Metaheuristic Optimization: Introduction and Simulated Annealing

AE 6310: Optimization for the Design of Engineered Systems

Spring 2017

Dr. Glenn Lightsey

Lecture Notes Developed By Dr. Brian German

Georgia Institute of Technology

Daniel Guggenheim School of Aerospace Engineering

Challenges to Classical Optimization Methods

The optimization methods that we have discussed so far are most suited for:

- ❖ Problems in which all of the design variables are continuous
- ❖ Problems in which there are few local optima
  - ▪ Ideally just “unimodal” problems with one local optimum that is also globally optimal

If our problem has many discrete variables and/or many local optima, we may need another approach!

Metaheuristic Optimization

A category of optimization methods that are intended for these types of problems is called *metaheuristic optimization.*

A *heuristic* is a “rule of thumb” that is devised by (human) intuition or experience based on observations and past experience.

In the context of optimization, examples of heuristics could be:

- “I should restart the algorithm at a different point in the design space to make sure that it was not trapped in a local minimum.”
- “Sometimes, I need to enable the algorithm to ‘search uphill’ to find a better local neighborhood in the design space.”

Metaheuristic Optimization

A *metaheuristic* (or *meta-heuristic*) is:

"…a master strategy that guides and modifies other heuristics to produce solutions **beyond those that are normally generated in a quest for local optimality**. The heuristics guided by such a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule."

Glover, F.W. and Laguna, M., *Tabu Search*, Vol. 1, pp. 17, Springer, 1998.

Metaheuristic Optimization

Metaheuristic optimization methods generally have the following characteristics in common:

- Employ algorithms that are *stochastic*, i.e. that involve selection of random values of design variables. Stochasticity is a key to enabling the methods to find a *global* optimum. However, the methods have more “smarts” than purely random search.

- Generally of “discrete origin,” i.e. design variables are often treated as discrete, even if they are truly continuous. This requires that all variables be discretized in some way.

Metaheuristic Optimization

- Zeroth-order, i.e. they do not use gradients or Hessians
- Inspired by analogies with physics, biology, or other fields
- Require many function calls and large computation time
- Difficult to know *a priori* how to adjust heuristic or metaheuristic parameters to solve a particular problem efficiently
- Must be careful that our metaheuristic algorithm does not require more function calls and time that a grid search or an exhaustive random search!

Metaheuristic Optimization

A note on nomenclature:

Metaheuristic optimization algorithms almost always involve stochastic elements. They could therefore be categorized as a subclass of *stochastic optimization* algorithms.

However, stochastic optimization is broader class that also include problems in which the objective function and the constraints are random variables, i.e. probabilistic optimization problems

Metaheuristic algorithms use stochasticity in the algorithms themselves to solve deterministic optimization problems.

Metaheuristic Optimization Algorithms

There are many types of metaheuristic algorithms. (You can create your own!)

We will discuss the following example algorithms:

- ❖ Simulated annealing
- ❖ Genetic algorithms
- ❖ Particle swarm algorithms

… and maybe a few others along the way.

Metaheuristic Optimization

Most metaheuristic optimization algorithms are naturally formulated for side-constrained optimization problems of the form:

Minimize: $f(\boldsymbol{x})$

Subject to: $\boldsymbol{x_L} \leq \boldsymbol{x} \leq \boldsymbol{x_U}$

More complicated constraints can be handled either by penalty functions, or in some cases, by directly “filtering” designs generated in the stochastic search.

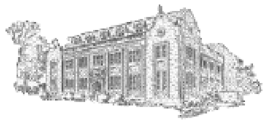
Without penalty functions, equality constraints are challenging!

Simulated Annealing

Simulated annealing is an optimization technique that is inspired by an analogy with the annealing process for metals.

