can handle problems with only up to about 15 points when this algorithm (which, of course, is so much better than exhaustive enumeration) is used. What is now generally considered as one of the “best” exact algorithms available [HELD 71] becomes impractical with problems involving more than about 70 points.15

Is our version of the TSP (symmetric distance matrix, complete connectivity, triangular inequality) any easier than a general TSP (e.g., one for which the distance matrix can be asymmetrical as in most street networks with one-way streets)? The answer, in principle, is unfortunately “no.” It has been shown [PAPA 77] that the two problems are equally difficult in the sense of NP-completeness. That is, if an efficient exact algorithm can be found for TSP1 (which is how we shall denote our simplified version of the TSP from now on), an efficient algorithm must also exist for the general TSP, as well.

Be that as it may, it is still true that the intuitive appeal of TSP1 greatly facilitates the construction of heuristic algorithms which lead to good, if not necessarily optimal, solutions of these problems. In recent years a few fine such algorithms have been devised which solve quite large TSP’s at very reasonable computational costs [LIN 73, CHRI 76]. Moreover, it is often possible to use such heuristics to obtain manually good solutions to modest-size TSP1’s.

The emphasis on good heuristic solutions is also justified by the fact that, in practice, data inaccuracies and the probabilistic nature of such things as travel times between points make the concept of an “optimal” solution a rather theoretical one.

In the next section we shall review in detail one interesting and well-performing heuristic algorithm. In the process we shall have occasion to discuss a couple of properties of optimal tours for Euclidean TSP1’s (i.e., TSP1’s with Euclidean travel metrics) that further aid the manual solution of these latter problems. A good review of many other TSP heuristic algorithms and comparisons of their performance can be found in [GOLD 79].

6.4.6 Solving TSP1

We shall now present a heuristic algorithm for TSP1 [CHRI 76]. The algorithm consists of three major steps, each of which, in turn, consists of the application of other well-known algorithms that we have already discussed. A fourth step, which usually offers further improvements in the solution, can be easily added to the procedure and will be described separately.

15To our knowledge, the largest problem that has been solved using this method has 121 points [KNUT 76]. However, that problem has a special structure that considerably facilitates its solution.

The basic three steps of our heuristic algorithm produce a tour that is guaranteed to be less than 50 percent longer than the optimal tour. Although this may not sound particularly exciting, it turns out that this is the best “worst-case performance” achieved by any efficient TSP algorithm devised so far.

Consider $n$ points that must be traversed by a TSP1 tour (symmetric distances, complete connectivity, triangular inequality). Then we have:

Heuristic Algorithm for TSP1 (Algorithm 6.6)

STEP 1: Find the minimum spanning tree that spans the $n$ points. Call this minimum spanning tree $T$.

STEP 2: Let $n_o$ of the $n$ nodes of $T$ be odd-degree nodes ($n_o$ is always an even number). Find a minimum-length pairwise matching of these $n_o$ nodes, using a matching algorithm. Let the graph consisting of the links contained in the optimal pairwise matching be denoted as $M$. Create a graph $H$ consisting of the union of $M$ and $T (H = M \cup T)$. Note that if one or more links are contained in both $M$ and $T$, these links will appear twice in $H$.

STEP 3: The graph $H$ is an Eulerian graph, since it contains no odd-degree nodes. Draw an Eulerian circuit on $H$ (beginning and ending at the starting node of the sought-after TSP tour, if such a starting node has already been specified). This Eulerian circuit is the (approximate) solution to the TSP.

Let us now pause to consider the characteristics and properties of the TSP solution whose derivation we just described. It should first be clear that the solution does indeed have the two properties of a traveling salesman tour that we specified earlier: namely, it is a circuit—beginning and ending at the same (specified or not) node—and it does visit each and every node at least once since it is an Eulerian circuit that covers all links of the $n$-node graph $H$ that we have created.

What is now the relationship between the length of the tour that we have just obtained (i.e., the length of $H$) and the length of the actual optimum traveling salesman tour? Although we have not found the optimum solution to the TSP, we can still place a bound on how far our solution can deviate from the optimum solution in the following way.

Let us denote by $L(H)$, $L(T)$, $L(M)$, and $L(TST)$ the length of $H$, $T$, and $M$, as defined, and of the (unknown) optimum traveling salesman tour, respectively.

Theorem: $L(H) < \frac{3}{2} L(TST)$ (6.5)
Proof: Suppose, for the moment, that somehow we have managed to obtain the optimum traveling salesman tour (TST). Since the TST is a circuit that covers all \( n \) points and visits each exactly once, it is clear that if we remove any one of the links on the TST, what will be left will be a spanning tree (not necessarily the minimum one) with \( n \) nodes and \( n - 1 \) links. Since \( T \), by definition, is the minimum spanning tree, and since all distances are positive, it then follows that

\[
L(T) < L(TST)
\] (6.6)

In a similar vein, suppose that we took the TST and identified on it the \( n_0 \) points that were optimally matched in Step 2 of our algorithm. Suppose that we then matched pairwise, in an optimum manner, these \( n_0 \) nodes by using only links that are contained in the TST. Let the set of all links used in this matching be denoted as \( M' \). For the length of the subgraph \( M' \), \( L(M') \), it must then be true that

\[
L(M') \leq \frac{L(TST)}{2}
\] (6.7)

For were this not so, \( M' \) could not possibly be the minimum length matching in TST of the \( n_0 \) nodes.

At the same time, however, it is obvious that \( L(M) \leq L(M') \), since \( M \) is a pairwise matching that is not restricted to the links that are contained in TST, and hence its length should be equal to or less than the length of \( M' \). Now combining (6.6) and (6.7) with the last statement and recognizing that

\[
L(H) = L(M) + L(T)
\]

we finally obtain the inequality

\[
L(H) < \frac{3}{2}L(TST)
\] (6.5)

A fourth step, which can further improve the solution obtained at the end of Step 3, can also be added to Algorithm 6.6:

**STEP 4 (Optional):** Check for nodes of \( H \) (points) that are visited more than once in the Eulerian tour and improve the traveling salesman tour of Step 3 by taking advantage of the triangular inequality.

For instance, if the tour at the conclusion of Step 3 reads, partly, as

\[
[A, B, C, D, B, E, \ldots] \text{ or } [A, C, D, B, E, \ldots]
\]

we now illustrate the application of Algorithm 6.6.

**Example 9: Refuse-Collection Tour**

The sanitation department of a small city faces the following problem. At nine distinct points in the city, solid refuse must be collected from the ends of dead-end alleys which are too narrow for regular-size garbage collection trucks. To deal with this situation the city has purchased a narrower vehicle which is now used exclusively for refuse collection at these nine special
The city wishes to find the shortest route for the vehicle in question. Every day the vehicle must begin its tour at a depot and must return there—presumably to transfer its load to a regular-size truck—after visiting all nine collection points.

The location of the depot is shown as point 1 on Figure 6.21 and the nine collection points are numbered arbitrarily as points 2 through 10. We shall assume here that the actual distances between points can be well approximated by the Euclidean distances between them. These Euclidean distances for all pairs of points are shown, in turn, on Table 6-1.

Now applying our heuristic algorithm for the TSP, we first obtain, at the conclusion of Step 1, the minimum spanning tree, $T$ (see Figure 6.22). The length of $T$ is 258. (This already provides a lower bound on the length of the optimum traveling salesman tour.) There are six odd-degree nodes on $T$: 2, 3, 4, 6, 7, and 10. These nodes must be pairwise-matched in Step 2.

After completion of Step 2, the optimal pairwise matchings are 10-7, 2-3, and 6-4, for a total length of 143 units for the links in $M$. The graph $H$ that results from the union of $M$ and $T$ is shown on Figure 6.23. The total length of $H$ is 401 units, and one possible tour of this length is the tour $[1, 2, 3, 2, 4, 6, 5, 7, 10, 9, 8, 7, 1]$. Now applying Step 4 we note that nodes 2 and 7 are visited twice. The sequences $[1, 3, 2, 4, ... ]$ and $[1, 2, 3, 4, ... ]$ must then be compared with the subsequences $[1, 3, 2, 4, ... ]$ and $[1, 2, 3, 4, ... ]$ with respect to visiting node 7. A tie exists between the two former subsequences (we choose $[1, 3, 2, 4, ... ]$ arbitrarily), whereas the subsequence $[1, 2, 3, 4, ... ]$ is the shorter of the two latter ones. Upon completion of Step 4 we thus obtain the tour $[1, 3, 2, 4, 6, 5, 7, 10, 9, ... ]$.

This is a simplified version of a problem actually encountered in Brookline, Massachusetts, by a colleague of the authors.

