The following solution is given by SAS:

Assigning Task to Workstation	WORKS			
	1	2	3	4
	amount	amount	amount	amount
	Sum	Sum	Sum	Sum
TASKS				
1	1.00	0.00	0.00	0.00
2	0.00	1.00	0.00	0.00
3	0.00	1.00	0.00	0.00
4	0.00	1.00	0.00	0.00
5	0.00	0.00	1.00	0.00
6	0.00	0.00	1.00	0.00
7	0.00	0.00	0.00	1.00
8	0.00	0.00	0.00	1.00

6 *Traveling Salesman Problem*

In this chapter, we present the traveling salesman problem and demonstrate how SAS/OR® can be applied to solve the problem to optimality.

6.1 Traveling Salesman Problem (TSP)

6.1.1 Concept of TSP

The *traveling salesman problem* (TSP) is one of the most widely studied integer programming problems and can be described as follows. Before visiting n distinct cities and then returning home, a salesman wants to determine the sequence of the travel so that the overall travel distance is minimized while visiting each city no more than once. Although the TSP is conceptually simple, it is difficult to obtain an optimal solution. In an n-city situation, any permutation of n cities yields a possible solution. As a consequence, $n!$ possible tours must be evaluated in the search space. By introducing decision variables x_{ij} to represent the tour of the salesman from city i to city j, the TSP can be formulated as shown in Model 6.1.1.

Model 6.1.1 Standard traveling salesman model

$$\text{Minimize } z = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij} \qquad (6.1.1)$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, 2, \ldots, n; i \neq j \qquad (6.1.2)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, 2, \ldots, n; i \neq j \qquad (6.1.3)$$

$$u_i - u_j + n x_{ij} \leq n - 1 \quad i, j = 1, 2, \ldots, n; i \neq j \qquad (6.1.4)$$

All $x_{ij} = 0$ or 1. All $u_i \geq 0$ and is a set of integers.

The distance between city i and city j is denoted as c_{ij}. Objective function 6.1.1 minimizes the total distance traveled in a tour. Constraint set 6.1.2 ensures that the salesman arrives at each city once. Constraint set 6.1.3 ensures that the salesman leaves each city once. Constraint set 6.1.4 is used to avoid a subtour. Because the decision variable x_{ij} is used, the solutions generated may form subtours, provided that constraint set 6.1.4 is not incorporated. Actually, the solutions consisting of subtours are infeasible because the travel sequence of the salesman is still unknown after all the decision variables x_{ij} are found. For example, the decision variables for a six-city problem are: $x_{12} = x_{23} = x_{31} = x_{46} = x_{65} = x_{54} = 1$. It is assumed that city 1 is the home city, which means that the salesman should start and end at this city. In this case, two subtours are formed:

- First subtour: city 1 → city 2 → city 3 → city 1
- Second subtour: city 4 → city 6 → city 5 → city 4

According to the first subtour, the salesman moves back to the home city, or city 1, after serving the customers in city 3. Which city is visited next after visiting all cities in the first subtour is not indicated. So this solution is unacceptable, and constraint set 6.1.4 in Model 6.1.1 must be included. Although constraint set 6.1.4 guarantees that the solution generated is feasible, it increases the complexity of the model because there are $n(n - 1)$ constraints in this subtour elimination constraint.

6.1.2 Example of TSP

Table 6.1 shows a TSP that has six cities. It is assumed that city 1 is the home city. The upper-right corner of each cell in the tableau represents the travel distance, c_{ij}. Because it is not permitted to visit a city more than once, x_{ii} is discarded.

By introducing decision variables x_{ij} to represent the tour of the salesman from city i to city j, the TSP can be formulated as shown in Model 6.1.2.