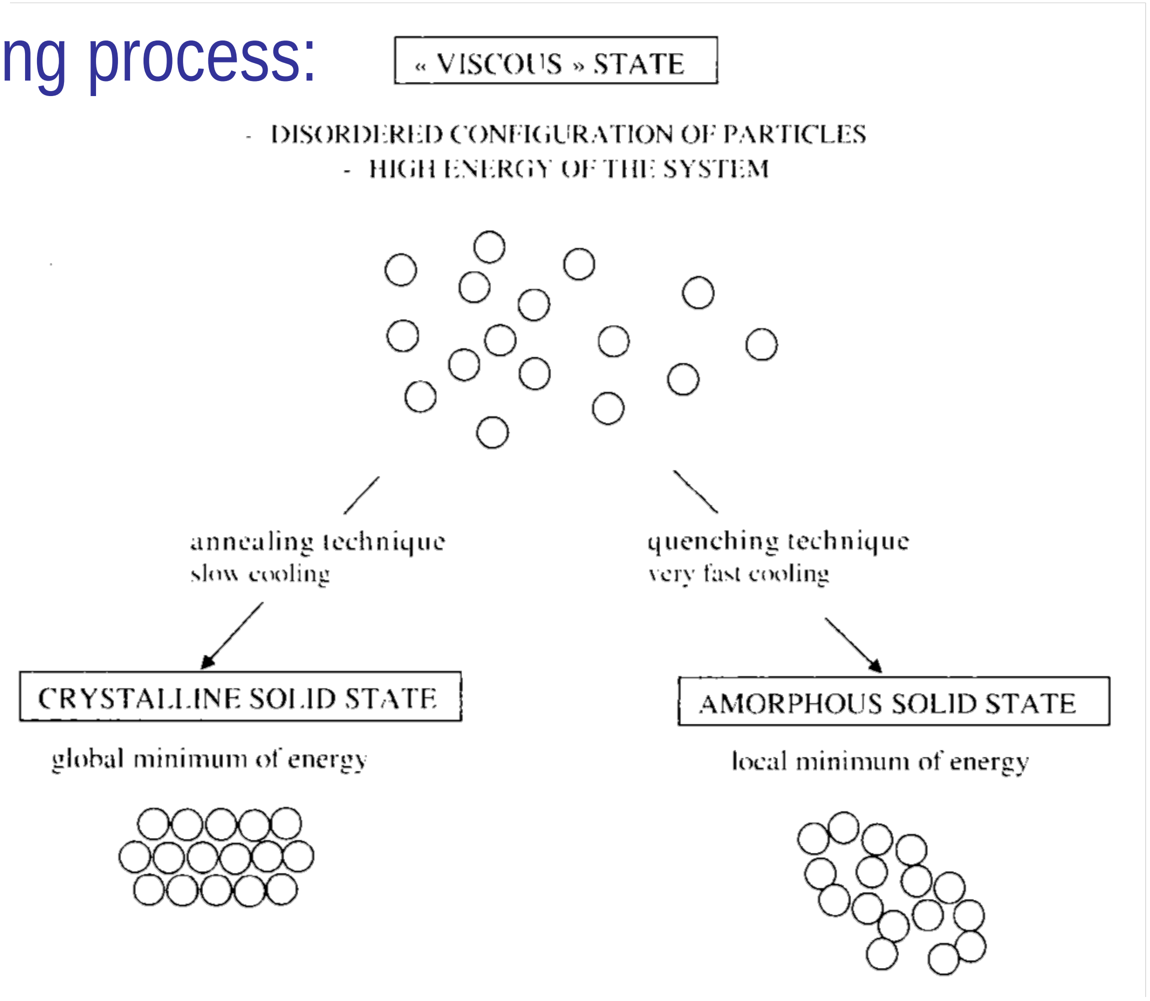
In annealing, a metal is heated and then cooled at a specified rate to alter its crystalline structure and material properties such as ductility.

The outcome of the process is dependent upon the rate at which the temperature is decreased.



