively, the search can be halted when it ceases to make progress. Lack of progress can be defined in a number of ways, but a useful basic definition is

- no improvement (i.e. no new best solution) being found in an entire Markov chain at one temperature, combined with
- the acceptance ratio falling below a given (small) value p_f .

Length of Markov Chains

An obvious choice for L_k, the length of the k-th Markov chain, is a value that depends on the size of the problem, so L_k is independent of k. Alternatively it can be argued that a minimum number of transitions N_{min} should be accepted at each temperature. However, as T_k approaches 0, transitions are accepted with decreasing probability so the number of trials required to achieve N_{min} acceptances approaches 1 . Thus, in practice, an algorithm in which each Markov chain is terminated after

- L_k transitions or
- N_{min} acceptances,

whichever comes first, is a suitable compromise.

Decrementing the Temperature

The simplest and most common temperature decrement rule is:

$$T_{k+1} = \alpha T_k \qquad (11)$$

where α is a constant close to, but smaller than, 1. This *exponential cooling scheme (ECS)* was first proposed with α= 0.95. In a *linear cooling scheme (LCS)* in which T is reduced every L trials:

$$T_{k+1} = T_k - T \qquad (12)$$

The reductions achieved using the two schemes have been found to be comparable, and the final value of f is, in general, improved with slower cooling rates, at the expense of greater computational effort. The algorithm performance depends more on the *cooling rate* $\Delta T/L$ than on the individual values of ΔT and L. Obviously, care must be taken to avoid negative temperatures when using the LCS.

Random number generation

A significant component of an SA code is the random number generator, which is used both for generating random changes in the control variables and for the (temperature dependent) increase-acceptance test. It is important, particularly when tackling large scale problems requiring thousands of iterations, that the random number generator used have good *spectral properties.*

8. Summary

As with genetic algorithms a major advantage of SA is its flexibility and robustness as a global search method. It is a *"weak method"* which does not use gradient information and makes relatively few assumptions about the problem being solved. It can deal with highly nonlinear problems and non-differentiable functions as well as functions with multiple local optima. It is also amenable to parallel implementation. Simulated annealing is a very powerful and important tool in a variety of disciplines.

A disadvantage is that the SA methods are computation-intensive. There exist faster variants of basic simulated annealing, but these apparently are not as quite easily coded and so they are not widely used. More well-documented, user-friendly code (e.g. menu-driven) would help.

As with neural nets and GAs, SA looks highly promising for portfolio optimisation and asset allocation problems, because the driving variables are highly nonlinear, noisy and often chaotic; it appears less appropriate for modelling financial time series.

9. References

1. Ingber, L., 1993, "Simulated annealing: practice versus theory", *Mathl. Comput. Modelling* **18**, 11, 29-57.
2. Deboeck, G. J. [Ed.], "Trading On The Edge", Wiley, 1994.
3. Crama, Y., and M. Schyns, 1999, "Simulated annealing for complex portfolio selection problems."
4. Goffe, W.L., G.D. Ferrier and J. Rogers, 1994, "Global optimisation of statistical functions with simulated annealing", *J. Econometrics* **60** (1/2), 65-100.
5. Ingber, L., M.F. Wehner, G.M. Jabbour and T.M. Barnhill, 1991, "Application of statistical mechanics methodology to term-structure bond-pricing models", *Mathl. Comput. Modelling* **15** (11), 77-98.