Model 6.1.2 Example of formulation of TSP

$$\begin{aligned}\text{Minimize } 4\,x_{12} + 3\,x_{13} + 4\,x_{14} + 7\,x_{15} + 8\,x_{16} \\ + 4\,x_{21} + 4\,x_{23} + 3\,x_{24} + 7\,x_{25} + 6\,x_{26} \\ + 3\,x_{31} + 4\,x_{32} + 2\,x_{34} + 4\,x_{35} + 6\,x_{36} \\ + 4\,x_{41} + 3\,x_{41} + 2\,x_{43} + 4\,x_{45} + 4\,x_{46} \\ + 7\,x_{51} + 7\,x_{52} + 4\,x_{53} + 4\,x_{54} + 4\,x_{56} \\ + 8\,x_{61} + 6\,x_{62} + 6\,x_{63} + 4\,x_{64} + 4\,x_{65}\end{aligned} \tag{6.1.5}$$

subject to

$$x_{21} + x_{31} + x_{41} + x_{51} + x_{61} = 1 \tag{6.1.6}$$

TABLE 6.1
A Traveling Salesman Tableau

	City j					
City i	1	2	3	4	5	6
1	– –	4 x_{12}	3 x_{13}	4 x_{14}	7 x_{15}	8 x_{16}
2	4 x_{21}	– –	4 x_{23}	3 x_{24}	7 x_{25}	6 x_{26}
3	3 x_{31}	4 x_{32}	– –	2 x_{34}	4 x_{35}	6 x_{36}
4	4 x_{41}	3 x_{42}	2 x_{43}	– –	4 x_{45}	4 x_{46}
5	7 x_{51}	7 x_{52}	4 x_{53}	4 x_{54}	– –	4 x_{56}
6	8 x_{61}	6 x_{62}	6 x_{63}	4 x_{64}	4 x_{65}	– –