**TABLE 6-1** Distance matrix for refuse-collection example.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>25</td>
<td>43</td>
<td>57</td>
<td>43</td>
<td>61</td>
<td>29</td>
<td>41</td>
<td>48</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>0</td>
<td>29</td>
<td>34</td>
<td>43</td>
<td>68</td>
<td>49</td>
<td>66</td>
<td>72</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>29</td>
<td>0</td>
<td>52</td>
<td>72</td>
<td>96</td>
<td>72</td>
<td>81</td>
<td>89</td>
</tr>
<tr>
<td>4</td>
<td>57</td>
<td>34</td>
<td>52</td>
<td>0</td>
<td>45</td>
<td>71</td>
<td>71</td>
<td>95</td>
<td>99</td>
</tr>
<tr>
<td>5</td>
<td>43</td>
<td>43</td>
<td>72</td>
<td>45</td>
<td>0</td>
<td>27</td>
<td>36</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>6</td>
<td>61</td>
<td>68</td>
<td>96</td>
<td>71</td>
<td>27</td>
<td>0</td>
<td>40</td>
<td>66</td>
<td>62</td>
</tr>
<tr>
<td>7</td>
<td>29</td>
<td>49</td>
<td>72</td>
<td>71</td>
<td>36</td>
<td>40</td>
<td>0</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>8</td>
<td>41</td>
<td>66</td>
<td>81</td>
<td>95</td>
<td>65</td>
<td>66</td>
<td>31</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>48</td>
<td>72</td>
<td>89</td>
<td>99</td>
<td>65</td>
<td>62</td>
<td>31</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>71</td>
<td>91</td>
<td>114</td>
<td>108</td>
<td>65</td>
<td>46</td>
<td>43</td>
<td>46</td>
<td>36</td>
</tr>
</tbody>
</table>

**FIGURE 6.22** Minimum Spanning tree for example. Odd-degree nodes indicated with *. 

**FIGURE 6.21** Depot and nine points to be visited.
FIGURE 6.23 Ten-Node problem after the matching of odd-degree nodes.

FIGURE 6.24 Solution that results from the heuristic algorithm.

FIGURE 6.25 True optimum solution.

8, 1) (shown in Figure 6.24), with a total length of 371 units. This we shall consider to be our "final" solution to the refuse-collection problem.

The true optimal solution to this particular traveling salesman problem happens to be the tour [1, 3, 2, 4, 5, 6, 10, 9, 8, 7, 1], with a total length of 331 units. Thus, our solutions at the end of Step 3 and Step 4 were 21 percent and 12 percent, respectively, inferior to the optimum (Figure 6.25).

It should also be clear that following completion of Step 4, one can often detect, by inspection, additional local permutations that lead to further improvements in the solution. For instance, from Figure 6.24 it is obvious that the subsequence [..., 4, 5, 6, 7, ...] is preferable to the one currently used, namely [..., 4, 6, 5, 7, ...]. Indeed, this change alone leads to a tour of length 347, or less than 5 percent longer than the optimum.

6.4.7 Euclidean TSP

When travel is Euclidean in TSP1, we have the Euclidean (or "geometrical") TSP. It has been shown that, theoretically, the Euclidean TSP is equally hard with the general TSP, in the same sense that TSP1 is just as hard as the general TSP [PAPA 77].

Nonetheless, the Euclidean TSP is probably the easiest version of TSP for finding good approximate solutions, either manually or with the aid of the computer. For one, since the Euclidean TSP is just a special case of TSP1, all
heuristics that have been designed for TSP1 (such as Algorithm 6.6) can also be used for the Euclidean TSP. Second, the following two properties hold in the case of the Euclidean TSP.

**Property 1:** The optimum traveling salesman tour does not intersect itself.

**Property 2:** Let \( m \) of the \( n \) points in the Euclidean TSP define the convex hull (see below) of the points. Then the order in which these \( m \) points appear in the optimum traveling salesman tour must be the same as the order in which these same points appear on the convex hull.

The validity of Property 1 is obvious: two intersecting links on a tour can always be replaced by two nonintersecting links whose total length is guaranteed to be less due to the triangle inequality (see Figure 6.26).

**FIGURE 6.26** The traveling salesman tour can be improved by substituting the intersecting links \((a, c)\) and \((b, d)\) by \((a, b)\) and \((c, d)\).

With regard to Property 2 (see Figure 6.27), the convex hull of a set of points in a two-dimensional Euclidean space is defined as the smallest (in terms of area) convex polygon that includes all the points in the set. For the case shown in Figure 6.27, a tour (that consists, in part, of the sequence \( \{\ldots, A, \ldots, C, \ldots, B, \ldots, D \ldots\} \) cannot be optimum according to Property 2, since the points \( A, B, C, \) and \( D \) do not appear in the tour in the same order as in the convex hull \( ABCDEF \). The validity of Property 2 is a direct consequence of Property 1. (Why?)

It is obvious that the foregoing two properties reduce enormously the number of candidate tours that must be considered in a geometrical TSP, when the number of points \( n \) is large. Because it is much easier for a person than for a computer to detect tour link intersections (as well as violations of Property 2), the two properties are particularly convenient to use when approximate manual solutions to geometrical TSP's are attempted. Indeed, they provide the conceptual basis for some very intuitive heuristic algorithms for the Euclidean TSP [CROE 58, WIOR 75] and for an excellent "man-machine interactive" algorithm for vehicle routing problems [KROL 72], which will be described briefly later in this chapter.

Finally, we note that for Euclidean TSP's Property 1 can easily be incorporated in Step 4 of Algorithm 6.6 by modifying the procedure to say: "Check for nodes of \( G \) (points) which are visited more than once in the Eulerian tour and for intersecting links on the Eulerian tour and improve... . "

### 6.4.8 Probabilistic View of the Traveling Salesman Problem

Before completing this discussion, we state a very useful result on the expected length of an optimum traveling salesman tour under a Euclidean travel metric.

First, it is helpful to make the following observation. Suppose that a traveling salesman tour has been drawn through \( n \) points all of which lie in a given area \( A \). Suppose now that this area is expanded uniformly \( m \)-fold. This can be done by changing the coordinates \((x, y)\) of every point in \( A \) to \((\sqrt{mx}, \sqrt{my})\). The same traveling salesman tour through the same \( n \) points as before will then be "stretched" in length to \( \sqrt{m} \) times its earlier length, for the simple reason that every linear segment in \( A \) has now been multiplied by \( \sqrt{m} \). Thus, quadrupling, say, the area \( A \) in the manner described above will only double the length of the given traveling salesman tour. More generally, we can state that a given traveling salesman tour varies in proportion to the square root of the area in which it is contained. We have seen equivalent results several times already in Chapter 3.

With this observation, we now turn to the result of interest. Assume that \( n \) points are randomly and independently dispersed over an area \( A \) with the
location of each point determined by a uniform distribution over \( A \) (i.e., each point is equally likely to be anywhere in \( A \)). Assume further that an optimum traveling salesman tour has been drawn to cover the \( n \) points in question, and let \( L(n, A) \) be the length of this optimum tour through the \( n \) points in \( A \). The following has then been shown to be true whenever the assumptions above hold [BEAR 59]:

**Theorem**

\[
\lim_{n \to \infty} \frac{E[L(n, A)]}{\sqrt{n}} = K \sqrt{A}
\]

(6.8)

where \( K \) is a constant. In fact, \( L(n, A) \) itself tends asymptotically with probability 1 to \( K \sqrt{nA} \) as \( n \to \infty \):

\[
L(n, A) \sqrt{n} \xrightarrow{n \to \infty} K \sqrt{A} \quad \text{(with probability 1)}
\]

(6.9)

Recently, \( K \) has been estimated as being approximately equal to 0.765 [STEI 78]. (An earlier set of simulation experiments had led to the estimate \( K \approx 0.75 \) [EILO 71].)

Like all limit theorems, the one above must be used carefully. For instance, the value of \( n \) which is “large enough” (for the quantity \( 0.765 \sqrt{nA} \)) to provide good approximations to the expected length of the optimal traveling salesman tour) depends on the shape of the area in which the \( n \) points are distributed uniformly. For “fairly compact and fairly convex” areas (see also Section 3.7.1) it seems that surprisingly small values of \( n \) may be adequate. For example, \( n = 15 \) or larger is quite adequate for equilateral triangles, circles, or squares (see [EILO 71]).

While the proof of the limit theorem above is difficult, it is quite easy to devise fairly intuitive arguments that lead to results of the form \( K \sqrt{nA} \) [EILO 71]. One of our homework problems develops such an argument, which also contains some of the key characteristics of the formal proof of the theorem.

The approximation formula \( 0.765 \sqrt{nA} \) is a very useful one for:

1. Preliminary planning of urban collection and delivery systems (i.e., for “sizing up” the requirements for vehicle fleets, estimating the number of points that can be served with given resources, etc.)
2. Assessing when a solution to a TSP obtained through some heuristic approach is probably “close enough” to the optimum.
3. Deriving approximate asymptotic estimates of the expected value of the optimal solutions to problems that are similar but not quite the same as the TSP.

**Example 10: Planning for a Parcel Service**

A parcel pickup and delivery company serves an extended metropolitan region. It estimates that on an average working day it will be making about 100 pickup or delivery visits to homes in the region, randomly and uniformly distributed over a 10- by 10-mile area. Company vehicles have an effective travel speed of 9 miles/hr. On-the-site time for each pickup or delivery visit averages 10 minutes. The effective working day for vehicle drivers is 6 hours and 30 minutes long. We assume that all parcels picked must eventually be brought to the central depot of the company for processing and shipment and, conversely, that parcels for delivery are all available at the depot at the beginning of a day. How many vehicles does the company need to satisfy its daily service requirements?

