

# SIMULATED ANNEALING TECHNIQUES AND OVERVIEW

Daniel Kitchener  
Young Scholars' Program  
Florida State University  
Tallahassee, Florida, USA

## 1. INTRODUCTION

Simulated annealing is a global optimization algorithm modeled after the natural process of crystallization, where a metal is melted and then allowed to cool slowly. If it is cooled too fast, the metal does not reach its lowest possible energy. The metal must be allowed to cool slowly in order to reach its lowest energy state. Similarly, with simulated annealing of a set of data containing a finite number of points, the points are heated (randomized) and then allowed to cool. In cooling, a random change is made to the system, usually either a random exchange or a reversal of a random section of points. The change is accepted or rejected based upon the Metropolis criteria[1]. If the change results in a lower amount of energy, then it is accepted unconditionally. However, to prevent the system from becoming stuck in one of the local minima, uphill moves (ones with a positive change in energy) are accepted according to the Boltzmann probability distribution of  $P(E)=\exp(-\delta E/kt)$  where  $k$  is Boltzmann's constant,  $t$  is the temperature, and  $\delta E$  is the change in energy. Initially, it begins at a number that is chosen by the user. The temperature is then reduced in coordination with a temperature schedule. Usually, the temperature schedule is exponential, meaning  $T_{k+1} = a * T_k$ . This is to regulate the number of uphill climbs. If the temperature is too large, the program accepts uphill climbs too readily, and will

potentially move away from the global minimum. If the temperature is too small, the program is likely to get stuck in a local minimum.

## 2. TRAVELING SALESMAN PROBLEM

The traveling salesman problem is one of the better-known optimization problems. The problem statement goes as follows: a salesman must travel between  $N$  cities. What path must he take so that he can visit all of the cities while traveling the least amount of distance? This is perhaps one of the best demonstrations of the applications of simulated annealing. It is a straightforward optimization problem whose goal is to find the lowest-energy configuration of a set of data.

Using the algorithm described in Numerical Recipes [2], the implementation of simulated annealing for this problem is relatively simple. First, the map is initialized, having a randomly ordered array of  $N$  cities numbered  $i=1..N$ , having the coordinates  $(x_i, y_i)$ . Then, the array is randomly rearranged. There are two methods of rearranging the array: reversal and cut-and-paste. The first method involves taking a random section of path, reversing its order, and inserting it back into the path (see figure 2-1). The cut-and-paste method involves simply taking a section of path and changing its location in the path (see figure 2-1).

After a re-arrangement is made, the system energy is evaluated. The re-arrangement is accepted or rejected according to the Metropolis criteria. Then, the temperature is reduced according to the annealing schedule. T is reduced after either 10N successful re-arrangements or 100N re-arrangements, depending upon which of the two criteria comes first. After a certain number of unsuccessful reconfigurations, the program is terminated.

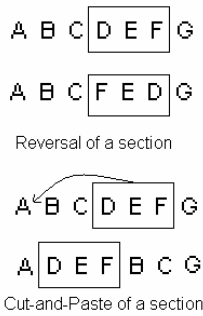


Figure 2-1: Reconfiguration methods

The optimal distance was found to be directly proportional to the square root of the number of cities. The relationship is expressed by \_\_\_ as:

$$0.749 = D_{opt}/\sqrt{N}$$

In experimentation using the Numerical Recipes code [10], the constant was found to be 0.828206. This difference is most likely due to the fact that the randomization function used by the Numerical Recipes code is not truly random, as it uses and manipulates a seed. However, the clear exponential relationship of the number of cities to the distance is seen in the graph below:

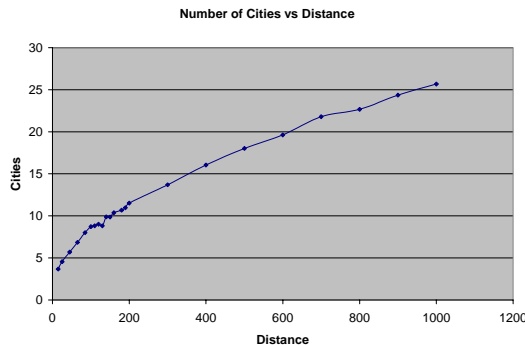


Figure 2-1: Number of Cities vs Distance

With any reasonable number of cities, the constant stays around 0.828206. The test data found for the number of cities compared to the optimal distance is shown in the following chart:

NCITIES	Distance	Constant
15	3.676583	0.94929
25	4.547462	0.909492
45	5.693996	0.848811
65	6.851365	0.849807
85	8.0063	0.868405
100	8.718895	0.87189
110	8.817994	0.840763
120	9.009679	0.822467
130	8.817994	0.773389
140	9.881117	0.835107
150	9.863593	0.805359
160	10.369942	0.819816
180	10.676188	0.795756
190	10.96799	0.795702
200	11.515588	0.814275
300	13.696528	0.790769
400	16.045284	0.802264
500	18.012543	0.805545
600	19.623619	0.801131
700	21.786695	0.82346
800	22.664007	0.801294
900	24.358061	0.811935
1000	25.678366	0.812021
average:		0.828206

Chart 2-1: Distance vs. number of cities, with calculated constant values

By this calculation, the optimal distance for a map of N cities is:

$$D_{opt} = 0.828206\sqrt{N}$$

### 3. SIMULATED ANNEALING OF CONTINUOUS FUNCTIONS

Perhaps one of the biggest challenges that simulated annealing algorithms face is the optimization of continuous functions in N-space. One of the biggest problems is the fact that continuous functions, even within the interval of a finite  $[a,b]$ , have an infinite number of  $x \in \mathbb{R}$  where  $a \leq x \leq b$  and  $\mathbb{R}$  is the set of real numbers, and thus an infinite number of possibilities of solutions of a

function  $f(x)$ . What continuous simulated annealing does is slowly change the  $x$ -values, slowly zeroing in on the minimum. One particularly hard test of a simulated annealing algorithm is the Rosenbrock test function. The Rosenbrock function is given as:

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 - (1 - x_1)^2$$

While this function has only one minimum, it is considered ill-conditioned and is thus a hard problem to optimize when considered in multi-dimensions. Of the algorithms proposed to solve continuous optimization problems, perhaps one of the better ones has to be the Corana algorithm [2].

#### 4. CORANA CONTINUOUS OPTIMIZATION

The Corana algorithm has several unique features that enable it to optimize more accurately than other algorithms. It has a step vector,  $v$ , which adjusts itself to follow function behavior. This way,  $x$  does not increase too much, and jump over a minimum or maximum, or too little, and thus take more time than necessary to find a minimum or maximum.

When the Corana code was run using the Judge test function[2], it found the accepted global minimum on every run, even with a reasonably large variation of the temperature and temperature change parameters. Only when the beginning temperature got to around  $10^{-5}$  did the program begin to get stuck in local minima. Part of this is due to the fact that the Judge function is comparatively easy to optimize due to its absence of large numbers of local minima, which are ubiquitous in the Rosenbrock function. The Judge function has only 2 minima: one at  $f(.864, 1.23) = 16.0817$  (the global minimum) and one at  $f(2.35, -.319) = 20.9805$ .

#### 5. C++ ANNEALING CODE

In an attempt to further understand annealing of continuous functions, code was written that would attempt to find the global minimum of the Rosenbrock function, which is located at  $f(1,1) = 0$ . The code used is included in Appendix A. The code came reasonably close to finding the global minimum, often coming within .05 of the  $x$ -coordinates of the global minimum when using a starting temperature of  $1e-5$  and a  $dt$  of 0.9.

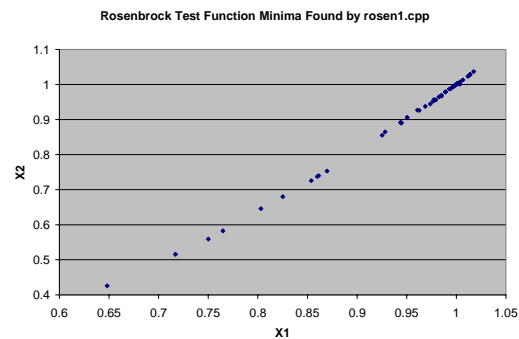


Figure 5-1 Distribution of Minima Found by rosen1.cpp

The code can also easily be adapted to solve any other optimization problem by simply changing the evaluation function. For example, it could easily find the minimum of the Judge function. During testing of the annealing function,  $f(x_1, x_2) = |x_1 + x_2|$  was used, having a global minimum at  $f(0,0) = 0$ . This function is an ideal test function for debugging because of its lack of local minima. Its only minimum is its global minimum.