$$x_{12} + x_{32} + x_{42} + x_{52} + x_{62} = 1 \tag{6.1.7}$$

$$x_{13} + x_{23} + x_{43} + x_{53} + x_{63} = 1 \tag{6.1.8}$$

$$x_{14} + x_{24} + x_{34} + x_{54} + x_{64} = 1 \tag{6.1.9}$$

$$x_{15} + x_{25} + x_{35} + x_{45} + x_{65} = 1 \tag{6.1.10}$$

$$x_{16} + x_{26} + x_{36} + x_{46} + x_{56} = 1 \tag{6.1.11}$$

$$x_{12} + x_{13} + x_{14} + x_{15} + x_{16} = 1 \tag{6.1.12}$$

$$x_{21} + x_{23} + x_{24} + x_{25} + x_{26} = 1 \tag{6.1.13}$$

$$x_{31} + x_{32} + x_{34} + x_{35} + x_{36} = 1 \tag{6.1.14}$$

$$x_{41} + x_{42} + x_{43} + x_{45} + x_{46} = 1 \tag{6.1.15}$$

$$x_{51} + x_{52} + x_{53} + x_{54} + x_{56} = 1 \tag{6.1.16}$$

$$x_{61} + x_{62} + x_{63} + x_{64} + x_{65} = 1 \tag{6.1.17}$$

$$u_1 - u_2 + 6\,x_{12} \le 5 \tag{6.1.18}$$

$$u_1 - u_3 + 6\,x_{13} \le 5 \tag{6.1.19}$$

$$u_1 - u_4 + 6\,x_{14} \le 5 \tag{6.1.20}$$

$$u_1 - u_5 + 6\,x_{15} \le 5 \tag{6.1.21}$$

$$u_1 - u_6 + 6\,x_{16} \le 5 \tag{6.1.22}$$

$$u_2 - u_1 + 6\,x_{21} \le 5 \tag{6.1.23}$$

$$u_2 - u_3 + 6\,x_{23} \le 5 \tag{6.1.24}$$

$$u_2 - u_4 + 6\,x_{24} \le 5 \qquad (6.1.25)$$
$$u_2 - u_5 + 6\,x_{25} \le 5 \qquad (6.1.26)$$
$$u_2 - u_6 + 6\,x_{26} \le 5 \qquad (6.1.27)$$
$$u_3 - u_1 + 6\,x_{31} \le 5 \qquad (6.1.28)$$
$$u_3 - u_2 + 6\,x_{32} \le 5 \qquad (6.1.29)$$
$$u_3 - u_4 + 6\,x_{34} \le 5 \qquad (6.1.30)$$
$$u_3 - u_5 + 6\,x_{35} \le 5 \qquad (6.1.31)$$
$$u_3 - u_6 + 6\,x_{36} \le 5 \qquad (6.1.32)$$
$$u_4 - u_1 + 6\,x_{41} \le 5 \qquad (6.1.33)$$
$$u_4 - u_2 + 6\,x_{42} \le 5 \qquad (6.1.34)$$
$$u_4 - u_3 + 6\,x_{43} \le 5 \qquad (6.1.35)$$
$$u_4 - u_5 + 6\,x_{45} \le 5 \qquad (6.1.36)$$
$$u_4 - u_6 + 6\,x_{46} \le 5 \qquad (6.1.37)$$
$$u_5 - u_1 + 6\,x_{51} \le 5 \qquad (6.1.38)$$
$$u_5 - u_2 + 6\,x_{52} \le 5 \qquad (6.1.39)$$
$$u_5 - u_3 + 6\,x_{53} \le 5 \qquad (6.1.40)$$
$$u_5 - u_4 + 6\,x_{54} \le 5 \qquad (6.1.41)$$
$$u_5 - u_6 + 6\,x_{56} \le 5 \qquad (6.1.42)$$
$$u_6 - u_1 + 6\,x_{61} \le 5 \qquad (6.1.43)$$
$$u_6 - u_2 + 6\,x_{62} \le 5 \qquad (6.1.44)$$
$$u_6 - u_3 + 6\,x_{63} \le 5 \qquad (6.1.45)$$
$$u_6 - u_4 + 6\,x_{64} \le 5 \qquad (6.1.46)$$
$$u_6 - u_5 + 6\,x_{65} \le 5 \qquad (6.1.47)$$

All $x_{ij} = 0$ or 1. All $u_i \ge 0$ and is a set of integers.

Objective function 6.1.5 calculates the total travel distance of the salesman for serving all customers in the cities. If the moving speed is incorporated, then the objective can be the minimization of the total travel time. Constraint sets 6.1.6 to 6.1.11 guarantee that exactly one city must be visited immediately before city j. Constraint sets 6.1.12 to 6.1.17 ensure that exactly one city must be visited immediately after city i. Constraint sets 6.1.18 to 6.1.47 are known as the *subtour elimination constraints*. Because the model contains binary variables (i.e., x_{ij}) and integer variables (i.e., u_i), Model 6.1.2 can be regarded as a *pure integer linear programming model.*

TABLE 6.2
Definitions of the TSP Variants

TSP Variants	Definitions
Asymmetric TSP	The bidirectional distances between a pair of customers are not necessarily identical.
Generalized TSP	The customers are divided into several groups, and not all customers need to be visited.
Multiple TSP	More than one salesman is allowed to be used, and each customer is served by a salesman only.
Period TSP	The vehicle has to visit each customer several times over a given period of time.
Pickup-and-delivery TSP	The vehicle may both receive products from the customers and deliver products to the customers at the same time.
Probabilistic TSP	Each customer has a predetermined probability of requiring a visit.
TSP with precedence constraints	There exist a delivering order between the customers.
TSP with time window	The vehicles must arrive at the customers before the latest arrival time, but arriving before the earliest arrival time results in waiting.

In many real-life situations, a TSP is not solved optimally by exact algorithms, especially when the problem size is huge. Instead, it is solved by heuristics, such as the nearest neighbor heuristic. The principle of the nearest neighbor heuristic is to start with the first city randomly, then to select the next city as close as possible to the previous one from those unselected cities to form the travel sequence until all cities are selected.