**Solution**

Assuming first that a single vehicle would suffice, we estimate that on the average \( 76.5 \sqrt{100} = 76.5 \) miles would be covered per day by a company vehicle making all pickups and deliveries. This amounts to \( 76.5 \sqrt{9} = 8 \) hours and 30 minutes = 510 minutes worth of travel time per day. Adding to that the 1,000 minutes needed for on-the-site time, we have a requirement for 1,510 vehicle-driver minutes per day or, with 390 minutes per vehicle-driver, for 4 vehicle-drivers per day. This assumes, at the very least, that:

1. In splitting the single traveling salesman tour into four approximately equal tours we do not incur a large penalty in terms of extra travel time. In practice, this may be an optimistic assumption and may necessitate adding a fifth vehicle.
2. There is somebody in the company who can, on a daily basis, design four efficient and approximately equal-length vehicle tours.
3. Vehicle load capacity is sufficient for the equivalent of about 25 pickup or delivery visits per day.
4. The travel metric in this area is approximately Euclidean.

**Example 11: Bounds on the Length of Routes of Dial-a-Ride Buses**

Consider a dial-a-ride bus that is supposed to pick up and deliver \( n \) passengers in a given area, with a distinct origin and a distinct destination point associated with each passenger. The location of each passenger's origin and destination is known beforehand and no new passengers are accepted once the bus has begun its route. (This is referred to in the dial-a-bus literature as a single-vehicle, many-to-many, advance-subscription service.)

The route of the bus through the \( 2n \) points that it must visit is a TSP-type route with the additional restriction that the destination of any given passenger can be visited only after that passenger's origin has been visited. A typical route of this type is shown in Figure 6.28.
Applications of Network Models

Ch. 6

Sec. 6.4 Routing Problems

Pickup 4

Deliver 1

Deliver 2

Deliver 4

Deliver 6

Figure 6.28 Computer-generated optimal tour for a dial-a-ride bus involving the pickup and delivery of seven passengers beginning at a depot (15 points total). Passengers are identified by a serial number [PSAR 78].

By using our theorem we can conclude that, for large n, the expected length of the bus route is bounded from below and above by \( \sqrt{2K \sqrt{nA}} \) and \( 2K \sqrt{nA} \), respectively (\( K = 0.765 \)). For the lower bound, we ignored the destination-after-origin restriction, while for the upper bound we assumed that the bus visits all origins first and then all delivery points. We have also ignored any restrictions that may be related to bus capacity. The fact that the route is not a tour but an open path is not important since, in the limit as \( n \) becomes large, the length of the last leg (from the final delivery point to the origin of the bus) is insignificant by comparison to the length of the rest of the tour. Sharper bounds and more realistic cases for dial-a-ride systems are discussed by Stein who has developed several limit theorems for this problem [STEI 78].

In closing this section, we note that, as long as points are reasonably well "spread around" over a region, the expression \( 0.765 \sqrt{nA} \) often provides good approximations to the length of optimum TSP tours, even in cases when the \( n \) points are not quite independently located or when the probability distribution for the location of individual points is not exactly uniform over the region. For instance, it has been estimated that the optimum tour through 48 major cities, one in each of the 48 continental states in the United States, and Washington, D.C., is 10,070 miles long, assuming Euclidean metric [BEAR 59]. Using instead the formula \( 0.765 \sqrt{nA} \), with \( A \) the area of the continental United States (= 3,022,400 square miles), one obtains 9,310 miles, for an error of only about -7.5 percent, despite the fact that the 49 cities are far from randomly distributed, since most of them are located at the outer boundaries of the region (Atlantic and Pacific coasts, Great Lakes region), with a further disproportionate concentration of almost one third of the cities in a small part of the country (Northeast).

R. Karp has written an excellent paper that shows how, through judicious partitioning of the area in which the points lie, good TSP tours can be constructed in problems that may involve thousands of points by taking maximum advantage of the limit theorem of this section [KARP 77].

6.4.9 Multiroute Problems

We have so far discussed only single-tour problems of the edge-covering or node-covering types. Yet most of the time, we have to deal with the routing of not one but several vehicles that must share the task of providing services to some specific populated area. In solid waste collection, for an obvious example, a sanitation department must route many vehicles through many districts into which a city has been subdivided.

Multiroute problems started to attract attention in the mathematical literature only rather recently (most of the available material dates after 1970). The reasons are manifold: (1) the single-route problems (Chinese postman, traveling salesman) are difficult in themselves and have thus attracted most of the attention directed to this area; (2) multiroute problems are more complex (and thus less inviting) than single-route ones; and (3) very powerful computers are needed even for the most straightforward heuristic approaches to these latter problems—and such computers did not become available until the late 1960s.

It should also be noted at the outset that available algorithms for multirouting are, at this time, almost exclusively heuristic. Generally, they combine single-route algorithms (for node or edge covering, as the case might be) with some method for partitioning the geographical region or the total workload at hand into "smaller" entities which are consistent with some set of problem constraints. In fact, in applications of these algorithms to urban service systems, the overall strategy, as a rule, has been either:

1. To partition the city (or region), first, into districts and then design optimal single routes within each one of the districts; or
2. To design, first, a single "grand optimal route" for the whole city and then subdivide the route into a number of subroutes each to be covered by a separate vehicle.

We shall refer to the former as the "cluster first, route second" approach and to the latter as the "route first, cluster second" approach [BODI 75]. Which is to be preferred depends on the relative efficiency of the algorithms and analytical tools that one can bring to bear on the particular situation at hand. To date, anyway, the "cluster first, route second" approach has been the one most often followed.

In the following sections we shall examine multi-route node-covering problems first since they have attracted a good deal more attention than have multi-route edge-covering problems.

6.4.10 Multi-route Node Covering

It is helpful, at this stage, to present a classification of node-covering problems. While the classical traveling salesman problem is a well-defined one, there is a large number of possible extensions and variations on it, and the names of the different types of problems often become confusing. The basic descriptors of a node-covering problem are three:

1. The number of vehicles (salesmen).
2. The number of tour origins (which are also the eventual destinations).
3. The existence of constraints on such items as individual vehicle (salesman) capacity, the maximum length of a tour, and so on.

In general, problems in which no constraints such as those mentioned under (3) are specified are known as "salesman" problems. Thus, the classical TSP is a problem in which a minimum tour must be designed for a single salesman–vehicle using a single origin–destination with no capacity or tour-length constraints.

What is known as the m-traveling salesmen problem (m-TSP), on the other hand, involves the design of a prespecified number, m, of distinct tours that collectively visit each of the demand points at least once while using a single common origin–destination. The objective is to minimize the total distance covered in the m tours.

When there are constraints of the vehicle-capacity or maximum-distance types, we have a vehicle routing problem (VRP). In these cases the (single or many) origins–destinations are usually referred to as depots. In VRP’s both the required number of vehicles and their routes are, in general, unknown, and the objective is to minimize an objective function that represents some aspect of or the total cost of the system. In some versions, however, the number of vehicles is specified in advance (chosen so that it is expected to satisfy the constraints of the situation) and the objective is to design feasible tours of minimum total length. Having already discussed the TSP, we shall now briefly examine:

1. The m-TSP problem.
2. The single-depot VRP problem.
3. The multidepot VRP problem.

Our choice is dictated by the applicability of these three problems to urban service systems and by the fact that these also seem to be the most often studied variations of multi-route node-covering problems.

6.4.11 m-TSP Problem

Keeping in mind that the m-traveling salesmen problem is, by definition, a single origin–destination problem, it can be easily shown that it can be reformulated as a classical TSP with little effort. Thus, the m-salesmen problem is no more difficult than its one-salesman counterpart, as measured by worst-case computational complexity. (Interestingly, this simple observation was not made until 1973 and 1974 in several independently published papers [BELL 74, ORLO 74, SVES 73].)

Suppose that a m-TSP is given with n points to be visited and with all tours beginning and ending at a common origin V. [Were m = 1 we would thus have a (n + 1)-node TSP.] The equivalent formulation is then obtained by replacing the origin V by m exact copies of it, V₁, V₂, ..., Vₘ, each connected to the other n nodes exactly as was the original origin and with the same distances. That is, if x is any one of the n points to be visited, then

\[ d(V_i, x) = d(V_j, x) = \ldots = d(V_m, x) = d(V, x) \]

However, the links connecting the m copies of the origin to each other are assigned "infinite" length, that is, very large by comparison to all other distances in the problem \[d(V_i, V_j) = \infty\] for all \(i, j = 1, 2, \ldots, m\).

If we now solve this as a classical single-salesman, \((m + n)\)-point TSP, it can be seen that a minimum tour will never use a link connecting two copies of the origin. In other words, in the shortest single-salesman tour, the sequence "\(V_i, V_j\)" followed by \(V_{j'}\) will never appear \((i, j = 1, 2, \ldots, m)\). Then, if the \((m + n)\)-point, single-salesman tour is "folded back" by merging together all copies of the origin into a single node \(V\), the single-salesman tour will decompose into m tours as required by the m-TSP.

Example 12: Two-Salesmen Problem

Consider the problem described by the (symmetric) distance matrix of Table 6-2. (The geometric problem on which the matrix is based is shown in Figure 6.29; ignore the indicated solution for the moment.) Four points are to be
visited from an origin $V$, using two salesmen. Hence, we need two copies of the origin $V$, which we call $V_1$ and $V_2$. The new distance matrix is then shown in Table 6-3. Note that the $V_1$ and $V_2$ rows (and columns) are identical and that we have set $d(V_1, V_2) = d(V_2, V_1) = \infty$, i.e. to a "very large" number by comparison to other distances in a computer solution of the problem. (We have also set all distances from a point to itself to infinity, to prevent any "self-loops" no matter what solution method is used for the TSP.)