#### 6. RUNTIME CONSIDERATIONS OF SIMULATED ANNEALING

As with any computer algorithm, runtime is an important factor in considering the value of simulated annealing. Is it better to have a fast algorithm that is, for example, 75% accurate or is it better to have an algorithm that is significantly slower but finds an optimal solution with perhaps 95% accuracy? Through tests of simulated

annealing code for the Traveling Salesman Problem, it was found that runtime is logarithmically related to the number of cities. That is,

$$\log_{10} t = a + b * \log_{10} n$$

a and b are constants, n is the number of cities, and t is the runtime. For the particular code used, it was found that a = -3.7 and b = 1.75, though these values need some adjusting by doing a logarithmic regression on the data using a more accurate utility. These values also vary by computer (runtime is dependent upon clock speed and the processor power).

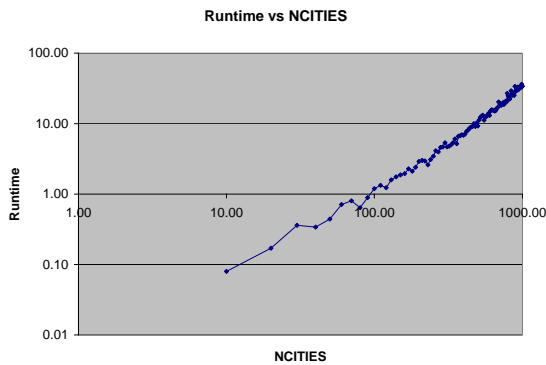


Figure 6-1 Runtime vs NCITIES Logarithmic-Scaled Graph

## 7. AMOEBAS DOWNHILL SIMPLEX METHOD FOR OPTIMIZATION

A simplex can be used to minimize or maximize a multidimensional function, or one that has more than one independent variable. While not efficient, the downhill simplex method is good for optimization problems that require very minimal use of a computer's resources. The algorithm for the downhill simplex method for an N-dimensional space is reasonably straightforward. First, an N-vector of N+1 points is initialized to random values. Take  $P_0$  to be the starting point. Then, the other points are given by

$$P_i = P_0 + \lambda e_i$$

$e_i$ 's are N unit vectors and  $\lambda$  is a constant approximately equal to the length scale of the problem. There are then 4 possible types

of moves for the simplex: reflection, reflection and expansion, contraction, and multiple contraction. These are shown in figure 7-1. When a simplex reaches a minimum, it contracts itself and oozes into the valley. Whenever it must pass through a small gap, it contracts itself in all directions. Whenever possible, the simplex expands itself in order to make larger steps. This algorithm can be used to solve continuous annealing problems. The code used for this particular research was the amoeba code from Numerical Recipes[10]. It works in a way identical to this algorithm.

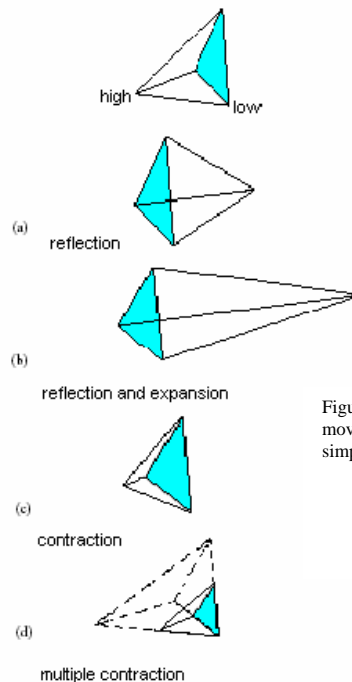


Figure 7.1: Possible moves for the downhill simplex process

## 8. GENETIC ALGORITHMS AND GENETIC PROGRAMMING

Genetic algorithms are optimization algorithms meant to simulate evolution. The specific type that will be discussed here is genetic programming (GP). The GP algorithm begins with a set of programs (organisms) each with a set number of variables (chromosomes) and a fitness function that determines how well each program can survive. In early