The most common TSP is known as the Euclidean TSP, in which distance matrix c is expected to be symmetrical, (i.e., $c_{ij} = c_{ji}$ for all i, j),and to satisfy the triangle inequality (i.e., $c_{ik} \le c_{ij} + c_{jk}$ for all distinct i, j, k). There are extensive variations of the TSP, including the asymmetric TSP, generalized TSP, multiple TSP, period TSP, pickup-and-delivery TSP, probabilistic TSP, TSP with precedence constraints, and TSP with time window. The definitions of all TSP variants are described in Table 6.2.

6.1.3 ORTSP: SAS Code for TSP

ORTSP is a macro that solves TSPs in which a salesman wants to visit n distinct cities and then return home (see program "sasor_6_1.sas"). The aim is to determine the sequence of the travel so that the overall travel distance is minimized while visiting each city not more than once. The primary procedure used for a TSP is PROC GA.

Figure 6.1 illustrates the data flow in the ORTSP. It shows:

- The distance matrix that is required for ORTSP, in which the distance between each pair of traveling points is specified

CHAPTER 15

SOLVING INTEGER PROGRAMS: MODERN HEURISTIC TECHNIQUES

In Chapter 14, we explored various approaches to solving integer programs exactly. Much of this dealt with ways to either decompose the problem into smaller feasible regions (branch and bound) or to reformulate our problem by adding additional constraints (cutting planes). Each approach attempts to reduce the computational time required to solve integer programs, which can take very long to solve; recall the computational effort needed to solve large traveling salesperson problems discussed in Chapter 3.

Another approach to solving integer programs is to use heuristic methods, which we first discussed in Chapter 5. These methods attempt to find a good solution in an efficient manner, but are not guaranteed to always find the optimal solution. However, some of the methods discussed in Chapter 5 have potential drawbacks that limit their effectiveness.

In this chapter, we explore a few modern heuristic approaches to discrete optimization problems. We highlight their strengths and discuss weaknesses for each method. Since there is no one "best" approach to a problem, it is useful to have as many techniques as possible available to solve a given problem.

15.1 REVIEW OF LOCAL SEARCH METHODS: PROS AND CONS

Before we discuss new heuristic methods, it is useful to review the notion of local search from Chapter 5, which forms the backbone for many of the methods to be

discussed. Local search algorithms begin with a complete feasible solution **x** and search its neighborhood $N(\mathbf{x})$, that is, those solutions that are "close" to **x**, to see if there is a solution **y** with better objective value. If such a **y** exists, we select this solution and repeat the process. This method is outlined in Algorithm 15.1.

Algorithm 15.1 General Local Search Algorithm

```
x ← generateInitialSolution().
repeat
  x ← FindBetterSolution(N(x))
until no better solution in N(x) available
```

There are some issues with local search methods that need to be addressed. First, its performance (in terms of finding a near-optimal solution) greatly depends not only on the neighborhood $N(\mathbf{x})$ used for each feasible solution **x** but also on how the next solution is chosen from the neighborhood, as well as the initial solution itself. In Chapter 5, we already saw that different neighborhood structures can be derived for the same feasible solution, so this choice is an important one. It needs to strike a balance between containing many solutions, which has a better chance of obtaining the optimal solution but requires a great deal of time to search, and containing too few solutions, which can be scanned very quickly but leaves little improvement options.

Choosing an improving solution from the neighborhood of the current solution is also vital to the performance of a local search method. Some algorithms use a greedy approach, where the best solution in the neighborhood is chosen. Others choose the first improving solution found, while some choose randomly from all improving solutions found. These selection methods are similar to those used to select entering variables in the simplex method, and as in the simplex method, various approaches have been suggested with none proving to be the best in all cases. Greedy methods not only have the advantage of quicker improvements but also can occasionally stop after fewer iterations. Random selection expands the number of possible paths to local optimal solutions our algorithm could explore, but to truly benefit from randomization multiple runs of the method must be done.

