Simulated Annealing Optimization over Continuous Spaces

William H. Press and Saul A. Teukolsky

Citation: Computers in Physics **5**, 426 (1991); doi: 10.1063/1.4823002
View online: https://doi.org/10.1063/1.4823002
View Table of Contents: https://aip.scitation.org/toc/cip/5/4
Published by the American Institute of Physics

Simulated Annealing Optimization Over Continuous Spaces

William H. Press and Saul A. Teukolsky

In this column we combine two previously unrelated techniques into a new and powerful algorithm. The problem to be solved is that of finding the global minimum—or a local minimum that is *almost* as good as the true global minimum—of a function $f(\mathbf{x})$ of N variables $\mathbf{x} = (x_1, x_2, ..., x_N)$. The function $f(\mathbf{x})$ has, possibly, many extraneous local minima. Conventional optimization methods (see, e.g., Chap. 10 of Refs. 1–3) are ineffective here, because they tend to converge to whichever local minimum they first encounter.

The method of simulated annealing[4,5] is a technique that has attracted significant attention as suitable for optimization problems where a desired global extremum is hidden among many, poorer, local extrema. For practical purposes, simulated annealing has effectively "solved" the famous traveling salesman problem of finding the shortest cyclical itinerary for a traveling salesman who must visit each of N cities in turn. The method has also been used successfully for designing complex integrated circuits: Some hundreds of thousands of circuit elements on a silicon substrate can be permuted and assigned to locations that minimize interference among their connecting wires.[6,7]

The notable successes of simulated annealing have been in problems of *combinatorial minimization*, where the space over which a function is defined is not the N-dimensional space of a vector $\mathbf{x}$, but rather, a discrete configuration space, like the set of possible orders of cities, or the set of possible allocations of silicon "real estate" blocks to circuit elements. The number of elements in the configuration space is factorially large, so that they cannot be explored exhaustively.

Optimization in continuous control spaces is in some ways a more challenging problem than combinatorial minimization. In high-dimensional spaces, functions with "long, narrow valleys" are generic. Simulated annealing, as we will see, tries "random" steps; but in a long, narrow valley, almost all random steps are uphill! Furthermore, while combinatorial spaces have an "atomic" minimum step (e.g., a change in the permutation) continuous spaces are prey to sequences of steps that get nowhere, converging instead to undesired limit points. We will see that a subtle modification of the Nelder–Mead downhill simplex method of function minimization[8] provides the means of overcoming these difficulties. Let us start, however, by briefly considering simulated annealing.

At the heart of the method of simulated annealing is an analogy with thermodynamics, specifically with the way that liquids freeze and crystallize, or metals cool and anneal. At high temperatures, the molecules of a liquid move freely with respect to one another. If the liquid is cooled slowly, thermal mobility is lost. The atoms are often able to line themselves up and form a pure crystal that is completely ordered over a distance billions of times the size of an individual atom in all directions. This crystal is the state of minimum energy for this system. The amazing fact is that, for slowly cooled systems, nature is able to find this minimum energy state. In fact, if a liquid metal is cooled quickly or "quenched," it does not reach this state but rather ends up in a polycrystalline or amorphous state having somewhat higher energy.

The essence of the process is *slow* cooling, allowing ample time for redistribution of the atoms as they lose mobility. This is the technical definition of annealing, and it is essential for ensuring that a low energy state will be achieved.

Although the analogy is not perfect, there is a sense in which conventional minimization algorithms correspond to rapid cooling or quenching. In most cases, from the starting point, these go immediately downhill as far as they can go. This, as remarked above, leads to a local, but not necessarily a global, minimum. Nature's own minimization algorithm is based on quite a different procedure. The so-called Boltzmann probability distribution,

$$\text{Prob}(E) \sim \exp(-E/kT), \tag{1}$$

expresses the idea that a system in thermal equilibrium at temperature T has its energy probabilistically distributed among all different energy states E. Even at low temperature, there is a chance, albeit very small, of a system being in a high energy state. Therefore, there is a corresponding chance for the system to get out of a local energy minimum in favor of finding a better, more global, one. In other words, the system sometimes goes *uphill* as well as downhill; but the lower the temperature, the less likely is any significant uphill excursion.