experimentation with GP, binary representation of chromosomes was used, though today this is becoming less commonplace. Until a set of termination criteria, set by the computer, is met, there are several possible moves that can be made, each with a certain probability of occurring: mutation, reproduction, and crossover. In mutation, one organism is chosen based upon fitness and its code is altered. With reproduction, the fittest organism is cloned over into the next step. The process of crossover is performed by selecting two individuals based upon fitness and combining half of each into a new organism. Crossover generally has the highest probability of occurring in the program, while mutation has the lowest probability. After a change is made to the population, the population usually undergoes a tournament. Each organism's fitness is evaluated in comparison to the fitness of the other organisms. Every time one organism is more fit than another, it gains a win. Whenever two organisms have the same fitness, they both gain a win. After all of the organisms are compared, the one with the least wins is eliminated and the new offspring is added.

## 9. RANDOMIZATION AND RANDOM NUMBER GENERATORS

A random number is a number whose value is determined by chance. However, computers rely upon sets of rules in order to operate properly. Because of this, they cannot generate truly random. Instead, the numbers that they generate are considered to be pseudo-random. A true random number is one that is generated without any set of rules, which is impossible for a computer using today's technology. Instead, the computers use programs with a set of rules for generating a "random" number. Because simulated annealing is a stochastic process,

the effectiveness or ineffectiveness of the random function that a simulated annealing program uses can greatly influence the outcome of the annealing process. Most random number generators rely upon a seed, a value that is used to initialize the random number generator. From this seed, all other random numbers are derived. Generally, the seed is put into a function that will return a number (the so-called "random" number) and then change the seed. When possible, the higher-order bits of a random number should be used, as these are often more random. For example, if the function `ran()`, generating a number between 0 and 1, is to be used to generate a random number between 1 and 10,

`1+int(10*ran())` should be used, rather than `1+(1000000*ran())%10`. This is because, due to the way the random function generates numbers, the first 2 decimal places are more accurate than the 6<sup>th</sup>, 7<sup>th</sup>, 8<sup>th</sup>, etc. places. Park and Miller propose using a random number generator based upon the formula  $I_{j+1} = (a * I_j) \% m$ , where  $a=7^5$  and  $m=2^{31} - 1$ . For full FORTRAN and C++ code, see Appendix B.

## 10. REFERENCES

1. Areibi, S., Moussa, M. and Abdullah, H. A Comparison of Genetic/Memetic Algorithms and Other Heuristic Search Techniques. *University of Gueleph* (2000).
2. Corana, A., Marchesi, M., Martini, C., and Ridella, S. Minimizing Multimodal Functions of Continuous Variables with the "Simulated Annealing" Algorithm. *ACM Transactions on Mathematical Software*. 13 (1987), 262-280.
3. Goffe, W.L. SIMANN: A Global Optimization Algorithm Using Simulated Annealing. *Studies in Nonlinear Dynamics and Econometrics* 3 (1996), 169-176.
4. Ingber, L. Simulated annealing: Practice versus theory. *Mathl. Comput. Modelling* 18 (1993), 29-57.
5. Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. Optimization by Simulated Annealing. *Science* 220 (1983), 671-680.

6. Kirkpatrick, S. *Journal of Statistical Physics* 34 (1984), 975-986
7. Koza, J.R. Genetic Programming Animated Tutorial. Retrieved July 15, 2003 from the World Wide Web: <http://www.genetic-programming.com/gpflowchart.html>
8. Martin, A.D., and Quinn, K.M. A Review of Discrete Optimization Algorithms. *The Political Methodologist* 7 (1996), 6-10.
9. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller A., and Teller, E. *Journal of Chemical Physics* 21 (1953), 1087-1092.
10. Press, W.H, Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. Numerical Recipes in Fortran 77. Cambridge University Press (1992), 266-277, 402-406, 436-448.
11. Vanderbilt, D. and Louie, S.G. A Monte Carlo Simulated Annealing Approach to Optimization over Continuous Variables. *Journal of Computational Physics* 56 (1984), 259-271.
12. Vecchi, M.P. and Kirkpatrick, S. Global Wiring by Simulated Annealing. *IEEE Transactions on Computer Aided Design CAD-2* (1983 October), 215-222.