In addition, local search methods stop once a local optimal solution is found, that is, one that has no improving solution within its neighborhood. This means that it is possible for an algorithm to search only a small part of the feasible region and ignore the remainder. We could run our local search approach multiple times, each time starting at a different solution (chosen by some randomization approach); however, as before we should do this many times to truly see the effect of randomization. If there are very few local optimal solutions, so that many initial solutions "funnel" to the same ones, then such multistart approaches would notice this, and reporting the best local optima found would give confidence to it being (possibly) the best solution; however, if there are many local optima, then a very large number of initial solutions would be needed to feel as confident about the quality of our final solution.

Because of these drawbacks, different strategies to guide the search process have been proposed. Known as **metaheuristics**, these methods typically incorporate some random component and often use search memory to guide future selections. They often attempt to avoid being "trapped" at a local optimal solution by using methods to "back out" of such solutions. In addition, their goal is often to balance the notion of *diversification*, or the exploration of much of the feasible region, and *intensification*, or the focusing around a small area for the best solutions. In the next few sections, we will introduce some of the more common metaheuristics: simulated annealing, tabu search, genetic algorithms, and GRASP.

15.2 SIMULATED ANNEALING

Simulated Annealing is probably one of the oldest metaheuristic approaches; its basic approach dates back to the study of Metropolis et al. [64] in 1953 describing an algorithm used to simulate the heating and cooling of solid material in a heat bath (annealing). In annealing, a solid is heated past its melting point, where the atoms become free of their initial positions and change randomly through various states. As the solid is cooled the atoms stick together, often resulting in structures that have lower internal energy. Different cooling rates can produce different properties of the solid.

To turn this approach into a heuristic method for optimization problems, we emulate the cooling mechanism by a temperature T. Suppose, we are at a solution $\mathbf{x}$ to our problem whose objective function is f and we randomly select a potential solution $\mathbf{x}'$. If $f(\mathbf{x}')$ is better than $f(\mathbf{x})$, we accept $\mathbf{x}'$ as our current solution and continue. However, if it is not better, we will still accept $\mathbf{x}'$ with probability $e^{-|f(\mathbf{x})-f(\mathbf{x}')|/T}$. This probability follows the *Boltzmann distribution*. Note that, for a fixed value T, the larger the difference between the function values of $\mathbf{x}$ and $\mathbf{x}'$, the smaller the probability of acceptance of $\mathbf{x}'$. A general simulated annealing algorithm is given in Algorithm 15.2. In this formulation, some possible termination conditions are (1) when we reach a maximum number of iterations, (2) the temperature T gets close to 0 and (3) the current solution does not change after too many iterations; the most common condition is based upon the temperature.

Algorithm 15.2 General Simulated Annealing Algorithm (maximization problems)

```
x ← generateInitialSolution().
repeat
  x' ← RandomSolution($N(\mathbf{x})$)
  if $f(\mathbf{x}')$ is better than $f(\mathbf{x})$ then
    x ← x'
  else
    $p$ ← generateRandomNumber().
    if $p \le e^{-\frac{|f(\mathbf{x})-f(\mathbf{x}')|}{T}}$ then
      x ← x'
    end if
  end if
  $T$ ← UpdateTemp($T$)
until termination conditions satisfied
```

Typically, the temperature cools according to a function $Q(T, k)$ of both the current temperature T and the iteration number k. This allows the temperature to cool at different rates during the algorithm's run, enabling the algorithm to balance the demands of diversification and intensification. For example, early in the search process T might be large and change at either a constant or a linear rate so as to examine different areas of the feasible region. Later in the process, we want T to decrease rapidly in order to converge to one solution. Other approaches where the temperature both increases and decreases have been suggested as a way to oscillate between the two demands. In fact, under some general conditions on the cooling rate of T, the algorithm will converge to a global optimal solution with probability 1 (note that this does not necessarily mean the solution is the global optimal solution, but that such instances are extremely rare). However, such required cooling rates are not practical since they decrease the temperature too slowly.

