optimization generally. The next section looks at a particular kind of discrete optimization problem.

REFERENCES AND FURTHER READING:

Bland, R.G. 1981, *Scientific American*, vol. 244, (June) p. 126.

Kolata, G. 1982, *Science*, vol. 217, p. 39.

Cooper, L., and Steinberg, D. 1970, *Introduction to Methods of Optimization* (Philadelphia: Saunders).

Dantzig, G.B. 1963, *Linear Programming and Extensions* (Princeton, N.J.: Princeton University Press).

Gass, S.T. 1969, *Linear Programming*, 3rd ed. (New York: McGraw-Hill).

Murty, K.G. 1976, *Linear and Combinatorial Programming* (New York: Wiley).

Land, A.H., and Powell, S. 1973, *Fortran Codes for Mathematical Programming* (London: Wiley-Interscience).

Kuenzi, H.P., Tzschach, H.G., and Zehnder, C.A. 1971 *Numerical Methods of Mathematical Optimization* (New York: Academic Press).

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §4.10.

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag).

10.9 Combinatorial Minimization: Method of Simulated Annealing

The *method of simulated annealing* is a technique that has recently attracted significant attention as suitable for optimization problems of very large scale. For practical purposes, it 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: The arrangement of several hundred thousand circuit elements on a tiny silicon substrate is optimized so as to minimize interference among their connecting wires. Amazingly, the implementation of the algorithm is quite simple.

Notice that the two applications cited are both examples of *combinatorial minimization*. There is an objective function to be minimized, as usual; but the space over which that function is defined is not simply the N-dimensional space of N continuously variable parameters. Rather, it is a discrete, but very large, configuration space, like the set of possible orders of cities, or the set of possible allocations of silicon "real estate" to circuit elements. The number of elements in the configuration space is factorially large, so that they cannot be explored exhaustively. Furthermore, since the set is discrete, we are deprived of any notion of "continuing downhill in a favorable direction." The concept of "direction" may not have any meaning in the configuration space.

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 up to 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.

So 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 all of the minimization algorithms thus far in this chapter correspond to rapid cooling or quenching. In all cases, we have gone greedily for the quick, nearby solution: from the starting point, go immediately downhill as far as you can go. This, as often 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,

$$\text{Prob}\,(E) \sim \exp(-E/kT) \tag{10.9.1}$$

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. The quantity k (Boltzmann's constant) is a constant of nature which relates temperature to energy. 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 coworkers 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 $p = \exp[-(E_2 - E_1)/kT]$. 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* took 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.

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 meaning of "high" and "low" in this context, and the assignment of a schedule, may require physical insight and/or trial-and-error experiments.

The Traveling Salesman Problem

A concrete illustration is provided by the traveling salesman problem. The salesperson visits N cities with given positions (x_i, y_i), returning finally to his or her city of origin. Each city is to be visited only once, and the route is to be made as short as possible. This problem belongs to a class known as *NP-complete* problems, whose computation time for an *exact* solution increases with N as exp(const. $\times N$), becoming rapidly prohibitive in cost as N increases. The traveling salesman problem also belongs to a class of minimization problems for which the objective function E has many local minima. In practical cases, it is often enough to be able to choose from these a minimum which, even if not absolute, cannot be significantly improved upon. The annealing method manages to achieve this, while limiting its calculations to scale as a small power of N.

As a problem in simulated annealing, the traveling salesman problem is handled as follows:

1. *Configuration.* The cities are numbered $i = 1 \ldots N$ and each has coordinates (x_i, y_i). A configuration is a permutation of the number $1 \ldots N$, interpreted as the order in which the cities are visited.

2. *Rearrangements.* An efficient set of moves has been suggested by Lin. The moves consist of two types: (a) A section of path is removed and then replaced with the same cities running in the opposite order; or (b) a section of path is removed and then replaced in between two cities on another, randomly chosen, part of the path.