In 1953, Metropolis and co-workers[9] first incorporated these kinds of principles into numerical calculations. Offered a succession of options, a simulated thermodynamic system was assumed to change its configuration from energy E_1 to energy E_2 with probability $p = \exp[-(E_2 - E_1)/T]$. Notice that if $E_2 < E_1$, this probability is greater than unity; in such cases the change is arbitrarily assigned a probability $p = 1$, i.e., the system *always* takes such an option. This general scheme, of always taking a downhill step while *sometimes* taking an uphill step, has come to be known as the Metropolis algorithm.

William H. Press is a professor of astronomy and physics at Harvard University. Saul A. Teukolsky is a professor of physics and astronomy at Cornell University.

To make use of the Metropolis algorithm for other than thermodynamic systems, one must provide the following elements: (1) A description of possible system configurations. (2) A generator of random changes in the configuration; these changes are the "options" presented to the system. (3) An objective function E (analog of energy) whose minimization is the goal of the procedure. (4) A control parameter T (analog of temperature) and an annealing schedule which tells how it is lowered from high to low values, e.g., after how many random changes in configuration is each downward step in T taken, and how large is that step.

The basic ideas of simulated annealing are applicable to optimization problems with continuous N-dimensional control spaces, e.g., finding the (ideally, global) minimum of the function $f(\mathbf{x})$ in the presence of many local minima, where $\mathbf{x}$ is an N-dimensional vector. The four elements required by the Metropolis procedure are as follows: The value of f is the objective function. The system state is the point $\mathbf{x}$. The control parameter T is, as before, something like a temperature, with an annealing schedule by which it is gradually reduced. And there must be a generator of random changes in the configuration, that is, a procedure for taking a random step from $\mathbf{x}$ to $\mathbf{x}+\Delta\mathbf{x}$.

The last of these elements is the most problematical. The literature to date[10–13] describes several different schemes for choosing $\Delta\mathbf{x}$, none of which, in our view, inspire complete confidence. The problem is one of efficiency: A generator of random changes is inefficient if, *when local downhill moves exist*, it nevertheless almost always proposes an uphill move. A good generator, we think, should not become inefficient in narrow valleys; nor should it become more and more inefficient as convergence to a minimum is approached. Except possibly for Ref. 10, all of the schemes that we have seen are inefficient in one or both of these situations.

A right way, we think, for doing simulated annealing minimization on continuous control spaces is to use a modification of the downhill simplex method of Nelder and Mead. We now review this method in its ordinary (nonannealed) form.

A *simplex* is the geometrical figure consisting, in N dimensions, of $N+1$ points (or vertices) and all their interconnecting line segments, polygonal faces, etc. In two dimensions, a simplex is a triangle. In three dimensions it is a tetrahedron, not necessarily the regular tetrahedron. We are interested in simplexes that are nondegenerate, i.e., which enclose a finite inner N-dimensional volume. If any point of a nondegenerate simplex is taken as the origin, then the N other points define vector directions that span the N-dimensional vector space.

The downhill simplex method must be started not just with a single point, but with $N+1$ points, defining an initial simplex. If you think of one of these points (it matters not which) as being your initial starting point $\mathbf{P}_0$, then you can take the other N points to be

$$\mathbf{P}_i = \mathbf{P}_0 + \lambda \mathbf{e}_i, \tag{2}$$

where the $\mathbf{e}_i$'s are N unit vectors, and where λ is a constant which is your guess of the problem's characteristic length scale. (Or, you could have different λ_i's for each vector direction.)

The downhill simplex method now takes a series of steps, most steps just moving the point of the simplex where the function is largest ("highest point") through the opposite face of the simplex to a lower point. These steps are called reflections, and they are constructed to conserve the volume of the simplex (and hence maintain its nondegeneracy). When it can do so, the method expands the simplex in one or another direction to take larger steps. When it reaches a "valley floor," the method contracts itself in the transverse direction and tries to ooze down the valley. If there is a situation where the simplex is trying to "pass through the eye of a needle," it contracts itself in all directions, pulling itself in around its lowest (best) point. In the *Numerical Recipes* books,[1–3] the routine name amoeba is intended to be descriptive of this kind of behavior; the basic moves are summarized here in Fig. 1.