When implementing a simulated annealing algorithm for a problem, there are three initial decisions we need to make: (1) initial temperature, (2) cooling method, and (3) stopping condition. Each of these affects the performance of the method, and so some thought must be put into each decision. For example, the initial temperature T_0 should be chosen high enough so that the acceptance rate for "worse" solutions is high in the initial iterations. The choice of an appropriate T_0 can be problem specific since knowing the magnitude of solutions in the neighborhood of the initial solutions should influence our choice. Many implementations typically try various initial values before settling on a specific value.

The cooling schedule has probably the most influence on the performance of the algorithm. Its role is to both allow initial random fluctuations in the solutions early in the process and then prevent good solutions to be replaced by the worse ones later. Some implementations allow multiple solutions to be generated at each temperature and larger temperature shifts, while others generate only one solution per temperature, but then reduce the temperature more slowly. In practice, two common cooling schemes are a geometric approach

$$T_{k+1} = \alpha T_k,$$

where α is close to 1, and the function

$$T_{k+1} = \frac{T_k}{1+\beta T_k},$$

where β is close to 0. Some implementations have included a temperature increase using

$$T_{k+1} = \frac{T_k}{1-\gamma T_k}$$

when a worsening move is rejected, although this is not standard.

■ EXAMPLE 15.1

Consider the following 15-city traveling salesperson problem whose distance matrix is

$$\begin{bmatrix}
- & 48 & 42 & 34 & 45 & 32 & 25 & 21 & 46 & 31 & 23 & 25 & 24 & 46 & 36 \\
48 & - & 27 & 24 & 45 & 46 & 21 & 37 & 39 & 43 & 30 & 38 & 44 & 45 & 48 \\
42 & 27 & - & 47 & 48 & 33 & 37 & 46 & 34 & 27 & 29 & 33 & 31 & 28 & 32 \\
34 & 24 & 47 & - & 46 & 35 & 20 & 25 & 32 & 33 & 26 & 47 & 26 & 22 & 33 \\
45 & 45 & 48 & 46 & - & 48 & 34 & 34 & 38 & 40 & 31 & 38 & 50 & 27 & 45 \\
32 & 46 & 33 & 35 & 48 & - & 22 & 36 & 36 & 22 & 24 & 24 & 29 & 45 & 29 \\
25 & 21 & 27 & 20 & 34 & 22 & - & 46 & 21 & 27 & 39 & 34 & 21 & 28 & 40 \\
21 & 37 & 46 & 25 & 34 & 36 & 46 & - & 28 & 26 & 23 & 31 & 32 & 38 & 29 \\
46 & 39 & 34 & 32 & 38 & 36 & 21 & 28 & - & 34 & 28 & 45 & 31 & 48 & 30 \\
31 & 43 & 27 & 33 & 40 & 22 & 27 & 26 & 34 & - & 39 & 44 & 42 & 27 & 28 \\
23 & 30 & 29 & 26 & 31 & 24 & 39 & 23 & 28 & 39 & - & 36 & 45 & 37 & 30 \\
25 & 38 & 33 & 47 & 38 & 24 & 34 & 31 & 45 & 44 & 36 & - & 39 & 42 & 32 \\
24 & 44 & 31 & 26 & 50 & 29 & 21 & 32 & 31 & 42 & 45 & 39 & - & 27 & 36 \\
46 & 45 & 28 & 22 & 27 & 45 & 28 & 38 & 48 & 27 & 37 & 42 & 27 & - & 25 \\
36 & 48 & 32 & 33 & 45 & 29 & 40 & 29 & 30 & 28 & 30 & 32 & 36 & 25 & -
\end{bmatrix}. \tag{15.1}$$

Suppose that our initial tour is $\mathbf{x} = 1 \to 2 \to 3 \to \cdots \to 15 \to 1$, which has distance 548. We will set our initial temperature at $T_0 = 500$ and use $\alpha = 0.99$.