Simulated Annealing

The annealing process:



Dreo J., Petrowski, A., Siarry, P., and Taillard, E., *Metaheuristics for Hard Optimization: Methods and Case Studies*, Springer 2006.

Simulated Annealing

Simulated annealing is based on the concept of the Boltzmann probability distribution in the form,

$$p(\Delta E) = \exp\left(-\frac{\Delta E}{kT}\right)$$

where

$\Delta E = E_{new} - E_{old}$

k is Boltzmann's constant

T is the temperature

Boltzmann's distribution describes the distribution of energy in matter at equilibrium at a given temperature.

Simulated Annealing

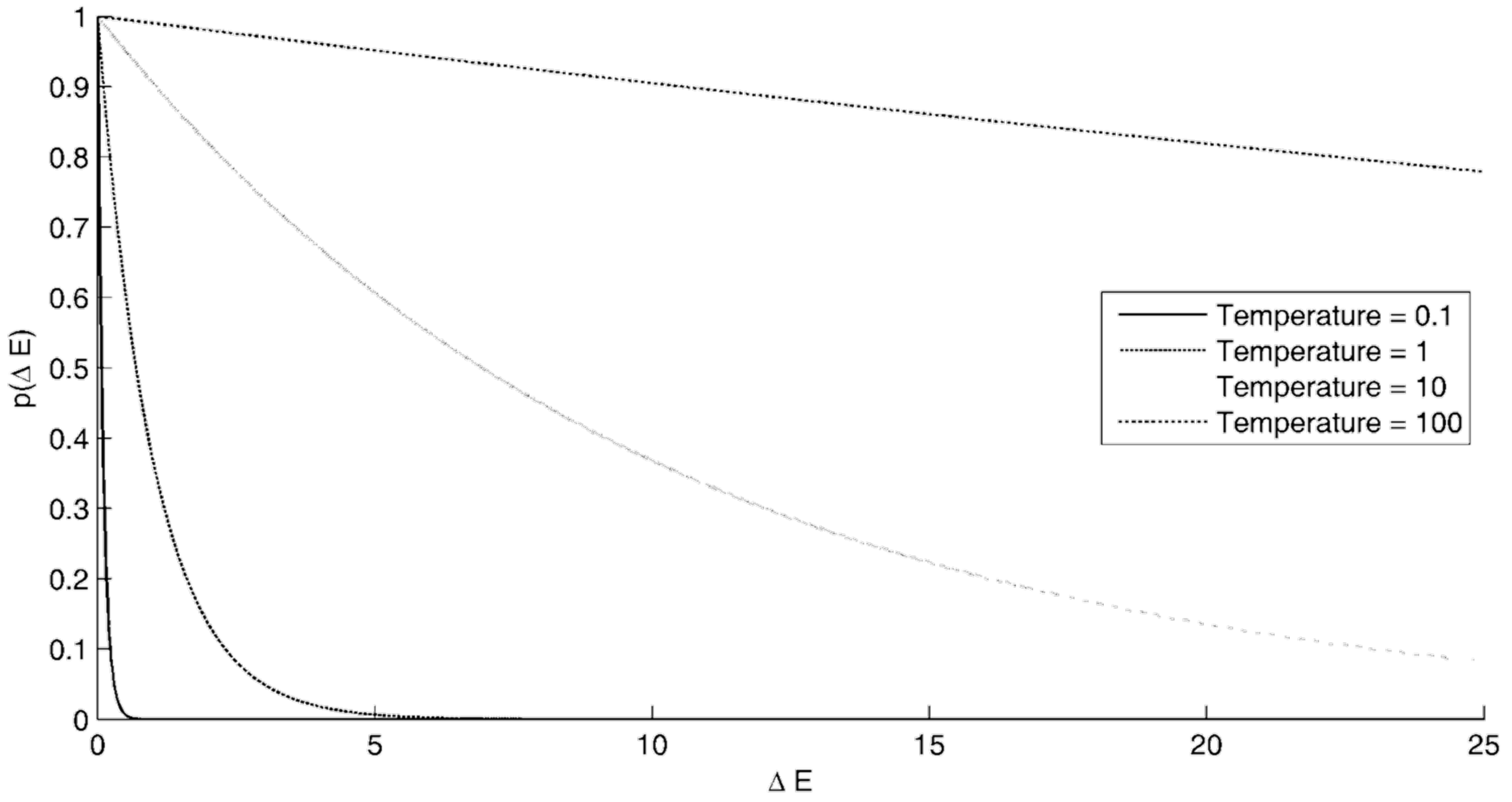
To apply the annealing analogy, we do the following:

- Let E_{old} be the value of the objective function for an initial point
- Let E_{new} be the value of the objective function for a trial point that is in some way chosen to be a "neighbor" to the initial point
- If $E_{new} < E_{old}$, then always accept the trial point as the new initial point
- If $E_{new} \geq E_{old}$, accept the trial point as the new initial point with probability $p(\Delta E) = \exp\left(-\frac{\Delta E}{kT}\right)$

This is called the *Metropolis criterion*.

Simulated Annealing

Boltzmann distribution for $k = 1$ at different temperatures



Simulated Annealing

The Metropolis criterion was originally implemented in the *Metropolis algorithm* in which the process from the previous slide is repeated multiple times for many trial points at a fixed temperature, T.

Eventually, the system reaches a state of equilibrium at this T in which the probability distribution of the accepted points approaches to the Boltzmann distribution.

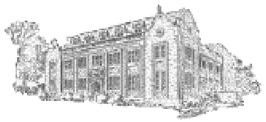
Simulated annealing modifies the Metropolis algorithm by introducing an *annealing schedule* that defines how quickly T is decreased.

Simulated Annealing

The overall concept of the algorithm is as follows:

1. Beginning at an initial T_0, pick an initial set of design variable values and determine E_{old}.
2. Randomly select another "neighboring" point within the design variable space (we will discuss this later) and calculate E_{new}.
3. Implement the Metropolis criterion as follows. Generate a random number U drawn from a uniform distribution in the range (0,1). If $U \leq \exp(-\frac{\Delta E}{T})$, then move to the new point. Always accept new points for which $E_{new} < E_{old}$.