3. *Objective Function.* In the simplest form of the problem, E is taken just as the total length of journey,

$$E = L \equiv \sum_{i=1}^{N} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \tag{10.9.2}$$

with the convention that point $N + 1$ is identified with point 1. To illustrate the flexibility of the method, however, we can add the following additional wrinkle: suppose that the salesman has an irrational fear of flying over the Mississippi River. In that case, we would assign each city a parameter μ_i,

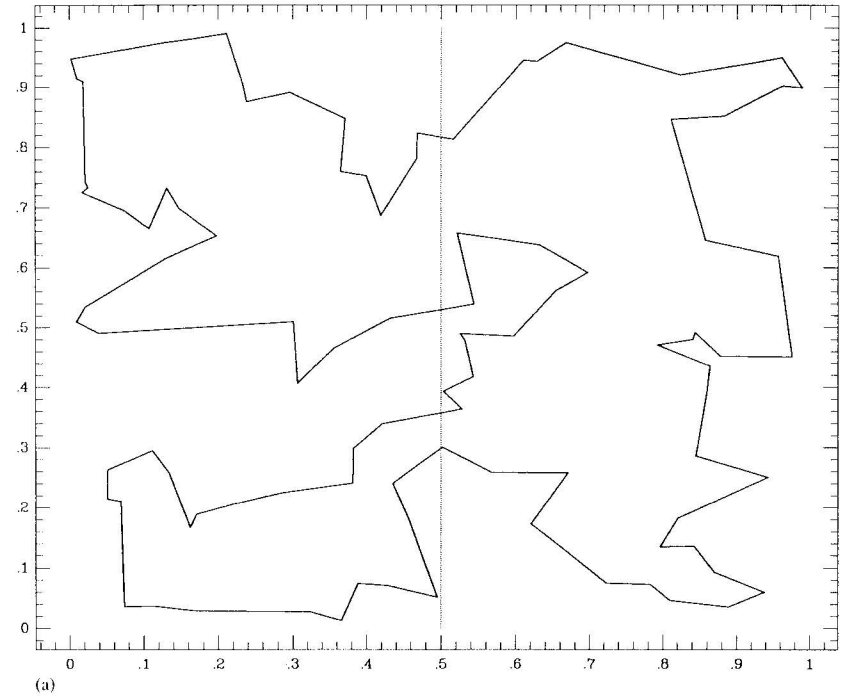

Figure 10.9.1. Traveling salesman problem solved by simulated annealing. The (nearly) shortest path among 100 randomly positioned cities is shown in (a). The dotted line is a river, but there is no penalty in crossing. In (b) the river-crossing penalty is made large, and the solution restricts itself to the minimum number of crossings, two. In (c) the penalty has been made negative: the salesman is actually a smuggler who crosses the river on the flimsiest excuse!

equal to $+1$ if it is east of the Mississippi, -1 if it is west, and take the objective function to be

$$E = \sum_{i=1}^{N} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} + \lambda(\mu_i - \mu_{i+1})^2 \tag{10.9.3}$$

A penalty 4λ is thereby assigned to any river crossing. The algorithm now finds the shortest path that avoids crossings. The relative importance that it assigns to length of path versus river crossings is determined by our choice of λ. Figure 10.9.1 shows the results obtained. Clearly, this technique can be generalized to include many conflicting goals in the minimization.

4. *Annealing schedule.* This requires experimentation. We first generate some random rearrangements, and use them to determine the range of values of ΔE that will be encountered from move to move. Choosing a starting value for the parameter T which is considerably larger than the largest ΔE normally encountered, we proceed downward in multiplicative steps each amounting to a 10 percent decrease in T. We hold each new value of T constant for, say, $100N$