In the first iteration, we randomly swap the cities in locations 6 and 12, resulting in the tour

$$\mathbf{x}' = 1 \to 2 \to 3 \to 4 \to 5 \to 12 \to 7 \to 8 \to 9 \to 10 \to 11$$
$$\to 6 \to 13 \to 14 \to 15,$$

which has distance 528. Since this is a better distance, we keep this solution. We now update our temperature to $T_1 = \alpha T_0 = (0.99)500 = 495$. In the next iteration, we propose to swap the 1st and 10th positions, resulting in the solution

$$\mathbf{x}' = 10 \to 2 \to 3 \to 4 \to 5 \to 12 \to 7 \to 8 \to 9 \to 1 \to 11$$
$$\to 6 \to 13 \to 14 \to 15,$$

which has distance 511; we keep this solution as well, and our new temperature is $T_2 = (0.99)495 = 490.05$. We next propose to swap the 10th and 15th positions, which leads to the solution

$$\mathbf{x}' = 10 \to 2 \to 3 \to 4 \to 5 \to 12 \to 7 \to 8 \to 9 \to 15 \to 11$$
$$\to 6 \to 13 \to 14 \to 1,$$

which has distance 526. Since this has a higher distance than our current solution, we generate a random number and compare it to

$$e^{(511-526)/T_k} = e^{(-15)/490.05} = 0.969855.$$

Suppose, our randomly generated number is 0.755436, which implies that our proposed solution is kept. We then update our temperature T_k and proceed.

When we continued this for 1000 iterations, we found the tour

$$\mathbf{x} = 13 \to 9 \to 15 \to 3 \to 10 \to 16 \to 12 \to 1 \to 8 \to 11$$
$$\to 5 \to 14 \to 4 \to 2 \to 7,$$

which has distance 381. Figure 15.1 shows a graph of the solution values generated over these 1000 iterations. Notice how the solution value varies greatly for the first 400 or so iterations, but then the temperature T_k becomes small enough to reduce the chance a worse solution is accepted. This illustrates the typical behavior of a simulated annealing approach. In Figure 15.2, we have the results of a second run of 1000 iterations, but this time with $\alpha = 0.95$. Note that in this case our final solution is only of length 383, and that we seem to converge in much fewer iterations.

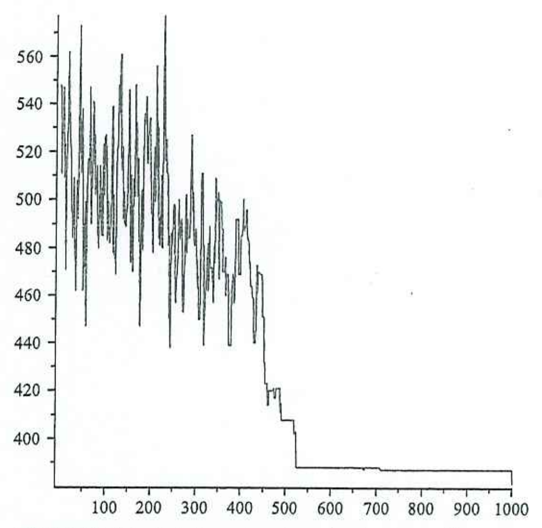

FIGURE 15.1 Solution values from simulated annealing run.