Table 6-3 now describes what is in effect a six-point single TSP. Careful consideration of the problem and some trial-and-error comparisons (or application of an exact TSP algorithm) lead to the conclusion that the optimum solution of the TSP consists of the tour $V_1-3-4-V_2-5-6-V_1$, with a total length of 237 units. This tour is shown schematically in Figure 6.30. Merging $V_1$ and $V_2$ into the single original origin $V$ gives Figure 6.29, the solution to the two-TSP, with a length of 237 units.

With regard to the solution of the $m$-TSP problem, it should be noted that, theoretically, the "MST-and-matching" heuristic algorithm that we presented earlier (Algorithm 6.6) is not applicable, even when the original distance matrix is symmetric and satisfies the triangular inequality. The reason is that, because of setting the distances between the copies of the origins to infinity (or to a "very large" number), the expanded matrix—with the $m - 1$ copies of the origin—does not satisfy the triangular inequality.

In practice, however, if one is careful, the algorithm still can be applied with success (in the great majority of cases) to obtain a single TSP tour with the expanded matrix. The only cases when the algorithm fails occur when more than half of the odd nodes of the minimum spanning tree on completion of Step 1 of Algorithm 6.6 are nodes that correspond to copies of the origin. This, in turn, would mean that in matching odd degree nodes (Step 2) we would be forced to match two copies of the origin to each other—leading to a tour of infinite length.

We conclude our discussion of the $m$-TSP by noting that, irrespective of whether or not the original distance matrix satisfies the triangular inequality,

$$L(m\text{-TSP}) \geq L(\text{TSP}) \quad \text{for all } m > 1 \quad (6.10)$$

where $L(m\text{-TSP})$ and $L(\text{TSP})$ are, respectively, the lengths of the optimum $m$-salesmen and single-salesman tours for any given problem. To see the
validity of (6.10) simply note that the \( m \)-TSP tour has been shown here to cover \( m - 1 \) points in addition to the original \( n + 1 \) that the 1-TSP tour covers. In fact, we have shown by construction that, more generally,

\[
L(m\text{-TSP}) \geq L((m - 1)\text{-TSP}) \quad \text{for all } m > 1
\]  

6.4.12 Single-Depot VRP

We shall examine next the following version of the vehicle routing problem. Let there be \( n \) demand points in a given area, each demanding a quantity of weight \( Q_i \) \((i = 1, 2, \ldots, n)\) of goods to be delivered to it (goods are assumed indistinguishable but for their weight). The goods in question are stored at a depot, \( D \), where a fleet of vehicles is also stationed. Vehicles have identical maximum weight capacities and maximum route-time (or distance) constraints. They must all start and finish their routes at the depot, \( D \). The problem is to obtain a set of delivery routes from the depot, \( D \), to the various demand points to minimize the total distance covered by the entire fleet. It is assumed that the weights \( Q_i \) \((i = 1, \ldots, n)\) of the quantities demanded are less than the maximum weight capacity of the vehicles and we require that the whole quantity \( Q_i \) demanded at a given point \( i \) be delivered by a single vehicle (i.e., we do not allow for the possibility that one third, say, of \( Q_i \) will be delivered by one vehicle and the remaining two thirds by another).

Obviously, the words “supply” and “quantity supplied” can be substituted for “demand” and “quantity demanded,” in which case the depot becomes a collection point. Thus, the VRP applies equally well to solid waste collection from a specific set of points and to parcel delivery to a set of points. A notable recent application, for instance, has been in routing of newspaper distribution vehicles delivering editions of a well-known newspaper to newsstands in an urban area [GOLD 77].

We also note that in specific applications of the VRP, either the maximum weight or the maximum route-time constraints may be relaxed. However, both usually play a role. For instance, in the newspaper delivery problem just mentioned, one constraint, in addition to the maximum number of newspapers that a vehicle can carry, was that all deliveries to newsstands must be made within an hour of press time.

When neither constraint applies, the VRP reduces to a traveling salesman problem: if the objective is simply to minimize total distance, it is a 1-TSP [from (6.10)]; if the number of vehicles to be used is specified, it is a \( m \)-TSP. It can thus be seen that TSP's can be viewed as special cases of VRP's. By inference, VRP's can be expected to be more difficult than TSP's as far as optimum solutions are concerned. Indeed, although several versions of VRP's have been formulated as mathematical programming problems by various investigators, the largest vehicle routing problems of any complexity that have been solved exactly reportedly involved less than 30 delivery points [CHR1 74]. By contrast, the heuristic approaches that we shall describe next can be used even with thousands of delivery points.

**Heuristics for the single-depot VRP.** By far the best-known approach to the VRP problem is the “savings” algorithm of Clarke and Wright. Its basic idea is very simple. Consider a depot \( D \) and \( n \) demand points. Suppose that initially the solution to the VRP consists of using \( n \) vehicles and dispatching one vehicle to each one of the \( n \) demand points. The total tour length of this solution is, obviously, \( 2 \sum_{i=1}^{n} d(D, i) \).

If now we use a single vehicle to serve two points, say \( i \) and \( j \), on a single trip, the total distance traveled is reduced by the amount

\[
s(i, j) = 2d(D, i) + 2d(D, j) - [d(D, i) + d(i, j) + d(D, j)] = d(D, i) + d(D, j) - d(i, j) \quad (6.12)
\]

The quantity \( s(i, j) \) is known as the “savings” resulting from combining points \( i \) and \( j \) into a single tour. The larger \( s(i, j) \) is, the more desirable it becomes to combine \( i \) and \( j \) in a single tour. However, \( i \) and \( j \) cannot be combined if in doing so the resulting tour violates one or more of the constraints of the VRP.

The algorithm can now be described as follows.

**Clarke–Wright Savings Algorithm (Algorithm 6.7)**

**STEP 1:** Calculate the savings \( s(i, j) = d(D, i) + d(D, j) - d(i, j) \) for every pair \((i, j)\) of demand points.

**STEP 2:** Rank the savings \( s(i, j) \) and list them in descending order of magnitude. This creates the “savings list." Process the savings list beginning with the topmost entry in the list (the largest \( s(i, j) \)).

**STEP 3:** For the savings \( s(i, j) \) under consideration, include link \((i, j)\) in a route if no route constraints will be violated through the inclusion of \((i, j)\) in a route, and if:

a. *Either*, neither \( i \) nor \( j \) have already been assigned to a route, in which case a new route is initiated including both \( i \) and \( j \).

b. *Or*, exactly one of the two points \((i \ or \ j)\) has already been included in an existing route and that point is not interior to that route (a point is interior to a route if it is not adjacent to the depot \( D \) in the order of traversal of points), in which case the link \((i, j)\) is added to that same route.
c. Or, both $i$ and $j$ have already been included in two different existing routes and neither point is interior to its route, in which case the two routes are merged.

**STEP 4:** If the savings list $s(i, j)$ has not been exhausted, return to Step 3, processing the next entry in the list; otherwise, stop: the solution to the VRP consists of the routes created during Step 3. (Any points that have not been assigned to a route during Step 3 must each be served by a vehicle route that begins at the depot $D$ visits the unassigned point and returns to $D$.)

**Example 13: Refuse-Collection, Revisited**

Consider once again the 10-point refuse-collection problem examined earlier (Example 9). The depot is node (point) 1 and the nine points to be visited are now recognized to pose different requirements in terms of the expected quantity of refuse to be collected on each daily tour. The pertinent distance and savings data are shown in Table 6-4. Because the distance matrix is symmetric,

\[
\begin{array}{cccccccccc}
  & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
 1 & - & 25 & 39 & 48 & 25 & 18 & 5 & 0 & 1 & 5 \\
 2 & 43 & 29 & - & 48 & 14 & 8 & 0 & 3 & 2 & 23 \\
 3 & 57 & 34 & 52 & - & 47 & 15 & 3 & 6 & 20 & \\
 4 & 43 & 43 & 72 & 45 & - & 77 & 36 & 19 & 26 & 49 \\
 5 & 61 & 68 & 96 & 71 & 27 & - & 50 & 36 & 47 & 86 \\
 6 & 29 & 49 & 72 & 71 & 36 & 40 & - & 39 & 46 & 57 \\
 7 & 41 & 66 & 81 & 95 & 65 & 66 & 31 & - & 78 & 66 \\
 8 & 49 & 72 & 89 & 99 & 65 & 62 & 31 & 11 & - & 83 \\
 9 & 71 & 91 & 114 & 108 & 65 & 46 & 43 & 46 & 36 & - \\
10 & & & & & & & & & & \\
\end{array}
\]

The processing of the savings list now proceeds as follows. The largest savings is associated with link (6, 10), so a tour consisting of {1, 6, 10, 1} is created. The second entry in the list is associated with link (9, 10). The initial tour is therefore expanded to {1, 6, 10, 9, 1} since the conditions under Step 3 are satisfied by such an expansion. Next is the entry for link (8, 9) and expansion of the route to {1, 6, 10, 9, 8, 1} also turns out to be acceptable. However, expansion to {1, 5, 6, 10, 9, 8, 1}, as suggested by the next entry for link (5, 6), is impossible since such a route would imply a load of 24 units (> 23). Thus, link (5, 6) is rejected. So is link (8, 10)—both 8 and 10 are already in the tour; and (7, 10)—since 10 is interior to the tour. The appearance of link (4, 5) as the next entry in the savings list leads to formation of another tour {1, 4, 5, 1}. Next, the inclusion of point 7 in the first tour is acceptable and that tour is expanded to {1, 7, 6, 10, 9, 8, 1} with a load of 23 (meaning that this tour cannot be expanded further).