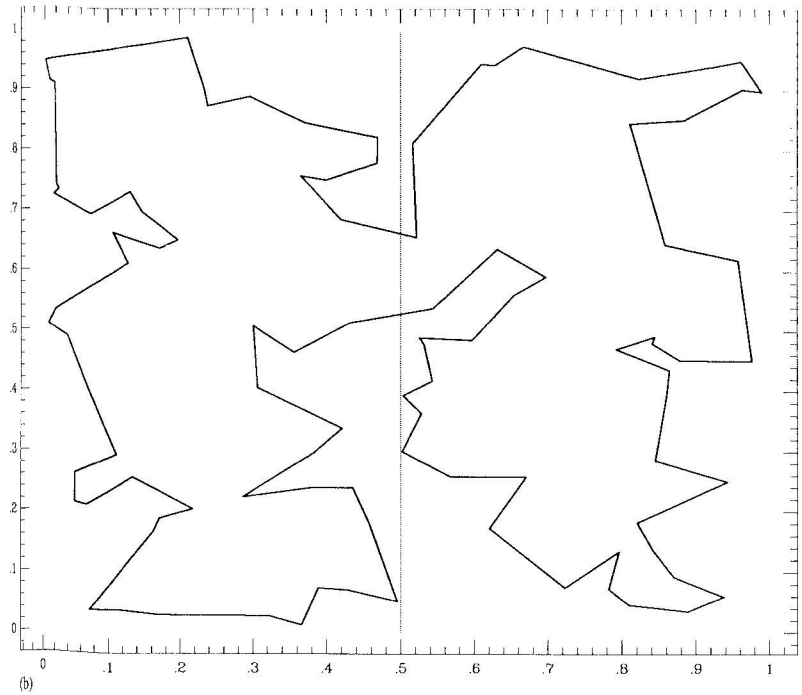

Figure 10.9.1. Traveling salesman problem solved by simulated annealing (see caption on previous page).

reconfigurations, or for $10N$ successful reconfigurations, whichever comes first. When efforts to reduce E further become sufficiently discouraging, we stop.

The following traveling salesman program, using the Metropolis algorithm, should illustrate for you the important aspects of the simulated annealing technique.

```
      SUBROUTINE ANNEAL(X,Y,IORDER,NCITY)
          This algorithm finds the shortest round-trip path to NCITY cities whose coordinates are
          in the arrays X(I),Y(I). The array IORDER(I) specifies the order in which the cities are
          visited. On input, the elements of IORDER may be set to any permutation of the numbers
          1 to NCITY. This routine will return the best alternative path it can find.
      DIMENSION X(NCITY),Y(NCITY),IORDER(NCITY),N(6)
      LOGICAL ANS
      ALEN(X1,X2,Y1,Y2)=SQRT((X2-X1)**2+(Y2-Y1)**2)
      NOVER=100*NCITY                 Maximum number of paths tried at any temperature.
      NLIMIT=10*NCITY                 Maximum number of successful path changes before continuing.
      TFACTR=0.9                      Annealing schedule - T is reduced by this factor on each step.
      PATH=0.0
      T=0.5
      DO 11 I=1,NCITY-1               Calculate initial path length.
        I1=IORDER(I)
        I2=IORDER(I+1)
        PATH=PATH+ALEN(X(I1),X(I2),Y(I1),Y(I2))
   11 CONTINUE
      I1=IORDER(NCITY)                Close the loop by tying path ends together.
      I2=IORDER(1)
      PATH=PATH+ALEN(X(I1),X(I2),Y(I1),Y(I2))
      IDUM=-1
      ISEED=111
      DO 13 J=1,100                   Try up to 100 temperature steps.
        NSUCC=0
        DO 12 K=1,NOVER
    1     N(1)=1+INT(NCITY*RAN3(IDUM))      Choose beginning of segment ..
          N(2)=1+INT((NCITY-1)*RAN3(IDUM))..and end of segment.
          IF (N(2).GE.N(1)) N(2)=N(2)+1
          NN=1+MOD((N(1)-N(2)+NCITY-1),NCITY) NN is the number of cities not on the segment.
          IF (NN.LT.3) GOTO 1
          IDEC=IRBIT1(ISEED)    Decide whether to do a segment reversal or transport.
          IF (IDEC.EQ.0) THEN   Do a transport.
            N(3)=N(2)+INT(ABS(NN-2)*RAN3(IDUM))+1
            N(3)=1+MOD(N(3)-1,NCITY)      Transport to a location not on the path.
            CALL TRNCST(X,Y,IORDER,NCITY,N,DE)  Calculate cost.
            CALL METROP(DE,T,ANS)     Consult the oracle.
            IF (ANS) THEN
              NSUCC=NSUCC+1
              PATH=PATH+DE
              CALL TRNSPT(IORDER,NCITY,N)  Carry out the transport.
            ENDIF
          ELSE                    Do a path reversal
            CALL REVCST(X,Y,IORDER,NCITY,N,DE)  Calculate cost.
            CALL METROP(DE,T,ANS)     Consult the oracle.
            IF (ANS) THEN
              NSUCC=NSUCC+1
              PATH=PATH+DE
              CALL REVERS(IORDER,NCITY,N)  Carry out the reversal.
            ENDIF
          ENDIF
          IF (NSUCC.GE.NLIMIT) GOTO 2  Finish early if we have enough successful changes.
   12   CONTINUE
    2   WRITE(*,*)
        WRITE(*,*) 'T =',T,' Path Length =',PATH
        WRITE(*,*) 'Successful Moves:  ',NSUCC
```