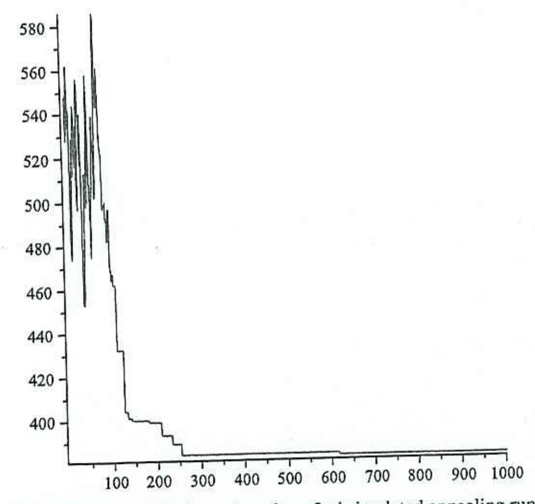

FIGURE 15.2 Solution values from 2nd simulated annealing run.

Simulated annealing has the benefit of being a very simple method to implement, requiring few parameter choices for the user. Its performance has historically been good, but typically other metaheuristic methods have produced better results. In fact, it is often used as part of other metaheuristics. For more information on simulated annealing, see Henderson, et al. [57].

15.3 TABU SEARCH

Simulated annealing moves out of local optimal solutions through a probabilistic approach; *Tabu Search* uses deterministic means (through the use of memory) to do the same. In tabu search, information from previous iterations is used to guide current and future moves. The ideas behind tabu search were first given by Glover [42] and by Hansen [54].

Tabu search works similar to a local improvement approach until it reaches a local optima, at which point it uses short-term memory to move away from this local optima. Recent solutions (or the moves that generated them) are kept in a *tabu list* and are thus ineligible for consideration for a set amount of time. This both restricts the neighborhood of available solutions and helps prevent cycling to recently visited solutions. Elements are kept in the tabu list according to a *tabu tenure*, which dictates both the size of the list and how long of a memory to use. At each iteration, the best nontabu solution from the neighborhood of the current solution is chosen, and this process continues until some termination condition is met, which is typically a maximum number of iterations.

■ EXAMPLE 15.2

To illustrate a tabu search method, we develop one for a TSP, using the specific 15-city data given in (15.1) from Example 15.1. We represent a tour by the sequence of cities visited, and the neighborhood of a given tour T is all tours T_k obtained from T by swapping the order of two cities. For example, given our 15-city TSP and the tour

$$T = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11$$
$$\rightarrow 12 \rightarrow 13 \rightarrow 14 \rightarrow 15,$$

one of the neighbors of T would be obtained by swapping the 4th and 12th cities, resulting in the tour

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 12 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11$$
$$\rightarrow 4 \rightarrow 13 \rightarrow 14 \rightarrow 15.$$

Given this definition of a neighborhood, one possible tabu list would simply be a collection of k recent tours, where k is some user-defined parameter indicating the size of the tabu list (and also the tabu tenure for each tour on the tabu list).

Keeping track of recent solutions in a tabu list can be impractical due to the size of the solutions. Instead, *solution attributes* are typically kept. In this case, the small adjustments made to move from one solution to another are kept in the tabu list, and it is these attributes that are made tabu for a period of time. Unfortunately, this can lead to problems since an attribute can be shared by multiple solutions; hence, instead of making tabu only one solution, many can be made tabu at that same time, including the potential global optimal solution. To remedy this, an *aspiration criteria* can be defined that allows tabu moves to be used. For example, we can set as an aspiration criteria that a tabu move can be used if it generates a better solution than our current best. An outline of generic tabu search approach is given in Algorithm 15.3.

■ EXAMPLE 15.3

For the TSP, instead of using tours as elements of our tabu list, one possible tabu attribute could be to prevent a specific element from being altered for some tenure length. If we use the tour T and its neighbor T_1 from Example 15.2, since the 1st and 10th positions were swapped, we could make swapping the 1st element tabu for some fixed number of iterations. Note that this prevents multiple solutions from being used. For example, if we consider the tour

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 12 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11$$
$$\rightarrow 4 \rightarrow 13 \rightarrow 14 \rightarrow 15$$