Proceeding in the same way, points 3 and then 2 are successively added to the second tour, ending up with the tour {1, 7, 6, 10, 9, 8, 1}, with a load of 19 units. Since all points have been included in a tour, that also means the completion of our procedure. The two tours are shown in Figure 6.31 and their combined length (total distance traveled) is 397 units.

**TABLE 6-5** Quantities of refuse to be collected at each point for Example 13.

<table>
<thead>
<tr>
<th>Node</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**TABLE 6-6** Savings list for Example 13.

<table>
<thead>
<tr>
<th>Link</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>(6, 10)</td>
<td>86</td>
</tr>
<tr>
<td>(9, 10)</td>
<td>83</td>
</tr>
<tr>
<td>(8, 9)</td>
<td>78</td>
</tr>
<tr>
<td>(5, 6)</td>
<td>77</td>
</tr>
<tr>
<td>(8, 10)</td>
<td>66</td>
</tr>
<tr>
<td>(7, 10)</td>
<td>57</td>
</tr>
<tr>
<td>(4, 5)</td>
<td>55</td>
</tr>
<tr>
<td>(6, 7)</td>
<td>50</td>
</tr>
<tr>
<td>(5, 10)</td>
<td>49</td>
</tr>
<tr>
<td>(3, 4)</td>
<td>48</td>
</tr>
<tr>
<td>(2, 4)</td>
<td>48</td>
</tr>
<tr>
<td>(6, 9)</td>
<td>47</td>
</tr>
</tbody>
</table>

The distances $d(i, j)$ and the savings $s(i, j)$ can both be exhibited in a single $9 \times 10$ matrix (no savings are associated with the depot, i.e., node 1). In Table 6-4, the savings $s(i, j)$ are shown above the diagonal. For instance,$$s(4, 6) = d(1, 4) + d(1, 6) - d(6, 4) = 57 + 61 - 71 = 47 \]$$Table 6-5 lists the quantities of refuse to be collected from each of the nine nodes 2 through 10. We assume that the capacity of each vehicle is equal to 23 units and apply the third step of the Clarke-Wright algorithm to the savings list, as shown in Table 6-6. No constraint other than the maximum-capacity one is assumed to exist.

As the reader may have already surmised, the Clarke-Wright algorithm can be programmed to run very efficiently and, since it involves very simple manipulations of the data set, it can be used with large problems. Because
Consider again the nine-point refuse-collection problem of the previous example and suppose that we make a single change in the earlier problem by setting vehicle capacity to 16 units (instead of 23).

Solution

The reader may wish to verify that the solution produced through the savings algorithm now becomes:

<table>
<thead>
<tr>
<th>Route</th>
<th>Load</th>
<th>Distance Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 6, 10, 9, 1)</td>
<td>15</td>
<td>191</td>
</tr>
<tr>
<td>(1, 3, 4, 5, 1)</td>
<td>15</td>
<td>183</td>
</tr>
<tr>
<td>(1, 2, 7, 8, 1)</td>
<td>12</td>
<td>146</td>
</tr>
</tbody>
</table>

520 = total

This solution is shown in Figure 6.32. By observing that tours are partially included in each other and that they intersect three times, we are led to

**Example 13 (continued)**

nodes are added to routes one or two at a time, an additional advantage of the algorithm is that it is possible to check whether each addition would violate any set of constraints, even when that set is quite complicated. For instance, besides the constraints on maximum capacity and maximum distance, other constraints might be included, such as a maximum number of points that any vehicle may visit.

On the other hand, there is no guarantee that the solution provided by such a naive algorithm will be anywhere close to the optimum. While experience has shown that the algorithm performs quite well most of the time, it is possible to devise "pathological" cases for which the Clarke-Wright solutions are very poor indeed. However, it is often possible to improve considerably, by inspection, a set of VRP tours produced by the savings algorithm. In fact, a powerful interactive "man-machine" approach has been developed for that purpose [KROL 72]. In this approach, a computer "suggests" a solution using the savings algorithm and projects that solution on a television screen. The human operator then attempts to improve on this solution using his/her knowledge of the problem as well as such geometrical properties of good tours as those we have already discussed for the Euclidean TSP (Section 6.4.8). The operator, using a light pen, suggests these improvements to the computer, which, in turn, comes up with a new solution to the VRP; and so on.

**FIGURE 6.31** Final solution to the refuse-collection VRP, using Algorithm 6.7, when vehicle capacity is 23 units.

**FIGURE 6.32** Final solution to refuse-collection VRP using Algorithm 6.7, when vehicle capacity is 16 units.
attempt some modifications seeking an improvement on the earlier solution. A natural modification to attempt, for instance, is the following:

<table>
<thead>
<tr>
<th>Route</th>
<th>Load</th>
<th>Distance Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 3, 2, 4, 1}</td>
<td>15</td>
<td>163</td>
</tr>
<tr>
<td>{1, 5, 6, 7, 1}</td>
<td>14</td>
<td>139</td>
</tr>
<tr>
<td>{1, 10, 9, 8, 1}</td>
<td>13</td>
<td>159</td>
</tr>
<tr>
<td></td>
<td></td>
<td>461 = total</td>
</tr>
</tbody>
</table>

This solution is both feasible and covers 11 percent less distance than the initial Clarke–Wright solution (Figure 6.33).

FIGURE 6.33 A better solution when the vehicle capacity is 18 units.

Several alternatives to the Clarke–Wright algorithm have been proposed. One that seems to produce good solutions to VRP's can be summarized as follows [GILL 74b].

**Sweep Algorithm for the VRP (Algorithm 6.8)**

Assume that polar coordinates are available for all points to be visited by the vehicles (the depot, for instance, might be used as the origin in the coordinate system). The points then can be ordered in terms of increasing angle by sweeping (clockwise or counterclockwise) a ray initially drawn from the depot to some arbitrary point—known as the seed point. Routes are then drawn up by adding demand points to a route as these demand points are swept: beginning at the seed (whose angle can be set to 0), points are included in a route as they are swept, until the load capacity of a vehicle precludes addition of the next point swept to the current route. That point then becomes the seed for the next route and the process is completed when all points have been swept (i.e., included in a route). Once the points that form each route are available, a TSP algorithm can be used to determine the order of point traversal for each individual route. Thus, the sweep algorithm is a good example of the “cluster first, route second” approach.

**Example 13 (continued)**

The improved solution (total distance covered = 461 units, Figure 6.33) obtained in the last example—with a vehicle capacity of 16—would have resulted from the sweep algorithm had we designated point 3 as the seed point and then swept the other eight points in a clockwise direction.

The major disadvantage of the sweep algorithm is that it does not generate the vehicle tours as it processes the nodes. This is done only after the nodes that constitute each tour have been specified and thus becomes a time-consuming procedure. In addition, whenever there are constraints on the maximum tour lengths as well, some clusters may prove to lead to tours that are unacceptably long. Finally, the algorithm requires Euclidean distances and satisfaction of the triangular inequality throughout.

In general, the Clarke–Wright algorithm seems to enjoy an advantage in terms of both efficiency and flexibility over other available VRP algorithms and has been used extensively. The method has been programmed as the VSPX (Vehicle Scheduling Program) package by IBM and is thus available commercially. The algorithm can be rendered even more powerful through a simple modification that forces the algorithm to generate several alternative solutions and through careful organization and storage of the information it utilizes [GOLD 76].

**6.4.13 Multidepot VRP**

The existence of multiple depots (or multiple origins in unconstrained, traveling-salesman-type problems) poses the additional requirement of assigning demand points to specific depots. This new “degree of freedom” further increases the computational complexity of the problem.

The approach to multidepot VRP problems so far has been based entirely on heuristic algorithms, of which the most common are of the “cluster first, route second” variety: first, demand points are assigned to depots; then, single depot VRP's are solved for each depot.
The best known of these approaches [GILL 74a] assigns demand points to depots in the following way: For each demand point \( i \), we compute the quantity

\[
    r(i) = \frac{d'(i)}{d''(i)}
\]

where \( d'(i) \) and \( d''(i) \) are the distances from \( i \) to the nearest and second nearest of its depots. A threshold value \( \delta \) such that \( 0 < \delta < 1 \) is specified next and compared to each \( r(i) \). If \( r(i) \leq \delta \), then the demand point \( i \) is immediately assigned to its nearest depot. However, if \( r(i) > \delta \), then the demand point is reserved for more careful consideration. After all points \( i \) such that \( r(i) \leq \delta \) have been assigned to depots, and therefore clusters have been formed around the depots, points with \( r(i) > \delta \) are processed again: if two points \( j \) and \( k \) have already been assigned to a given depot, say depot \( D_j \), then inserting point \( i \) between nodes \( j \) and \( k \) on a route originating (and terminating) at \( D_j \) increases the length of that route by \( d_{jk}(i) = d(j, i) + d(i, k) - d(j, k) \). Point \( i \) is then assigned to the depot associated with the minimum of the quantities \( d_{jk}(i) \), for all pairs of points \( (j, k) \) already assigned to a depot. When all demand points have been assigned to a depot in this manner, a single-depot VRP algorithm can be used to design the vehicle routes.