Adapted from Brooks and Morgan, 1995



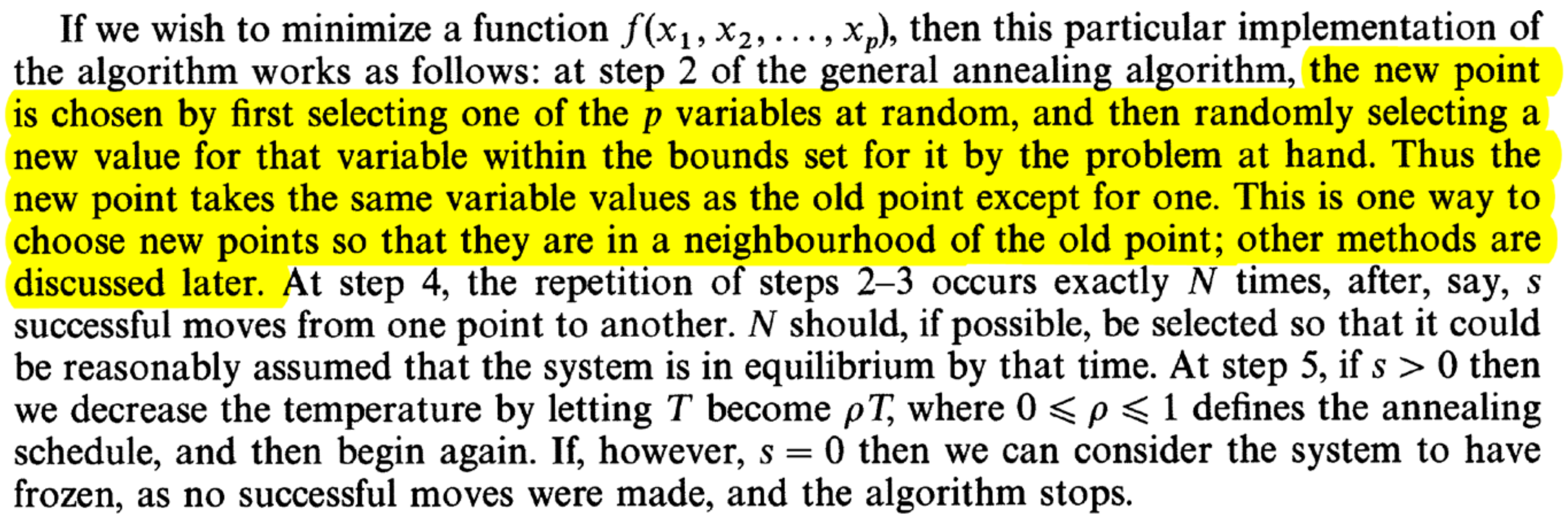
Simulated Annealing

4. Whether the point has moved or not, repeat steps 2-3. At each stage, compare the new point to the previous point until the sequence of points is judged to have reached a state of equilibrium.
5. Once an equilibrium has been achieved for a given T, lower T as defined by the annealing schedule. Then repeat from step 2, with the initial point chosen as the final point from the prior T. Repeat until a convergence stopping criterion is met.

Adapted from Brooks and Morgan, 1995

Simulated Annealing

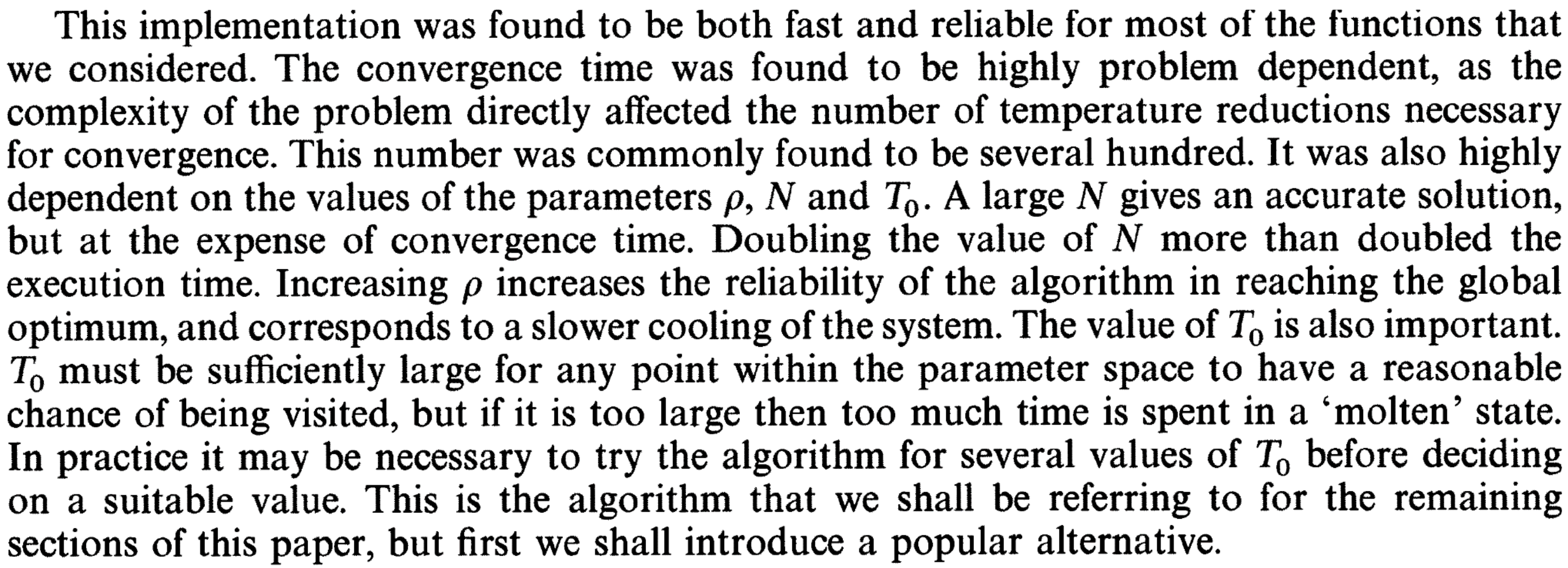
A simple implementation, quoted from (Brooks and Morgan, 1995):