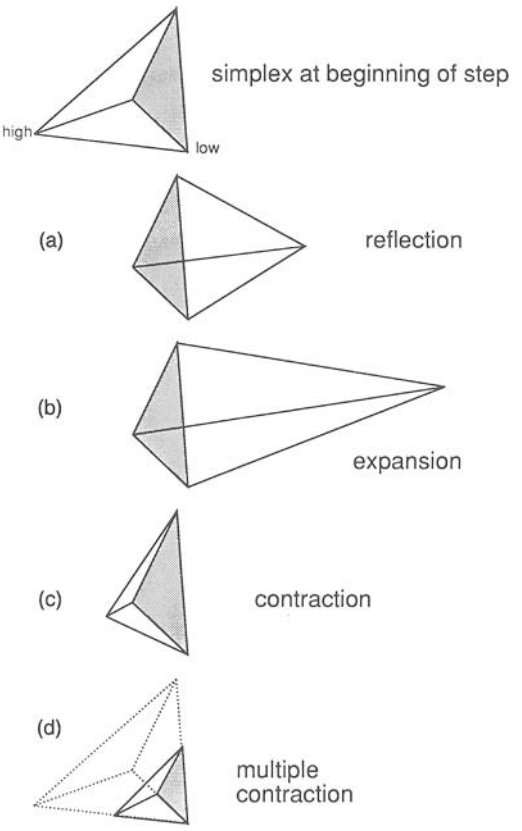

FIG. 1. Possible outcomes for a step in the downhill simplex method. The simplex at the beginning of the step, here a tetrahedron, is shown at the top. The simplex at the end of the step can be either (a) a reflection away from the high point, (b) a reflection and expansion away from the high point, (c) a contraction along one dimension from the high point, or (d) a contraction along all dimensions toward the low point. An appropriate sequence of such steps will always converge to a minimum of the function.

To graft the simplex method into the paradigm of simulated annealing, we replace the single point **x** as a description of the system state by a simplex of $N + 1$ points. The "moves" are the same as those just described, namely reflections, expansions, and contractions of the simplex. The implementation of the Metropolis procedure is slightly subtle: we *add* a positive, logarithmically distributed random variable, proportional to the temperature T, to the stored function value associated with every vertex of the simplex, and we *subtract* a similar random variable from the function value of every new point that is tried as a replacement point. Like the ordinary Metropolis procedure, this method always accepts a true downhill step, but sometimes accepts an uphill one. In the limit $T \rightarrow 0$, this algorithm reduces exactly to the downhill simplex method and converges to a local minimum.

At a finite value of T, the simplex expands to a scale that approximates the size of the region that can be reached at this temperature, and then executes a stochastic, tumbling Brownian motion within that region,