An extension of the savings algorithm to multidepot VRP’s that avoids the early partitioning of the demand points into clusters around depots has also been developed [TILL 72]. The latter algorithm has been recently combined with the cluster-forming approach outlined above into an efficient algorithm capable of handling large-scale problems [GOLD 76].

6.4.14 Multiroute Chinese Postman Problem

Just as in the case of node covering, multiroute edge-covering problems are very meaningful and applicable in the urban environment. Urban areas are obviously subdivided on a routine basis into smaller districts that can be covered by a single mailman or refuse-collection truck or parking-meter reader or snowplowing truck. This districting aspect is an integral part of the multiroute Chinese postman problem. This problem is usually referred to as the constrained Chinese postman problem (CCPP), since the need to subdivide an area into many routes arises due to some constraint(s), such as the maximum distance that a mailman can cover walking during a normal day or the maximum weight or volume of solid waste that a refuse collection truck can carry or, very often, other limits on some measures of workload that have been agreed on in a labor contract.

The CCPP has not been investigated extensively to date, but practical approaches to it—in the context of the delivery of urban services—have been suggested for both undirected [STRI 70] and directed [BELT 74] networks.

Because of the relative ease with which the single-tour Chinese postman problem can be solved, the “route first, cluster second” strategy seems to be the favored one in this case: a giant tour is first found and then divided into \( m \) subtours, where \( m \) is the number of available vehicles. However, with no “best way” available for breaking up the giant tour into shorter subtours, this approach depends to a large extent on the ability and experience of the analyst. In fact, the approach described below for an undirected network [STRI 70] is most effective when carried out manually with the assistance of a good map.

The key to the success of the approach is to subdivide the graph \( G' \), on which the large, single tour is drawn (see Section 6.4.4) in such a way as not to create odd-degree nodes on the boundaries between subtours. Since \( G' \) has been derived by applying a CP algorithm to the original graph \( G \), \( G' \) has no odd-degree nodes. Therefore, all the nodes in the interior of subtours will be even-degree nodes and the partitioning process can create odd-degree nodes only on the boundaries between subtours. To avoid this, it is important to draw continuous boundaries for each subtour, so that an even number of edges is incident on each node. The following describes informally a possible heuristic approach:

Constrained Chinese Postman “Algorithm” (Algorithm 6.9)

**STEP 1:** Using a CP algorithm, create an Eulerian graph from the given network whose edges are to be covered.

**STEP 2:** Sketch out roughly the boundaries of the \( m \) subtours in accordance with the given constraints on tour lengths, vehicle capacities, and so on.

**STEP 3:** Carefully draw a continuous boundary for each subtour so that an even number of edges is incident to every node.

It is clear that the procedure above does not really describe an algorithm in the strict sense but rather outlines a procedure for obtaining a good solution to the CCPP.

**Example 14: Tour of the Mailman, Revisited**

Consider again the mailman’s problem that we used in an earlier example (Figure 6.12). The modified graph \( G' \) which we found by applying our CP algorithm for the solution of the single tour was shown in Figure 6.18 and is copied in Figure 6.34. It is a network with no nodes of odd degree and with total edge length equal to 3,830 distance units.
Applications of Network Models

Ch. 6

Facility Location Problems

FIGURE 6.34 Tentative partitioning of the single-CP tour into three approximately equal subtours.

Suppose now that an upper limit of 1,500 distance units is placed on the length of a mailman's tour. We then attempt to subdivide the single 3,830-unit tour into three approximately equal tours, each of which satisfies the 1,500-unit limit. (Alternatively, it might have been specified that the district in question must be covered by three men.)

On Figure 6.34 the rough outlines of three approximately equal-length tours are sketched in accordance with Step 2 of the CCPP algorithm. Note that these outlines may overlap since they serve only as an aid in defining the approximate physical boundaries of the subtours. In Step 3 the three subtours are designed in detail with continuous boundaries to ensure both the existence of an Eulerian tour and an increase in the total distance covered, which is as small as possible. The three subtours shown in Figure 6.35 are 1,210, 1,300, and 1,320 units long. Their total length in this particular case turns out to be exactly equal to the length of the single tour from which they were derived.

Two disadvantages of the approach that we just illustrated are readily apparent. First, some trial-and-error work may be required before a set of feasible tours is obtained. This is due to the fact that the subdivision of the tour is initially made by inspection alone. Second, the algorithm above does not take into consideration the distances involved in getting to each district from the central station (post office, depot, etc.) and back. These distances—or, better, the time required in practice to cover them—are considered to be second-order-effect quantities.

6.5 FACILITY LOCATION PROBLEMS

We turn next to another extensive category of urban service system problems, those concerned with determining good locations for the stationing of service vehicles or the construction of major facilities. These problems arise in the context of both routine and emergency services, but the objectives are usually different in the two cases.

As one might readily suspect, the “goodness” of a location depends on the measure of effectiveness being used. To take an extreme example, the center of a town might be considered an ideal location for a post office but would certainly be a very poor choice for use as a garbage-incineration point. In the case of the post office a reasonable objective might be to minimize the average walking distance to the building for town residents—hence the choice of the center of the town. For the garbage incinerator, a more...
appropriate objective would be to maximize the minimum distance between the chosen point and any home or building in the town.

In the following sections we shall use the measure of effectiveness at hand as our principal guide to the classification of facility location problems. We shall examine here problems in the following three categories:

1. **Median problems.** Here a prespecified number of facilities must be located so as to minimize the average distance (or the average travel time or the average travel cost) to or from the facilities for the population of their users. Median problems arise very often in the context of facility construction for delivery of nonemergency services (e.g., post offices, transportation terminals, telephone interchanges, “little town halls,” offices for government agencies dealing extensively with the public, etc.).

2. **Center problems.** Here a prespecified number of facilities must be located so as to minimize the maximum distance (or time or cost), to or from the facilities, that any user will have to travel. Center problems (also sometimes referred to as minimax problems) are more applicable in the context of emergency urban services, notably emergency medical care, fire fighting, and emergency repair services.

3. **Requirements problems.** These are problems in which certain standards of performance have already been prespecified for a service system and one seeks the number of facilities required to meet these standards as well as the locations of these facilities. Obviously, this type of problem is a more general one than the median or center problems and is applicable to both emergency and nonemergency services.

In the course of discussing these three types of problems, we shall find opportunities to mention some variations, extensions, or combinations of the three themes described above, all of which lead to significant applications in the urban service field.

### 6.5.1 Basic Model

Network models of an urban or metropolitan area are particularly convenient for the discussion of facility location problems. We shall therefore use such models throughout this section. Specifically, we shall be representing the various transportation arteries as links of a network and their intersections as nodes on it. Thus, travel is restricted to take place solely along the links and nodes of a network. A further assumption will be that demands for services will be generated only at a finite number of points, also designated as a set of nodes on the network. The latter assumption may initially appear to limit the potential usefulness of the models. However, one can always place as many nodes as desired along the links of the network to represent demand-generation points, and thus the models can be made as detailed and realistic as called for by the case at hand.

We shall use a “demand weight” $h_j$ to indicate the rate (or “intensity”) at which demands for service originate from node $j$. Otherwise, our notation will not differ from the one used so far.

#### 6.5.2 Median Problems

Let us consider an undirected network $G(N, A)$ with $n$ nodes. Let $k$ be some positive integer ($k = 1, 2, 3, \ldots$) and let us choose $k$ distinct points on the graph $G$ to be indicated as the set $X_k = \{x_1, x_2, \ldots, x_{k-1}, x_k\}$. We shall then indicate by $d(X_k, j)$ the minimum distance between any one of the points $x_i \in X_k$ and the node $j$ on $G$. That is,

$$d(X_k, j) = \text{Min}_{x_i \in X_k} d(x_i, j) \quad (6.13)$$

We now define the $k$-medians of network $G$ as follows:

**Definition:** A set of $k$ points $X_k^*$ on $G$ is a set of $k$-medians of $G$ if, for every $X_k \in G$,

$$J(X_k^*) \leq J(X_k) \quad (6.14)$$

where

$$J(X_k) = \sum_{j=1}^{n} h_j d(X_k, j) \quad (6.15)$$

Now, if the $k$ points in $X_k^*$ are to be the points where $k$ facilities providing a given service will be located and if $h_j$, the demand weight of node $j$, is set equal to the fraction of all calls for the service in question that originate from $j$ (i.e., $\sum_{i=1}^{n} h_i = 1$), then finding the $k$-medians, $X_k^*$, of $G$ amounts to finding the set of $k$ locations that minimize the average travel distance to (or from) the facilities by service users. This should be clear from the definition of the function $J(X_k)$ in (6.15), which is now nothing but an expression for the average travel distance. Note, too, that the implicit assumption in all of the above is that demands originating at any given node $j$ will be served exclusively by the facility that is closest to $j$.

We can now state a most important result, known as **Hakimi's theorem** [HAK1 64].

---

17 This concept if very similar to the concept of “atoms” in an urban area, which was used in Chapter 3.
Theorem: At least one set of \( k \)-medians exist solely on the nodes of \( G \).