If we wish to minimize a function $f(x_1, x_2, \ldots, x_p)$, then this particular implementation of the algorithm works as follows: at step 2 of the general annealing algorithm, the new point is chosen by first selecting one of the p variables at random, and then randomly selecting a new value for that variable within the bounds set for it by the problem at hand. Thus the new point takes the same variable values as the old point except for one. This is one way to choose new points so that they are in a neighbourhood of the old point; other methods are discussed later. At step 4, the repetition of steps 2–3 occurs exactly N times, after, say, s successful moves from one point to another. N should, if possible, be selected so that it could be reasonably assumed that the system is in equilibrium by that time. At step 5, if $s > 0$ then we decrease the temperature by letting T become ρT, where $0 \leqslant \rho \leqslant 1$ defines the annealing schedule, and then begin again. If, however, $s = 0$ then we can consider the system to have frozen, as no successful moves were made, and the algorithm stops.

Simulated Annealing

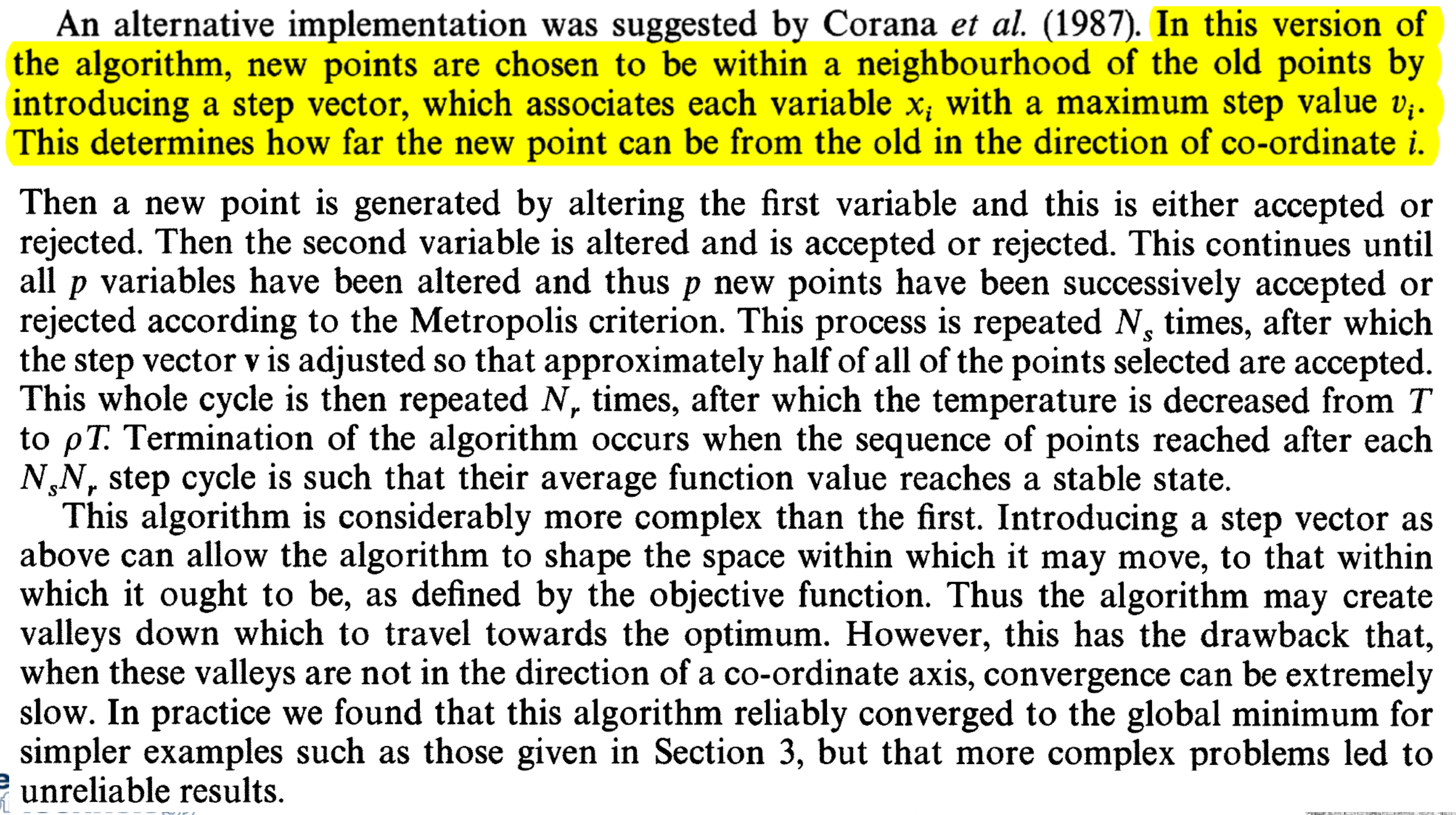
Continuing from (Brooks and Morgan, 1995):