Box 1

```
      SUBROUTINE amebsa(p,y,mp,np,ndim,pb,yb,ftol,funk,iter,temptr)
      INTEGER iter,mp,ndim,np,NMAX
      REAL ftol,temptr,yb,p(mp,np),pb(np),y(mp),funk,ran1,amotsa
      PARAMETER (NMAX=200)
C     USES amotsa,funk,ran1
        Multidimensional minimization of the function funk(x) where x(1:ndim) is a vector in
        ndim dimensions, by simulated annealing combined with the downhill simplex method of
        Nelder and Mead. The input matrix p(1..ndim+1,1..ndim) has ndim+1 rows, each an
        ndim-dimensional vector which is a vertex of the starting simplex. Also input is the vector
        y(1:ndim+1), whose components must be pre-initialized to the values of funk evaluated at
        the ndim+1 vertices (rows) of p; ftol, the fractional convergence tolerance to be achieved
        in the function value for an early return; iter, and temptr. The routine makes iter func-
        tion evaluations at an annealing temperature temptr, then returns. You should then decrease
        temptr according to your annealing schedule, reset iter and call the routine again (leaving
        other arguments unaltered between calls). If iter is returned with a positive value, then early
        convergence and return occurred. If you initialize yb to a very large value on the first call,
        then yb and pb(1:ndim) will subsequently return the best function value and point ever
        encountered (even if it is no longer a point in the simplex).
      INTEGER i,idum,ihi,ilo,inhi,j,m,n
      REAL rtol,sum,swap,tt,yhi,ylo,ynhi,ysave,yt,ytry,psum(NMAX)
      COMMON /ambsa/ idum,tt
      external funk
      tt=-temptr
1     do 12 n=1,ndim                    Enter here when starting or have just overall contracted.
        sum=0.                          Recompute psum.
        do 11 m=1,ndim+1
          sum=sum+p(m,n)
        enddo 11
        psum(n)=sum
      enddo 12
2     ilo=1                             Enter here when have just changed a single point. Determine
      inhi=1                            which point is the highest (worst), next-highest, and lowest
      ihi=2                             (best).
      ylo=y(1)+tt*log(ran1(idum))       Whenever we "look at" a vertex, it gets a random thermal
      ynhi=ylo                          fluctuation.
      yhi=y(2)+tt*log(ran1(idum))
      if (ylo.gt.yhi) then
        ihi=1
        inhi=2
        ilo=2
        ynhi=yhi
        yhi=ylo
        ylo=ynhi
      endif
      do 13 i=3,ndim+1                  Loop over the points in the simplex.
        yt=y(i)+tt*log(ran1(idum))      More thermal fluctuations.
        if(yt.le.ylo) then
          ilo=i
          ylo=yt
        endif
        if(yt.gt.yhi) then
          inhi=ihi
          ynhi=yhi
          ihi=i
          yhi=yt
        else if(yt.gt.ynhi) then
          inhi=i
          ynhi=yt
        endif
      enddo 13
      rtol=2.*abs(yhi-ylo)/(abs(yhi)+abs(ylo))
        Compute the fractional range from highest to lowest and return if satisfactory.
      if (rtol.lt.ftol.or.iter.lt.0) then   If returning, put best point and value in slot 1.
        swap=y(1)
        y(1)=y(ilo)
        y(ilo)=swap
        do 14 n=1,ndim
          swap=p(1,n)
          p(1,n)=p(ilo,n)
          p(ilo,n)=swap
        enddo 14
        return
      endif
      iter=iter-2
      Begin a new iteration. First extrapolate by a factor -1 through the face of the simplex across from the
      high point, i.e., reflect the simplex from the high point.
      ytry=amotsa(p,y,psum,mp,np,ndim,pb,yb,funk,ihi,yhi,-1.0)
      if (ytry.le.ylo) then
      Gives a result better than the best point, so try an additional extrapolation by a factor 2.
        ytry=amotsa(p,y,psum,mp,np,ndim,pb,yb,funk,ihi,yhi,2.0)
      else if (ytry.ge.ynhi) then
      The reflected point is worse than the second-highest, so look for an intermediate lower point, i.e., do a
      one-dimensional contraction.
        ysave=yhi
        ytry=amotsa(p,y,psum,mp,np,ndim,pb,yb,funk,ihi,yhi,0.5)
        if (ytry.ge.ysave) then         Can't seem to get rid of that high point. Better contract
          do 16 i=1,ndim+1              around the lowest (best) point.
            if(i.ne.ilo)then
              do 15 j=1,ndim
                psum(j)=0.5*(p(i,j)+p(ilo,j))
                p(i,j)=psum(j)
              enddo 15
              y(i)=funk(psum)
            endif
          enddo 16
          iter=iter-ndim
          goto 1
        endif
      else
        iter=iter+1                     Correct the evaluation count.
      endif
      goto 2
      END

      FUNCTION amotsa(p,y,psum,mp,np,ndim,pb,yb,funk,ihi,yhi,fac)
      INTEGER ihi,mp,ndim,np,NMAX
      REAL amotsa,fac,yb,yhi,p(mp,np),pb(np),psum(np),y(mp),funk,ran1
      PARAMETER (NMAX=200)
C     USES funk,ran1
        Extrapolates by a factor fac through the face of the simplex across from the high point, tries
        it, and replaces the high point if the new point is better.
      INTEGER idum,j
      REAL fac1,fac2,tt,yflu,ytry,ptry(NMAX)
      COMMON /ambsa/ idum,tt
      fac1=(1.-fac)/ndim
      fac2=fac1-fac
      do 17 j=1,ndim
        ptry(j)=psum(j)*fac1-p(ihi,j)*fac2
      enddo 17
      ytry=funk(ptry)
      if (ytry.le.yb) then              Save the best-ever.
        do 18 j=1,ndim
          pb(j)=ptry(j)
        enddo 18
        yb=ytry
      endif
      yflu=ytry-tt*log(ran1(idum))      We added a thermal fluctuation to all the current vertices,
      if (yflu.lt.yhi) then             but we subtract it here, so as to give the simplex a thermal
        y(ihi)=ytry                     Brownian motion: it likes to accept any suggested change.
        yhi=yflu
        do 19 j=1,ndim
          psum(j)=psum(j)-p(ihi,j)+ptry(j)
          p(ihi,j)=ptry(j)
        enddo 19
      endif
      amotsa=yflu
      return
      END
```