The practical significance of this theorem is great. It states in effect that the search for the set of the \( k \) optimal locations for the \( k \) facilities can be limited to the node set of \( G \) (i.e., to a total of \( n \) points only) instead of the infinite number of points that lie on the links of \( G \).

The validity of the theorem is obvious for the trivial case when the required number of facilities \( k \) is greater than or equal to the number of nodes \( n \). Then we only have to locate one facility on each node to reduce average travel distance to zero.

We shall now prove the theorem for the special case of a single facility \((k = 1)\). A line of approach similar to the one outlined below can be used to prove the more general case \((k \geq 1)\) (see also Problem 6.9).

Proof (For a single facility, 1-median): Suppose that the optimum location for the single facility is at a point \( x \) which lies on the link \((p, q)\) between nodes \( p \) and \( q \). The distance \( d(x, j) \) between \( x \) and any vertex \( j \in N \) can then be written as

\[
d(x, j) = \min (d(x, p) + d(p, j), d(x, q) + d(q, j))
\]

That is, any node \( j \in N \) will be reached from \( x \in (p, q) \) either through \( p \) or through \( q \). Let now \( P \) be the set of nodes that point \( x \) reaches most efficiently through \( p \) and \( Q \) the set of nodes reached more efficiently through \( q \) (ties can be broken arbitrarily). \( P \) and \( Q \) are thus mutually exclusive sets whose union is equal to \( N \).

Assume now without loss of generality that more users are reached through \( p \) than through \( q \); that is

\[
\sum_{j \in P} h_j \geq \sum_{j \in Q} h_j
\]

(If the opposite is true, the following argument is simply reversed.) Using (6.15) and (6.16), we can then write

\[
J(x) = \sum_{j \in N} h_j d(x, j)
\]

\[
= \sum_{j \in P} h_j (d(x, p) + d(p, j)) + \sum_{j \in Q} h_j (d(x, q) + d(q, j))
\]

\[
= \sum_{j \in P} h_j (d(x, p) + d(p, j)) + \sum_{j \in Q} h_j (d(p, q) - d(x, p) + d(q, j))
\]

\[
= d(x, p) \left( \sum_{j \in P} h_j - \sum_{j \in Q} h_j \right) + \sum_{j \in Q} h_j d(p, j)
\]

\[
+ \sum_{j \in Q} h_j (d(p, q) + d(q, j))
\]

From the definition of the distance \( d(p, j) \), we have

\[
d(p, j) \leq d(p, q) + d(q, j) \quad \text{for all } j \in N
\]

Using the inequality (6.18) in the expression for \( J(x) \) above, we obtain

\[
J(x) \geq d(x, p) \left( \sum_{j \in P} h_j - \sum_{j \in Q} h_j \right) + \sum_{j \in Q} h_j d(p, j)
\]

\[
= d(x, p) \left( \sum_{j \in P} h_j - \sum_{j \in Q} h_j \right) + \sum_{j \in Q} h_j d(p, j)
\]

\[
= d(x, p) \left( \sum_{j \in P} h_j - \sum_{j \in Q} h_j \right) + \sum_{j \in Q} h_j + J(p)
\]

But the term \( d(x, p) \left( \sum_{j \in P} h_j - \sum_{j \in Q} h_j \right) \) is, by assumption (6.17), greater than or equal to zero. Therefore, we conclude that \( J(x) \geq J(p) \), which contradicts the assumption that the 1-median is located at an interior point of the link \((p, q)\). Stated differently, we have proved that we can do “at least as well” by moving the facility from \( x \) to the node \( p \). This also completes our proof.

Example 15: Location of a Single Median and of Two Medians

Consider the facility location problem of Figure 6.36, where a network model of an urbanized area is shown. Nodes \( A \) through \( H \) represent points where demands for service are generated and/or points where major roads in the area intersect. A single facility is to be located in the area and its prospective users will have to travel to the facility to partake of the service provided there. Daily demand figures for the service (in units of 100's) are indicated by the figures in parentheses next to the nodes where they originate. The lengths of the various road segments are also indicated (in miles). Where should the facility be located to minimize the average travel distance to it?

Solution

Because of Hakimi's theorem there are only eight candidate points on the network for the placement of the facility; these are the eight nodes \( A \) through \( H \). By using Algorithm 6.2 (or some other shortest-path algorithm) or, in this case by inspection, we can compute the distance (shortest-path) matrix \([d(i, j)]\) for all pairs of nodes, \( i \) and \( j \), of the graph. The distance matrix is given in Table 6-7.

We next compute the terms \( h_j \cdot d(i, j) \) by multiplying each column of the distance matrix by the weight of node \( j \). The result of that operation is shown in Table 6-8. Note that the entries of the \([h_j \cdot d(i, j)]\) matrix have very real physical meaning. For instance, the entry of row \( B \), column \( D \), indicates that if the facility is located at node \( B \), users from node \( D \) must travel a total of 900 "passenger-miles" a day (= 300 persons \times 3\) miles) to use the facility at \( B \). In view of this, it should now be obvious how the optimum location for the
facility can be found: by summing across the entries for each row \(i\) of the \([h_d(i,j)]\) matrix, we can compute the total distance traveled by users if the facility is located at row \(i\). Normalizing the quantities \(h_d\) by dividing by the total demand (= 15), we can find the average user travel distance associated with each of the eight candidate locations. This procedure is summarized in Table 6-9. The optimum location for the facility is at node \(C\), and the associated average travel distance is 2.67 miles.

What would now happen if two facilities were desired? To solve the 2-median problem we can still take advantage of Hakimi's theorem and consider only sets of points composed of two nodes. With a total of eight nodes, there are \(\binom{8}{2} = 28\) possibilities. Total (or average) distances for each combination of locations can still be obtained directly from the \([h_d(i,j)]\) matrix of Table 6-8: demands from each node will be "assigned" to the facility closest to it (i.e., the one that requires the least amount of travel). Thus, if, for instance, the two facilities are located at nodes \(A\) and \(G\), the amount of travel contributed by users from \(D\) is given by \(\min[h_d(A, D), h_d(G, D)] = \min[6, 18] = 6\), directly from the \([h_d(i,j)]\) matrix.

We list below the total distances associated with a few of the 28 combinations of locations:

\[
\begin{align*}
A, B: & \quad 0 + 0 + 4 + 6 + 4 + 0 + 28 + 5 = 47 \\
C, D: & \quad 6 + 2 + 0 + 0 + 3 + 0 + 20 + 3 = 34 \\
D, G: & \quad 6 + 3 + 2 + 0 + 4 + 0 + 0 + 3 = 18
\end{align*}
\]

By exhaustive consideration of all possibilities, we can reach the conclusion that the solution of the 2-median problem consists of locations at nodes \(D\) and \(G\) for a total distance of 18 units (or an average distance of 1.2 miles). Under this solution, demand from nodes \(A, B, C, D, \) and \(E\) (total of 10 units of demand) is assigned to the facility at \(D\) while demand from \(G\) and \(H\) (total of 5 units) is assigned to \(G\). Thus, the facility at \(D\) assumes double the
load of the facility at $G$. Note also that despite the considerable overall reduction in the average travel distance, service users from nodes $B$, $C$, and $E$ now have to travel farther than they had to with a single facility.

From the discussion above we immediately deduce an algorithm for finding the one-median of an undirected graph $G(N, A)$.

Single-Median Algorithm (Algorithm 6.10)

**STEP 1:** Obtain the minimum distance matrix for the nodes of $G$.

**STEP 2:** Multiply the $j$th column of the minimum distance matrix by the demand weight $h_j$ ($j = 1, 2, \ldots, n$) to obtain the matrix $[h_j \cdot d(i, j)]$.

**STEP 3:** For each row $i$ of the $[h_j \cdot d(i, j)]$ matrix, compute the sum of all the terms in the row. The node that corresponds to the row with the minimum sum of terms is the location for the 1-median.

Algorithm 6.10 can also be used, in principle, to obtain the $k$-medians for any value of $k \geq 1$. Only Step 3 must be modified to provide for consideration of sets of $k$ rows (rather than of single rows) in the manner indicated for the 2-median case in our example.

Unfortunately, the combinations—and attendant required comparisons at Step 3—become too many to handle even with a computer, as soon as the number of nodes, $n$, and number of facility locations required, $k$, reach moderate size. For instance, for $n = 100$ and $k = 5$ there exist some 75,000,000 possible combinations of 5-medians, with the computation of the total distance for each requiring some 500 comparisons at Step 3. Thus, the essentially “brute-force” approach of the modified Algorithm 6.10 soon becomes infeasible for $k > 1$ and more sophisticated approaches must be sought.

Several exact algorithms have been proposed for the $k$-median problem [REVE 70, GARF 74]. Basically, these algorithms attempt to solve efficiently integer programming formulations of the problem. Better known is a conceptually simple heuristic algorithm [TEIT 68], which, however, does not always terminate with the optimum $k$-median solution. We describe below an improved version of that algorithm. It begins by finding the 1-median of the network and then increases the number of medians in steps of one at a time, until they become equal to the required number, $k$ ($k > 1$). Because of Hakimi’s theorem we shall only be concerned with locations on nodes. We shall use $S$ to indicate the set of nodes where medians have been (tentatively) located at any given stage in the execution of the algorithm and $m$ to indicate the number of nodes in $S$. During the execution of the algorithm $m$ will increase from 1 to $k$.