This implementation was found to be both fast and reliable for most of the functions that we considered. The convergence time was found to be highly problem dependent, as the complexity of the problem directly affected the number of temperature reductions necessary for convergence. This number was commonly found to be several hundred. It was also highly dependent on the values of the parameters ρ, N and T_0. A large N gives an accurate solution, but at the expense of convergence time. Doubling the value of N more than doubled the execution time. Increasing ρ increases the reliability of the algorithm in reaching the global optimum, and corresponds to a slower cooling of the system. The value of T_0 is also important. T_0 must be sufficiently large for any point within the parameter space to have a reasonable chance of being visited, but if it is too large then too much time is spent in a 'molten' state. In practice it may be necessary to try the algorithm for several values of T_0 before deciding on a suitable value. This is the algorithm that we shall be referring to for the remaining sections of this paper, but first we shall introduce a popular alternative.

Simulated Annealing

A more complicated implementation, quoted from (Brooks and Morgan, 1995):



An alternative implementation was suggested by Corana *et al.* (1987). In this version of the algorithm, new points are chosen to be within a neighbourhood of the old points by introducing a step vector, which associates each variable x_i with a maximum step value v_i. This determines how far the new point can be from the old in the direction of co-ordinate i.

Then a new point is generated by altering the first variable and this is either accepted or rejected. Then the second variable is altered and is accepted or rejected. This continues until all p variables have been altered and thus p new points have been successively accepted or rejected according to the Metropolis criterion. This process is repeated N_s times, after which the step vector **v** is adjusted so that approximately half of all of the points selected are accepted. This whole cycle is then repeated N_r times, after which the temperature is decreased from T to ρT. Termination of the algorithm occurs when the sequence of points reached after each $N_s N_r$ step cycle is such that their average function value reaches a stable state.

This algorithm is considerably more complex than the first. Introducing a step vector as above can allow the algorithm to shape the space within which it may move, to that within which it ought to be, as defined by the objective function. Thus the algorithm may create valleys down which to travel towards the optimum. However, this has the drawback that, when these valleys are not in the direction of a co-ordinate axis, convergence can be extremely slow. In practice we found that this algorithm reliably converged to the global minimum for simpler examples such as those given in Section 3, but that more complex problems led to unreliable results.