sampling new, approximately random, points as it does so. The efficiency with which a region is explored is independent of its narrowness (for an ellipsoidal valley, the ratio of its principal axes) and orientation. If the temperature is reduced sufficiently slowly, it becomes highly likely that simplex will shrink into that region containing the lowest relative minimum encountered.

As in all applications of simulated annealing, there can be quite a lot of problem-dependent subtlety in the phrase "sufficiently slowly"; success or failure is quite often determined by the choice of annealing schedule. Here are some possibilities worth trying:

- Reduce T to $(1-\epsilon)T$ after every m moves, where ϵ/m is determined by experiment.
- Budget a total of K moves, and reduce T after every m moves to a value $T = T_0(1-k/K)^\alpha$, where k is the cumulative number of moves thus far, and α is a constant, say 1, 2, or 4. The optimal value for α depends on the statistical distribution of relative minima of various depths. Larger values of α spend more iterations at lower temperature.
- After every m moves, set T to β times $f_1 - f_b$, where β is an experimentally determined constant of order 1, f_1 is the smallest function value currently represented in the simplex, and f_b is the best function ever encountered. However, never reduce T by more than some fraction γ at a time.

Another strategic question is whether to do an occasional *restart*, where a vertex of the simplex is discarded in favor of the "best-ever" point. (You must be sure that the best-ever point is not currently in the simplex when you do this!) We have found problems for which restarts—every time the temperature has decreased by a factor of 3, say—are highly beneficial; we have found other problems for which restarts have no positive, or a somewhat negative, effect.

Box 1 lists a subroutine implementing these ideas. The name amebsa is a blend of "amoeba" with "-sa", for simulated annealing. We have had considerable success with amebsa on otherwise intractable problems. The subroutine amotsa is captive to amebsa, used by it to construct the interpolations and extrapolations implied by the moves shown in Fig. 1. A random number generator ran1(idum) is assumed.

We thank our coauthors for permission to incorporate verbatim material from Refs. 1–3 (descriptions of simulated annealing and Nelder–Mead method) in this column. ■

In our next column: Multigrid Methods for Boundary Value Problems.

References

1. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing* (Cambridge U. P., New York, 1986).
2. W. H.Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing* (Cambridge U. P., New York, 1988).
3. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in Pascal: The Art of Scientific Computing* (Cambridge, U. P., New York, 1989).
4. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Science **220**, 671 (1983).
5. S. Kirkpatrick, J. Stat. Phys. **34**, 975 (1984).
6. M. P. Vecchi and S. Kirkpatrick, IEEE Trans. Comput. Aided Design **CAD-2**, 215 (1983).
7. R. H. J. M. Otten and L. P. P. P. van Ginneken, *The Annealing Algorithm* (Kluwer, Boston, 1989).
8. J. A. Nelder and R. Mead, Comput. J. **7**, 308 (1965).
9. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, J. Chem. Phys. **21**, 1087 (1953).
10. D. Vanderbilt and S. G. Louie, J. Comput. Phys. **56**, 259 (1984).
11. I. O. Bohachevsky, M. E. Johnson, and M. L. Stein, Technometrics **28**, 209 (1986).
12. A. Corana, M. Marchesi, C. Martini, and S. Ridella, ACM Trans. Math. Software **13**, 262 (1987).
13. C. J. P. Bélisle, H. E. Romeijn, and R. L. Smith, Tech. Rep. 90-25, Department of Industrial and Operations Engineering, University of Michigan (1990).