Multimedian Heuristic Algorithm (Algorithm 6.11)

**STEP 1:** Let $m = 1$. Find the 1-median of the network $G(N, A)$ using Algorithm 6.10. Let the 1-median be at node $i$. Set $S = \{i\}$.

**STEP 2 (Facility Addition):** Add a new facility to the current membership of the set $S$ by choosing that location among the nodes in $N - S$, the nodes which are not in $S$, which produces the maximum possible improvement in the objective function as the number of medians increases by 1. Let $m = m + 1$.

**STEP 3 (Solution Improvement):** Attempt to improve the objective function by substituting in a systematic way, one at a time, one of the nodes in $S$ with a node that is in $N - S$. Every time an improved solution is obtained, use this as the new “incumbent” solution, $S$, and repeat Step 3. When all possible single-node substitutions for a set $S$ have been attempted without improving the objective function, go to Step 4.

**STEP 4:** If $m = k$, stop; otherwise, return to Step 2.

**Example 15 (continued)**

We now apply Algorithm 6.11 to the 2-median problem on the network of Example 15. In Step 1 we find the 1-median at $C$ and thus set $S = \{C\}$. In Step 2, we must compare the value of the objective function for the sets of facility locations $\{A, C\}$, $\{B, C\}$, $\{C, D\}$, $\{C, E\}$, $\{C, F\}$, $\{C, G\}$, and $\{C, H\}$.

Working with Table 6.8, the respective values of the objective function are found to be 31, 38, 34, 37, 30, 20, and 29. Thus, $S = \{C, G\}$. In Step 3, we now compare the incumbent solution successively with $\{A, G\}$, $\{B, G\}$, $\{D, G\}$. We obtain our first improvement with $\{D, G\}$ (objective function = 18). So $S = \{D, G\}$ becomes the new incumbent solution and Step 3 is repeated. We compare the new solution to $\{A, G\}$, $\{B, G\}$, $\{C, G\}$, $\{E, G\}$, $\{F, G\}$, $\{H, G\}$, $\{D, A\}$, $\{D, B\}$, $\{D, H\}$ without obtaining any further improvement in the objective function. (Note the meaning of attempting “all possible single-node substitutions.”) Since, in Step 4, $m = 2 = k$, the algorithm stops with the solution $S = \{D, G\}$. In this case this also happens to be the optimum 2-median solution.

Algorithm 6.11 is typical of a number of heuristic network algorithms that use the substitution method to improve initial solutions. For example, one of the best-known algorithms for the traveling salesman problem [LIN 73] begins with an initial tour and improves that tour by substituting one link...
of the initial tour at a time with another link which was not in the initial tour. The solutions obtained from such algorithms are sometimes referred to as 1-optimal, because they cannot be improved by replacing any single member (node, link, or whatever) of the final solution set. Algorithm 6.11 could be easily modified to be 2-optimal (or 3-optimal, etc.) by permitting the substitution in Step 3 of up to two (or three, etc.) nodes of \( S \) at a time (instead of exactly one) with nodes in \( N - S \). The solution would then be at least as good or better than the 1-optimal solution. However, one would expect the cost of using a 2-optimal algorithm to be considerably higher for large-size problems, since the number of substitutions that must be attempted in Step 3 increases greatly. Experience has shown that the \( l \)-optimal version of Algorithm 6.11, as described, is quite effective. It is simple, easy to program, very fast, and usually produces solutions that are close to the optimum.

### 6.5.3 Generalization and Extensions

Since Hakimi's theorem first appeared, that very powerful result has been considerably generalized. The most general version [LEVY 67] defines optimal \( k \)-medians as follows:

**Definition:** Let \( u(d(x, y)) \) be a function representing the utility of covering the distance \( d(x, y) \) on a network \( G \). Then, a set of \( k \) points \( X_k \) on \( G \) is a set of optimal \( k \)-medians of \( G \) if, for every \( X_k \in G \),

\[
J_k(X_k) \geq J_k(X_k)
\]

where

\[
J_k(X_k) = \sum_{i=1}^{k} h \mu(d(X_k, j))
\]

In words, the optimal \( k \)-medians maximize the total utility associated with the travel by all service users of the \( k \) facilities. The following theorem has then been shown to be true:

**Theorem:** At least one set of \( k \)-medians exists solely on the nodes of \( G \) as long as the utility function \( u(\cdot) \) is a convex function of the distance \( d \).

Several other results applicable to practically significant extensions of the basic \( k \)-median problem are also available.

**Example 16:** Locating Supplementary Facilities on an Urban Network

It often happens, particularly with regard to urban transportation services, that important service facilities in a given area are severely congested due to high demand. It may then be deemed desirable to establish a number of secondary facilities, whose sole purpose is to "preprocess" the prospective users of the primary facilities. Those users who choose (or, for that matter, are compelled) to pass through the secondary facilities first, presumably receive some type of "reward" either in the form of faster access to the primary facilities or in the form of faster service once they get there or both. A good example of this type of setup is the often-discussed concept of constructing secondary remote terminals for airports where prospective air passengers will congregate, will go through the check-in procedures, and will then be transferred to the airport. Elaborate plans for this type of system have been drawn up for several major cities (as, for instance, the plans for high-speed links between a terminal on Manhattan and the Kennedy and LaGuardia airports in New York, or between a terminal in downtown London and its contemplated third airport).

Problems such as these, in which primary facilities already exist (and quite often are less than optimally located) and secondary facilities must be established to provide supportive services, are known as "supporting facility" problems. When the location of the supporting facilities must be determined so as to minimize the average travel distance (or time or cost) to all users, then, by analogy to all the above, we have the "supporting \( k \)-medians" problems. It can be shown [MIRC 79a] that supporting \( k \)-medians are optimally located on nodes, irrespective of the locations of primary facilities, as long as the utility of travel is a convex function of travel distance (time, cost). Problem 6.10 provides a detailed example for this type of case.

All that has been said so far on median problems was with reference to undirected graphs. With a few modifications (and some changes in the definitions), essentially the same results could have been obtained for directed graphs (i.e., for cases where some or all of an area's transportation links are used for one-way travel). It is important to realize that, with directed graphs, it makes a difference whether the average distance to be minimized is the distance to the facilities, from the facilities, or the round-trip distance. In fact, the distinction is made in the literature between inward medians (minimize travel to the facilities), outward medians (minimize travel from the facilities), and simply medians (minimize total round-trip travel) [MIRC 79b]. While optimal locations for these three cases will be identical for undirected networks, this will generally not be the case for directed networks.

Finally, we note that Algorithms 6.10 and 6.11 can be used, with minor modifications, for the solution of all the problems discussed in this section.

### 6.5.4 Center Problems

**Example 17:** Locating a Firehouse in a Rural Area

Consider the graph shown on Figure 6.37 which depicts a rural area with five towns, shown as nodes \( A \) through \( E \), connected by a rather sparse transportation network with link lengths in miles also indicated. Town populations in thousands are listed in parentheses next to each node.
FIGURE 6.37 A five-town example of center location.

The towns have entered into a cooperative agreement to obtain joint tire protection for certain types of tires. They are planning to build a tirehouse where a single special-purpose tire engine, yet to be purchased, will be stationed. A considerable amount of discussion has led to the conclusion that the location of the firehouse must be such as to minimize the farthest distance that the tire engine will ever have to travel in responding to a tire alarm. This, indeed, is a quite reasonable objective for an emergency-type service such as the fire department.

Suppose, first, that the location of the tirehouse were restricted to be at one of the five cooperating towns. By first obtaining the minimum-distance matrix \([d(i,j)]\) for the given network and then choosing the row with the minimum maximum entry, we can find the node that minimizes the maximum distance to all other nodes. This procedure is shown in Table 6-10. The optimum location for the facility is town C with a maximum distance of 3 miles to both town A and town E.

TABLE 6-10 Shortest-distance matrix and solution of the vertex center problem for the network of Figure 6.37.

<table>
<thead>
<tr>
<th>From</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Then let us ask whether the unrestricted optimum location for the firehouse is also on node C. That is, if we are free to locate the firehouse at any point on the network, will it still be located at town C?

This turns out to be a rather difficult question to answer. We shall do this after introducing a convenient notation and some definitions.

Let \(G(N, A)\) be an undirected network. (Entirely similar concepts with some minor modifications to account for link directivity apply to directed networks.) Let \(x \in G\) be any point on the network. Then, we denote the distance between \(x\) and the node of \(G\) which is farthest away from it as

\[
m(x) = \max_{j \in N} d(x, j)
\]

where the maximum is taken over all nodes \(j \in N\). We then have the following definitions.

Definition: A point \(x\) on a link \((p, q)\) is a local center if for every \(x \in (p, q)\), including the nodes \(p\) and \(q\),

\[
m(x) \leq m(x_i)
\]

Definition: A node \(i^* \in N\) is a vertex center of graph \(G\), if for every \(i \in N\),

\[
m(i^*) \leq m(i)
\]

Definition: A point \(x^* \in G\) is an unrestricted (or absolute) center of graph \(G\), if for every \(x \in G\),

\[
m(x^*) \leq m(x)
\]

With reference to our last example, we have already found the vertex center of the graph of Figure 6.37 to be at node C. We now wish to find the absolute center of that graph.

An algorithm for finding the absolute center of an undirected graph \(G\) can be simply described as follows.

Single-Center Algorithm (Algorithm 6.12)

**STEP 1:** For each link \(\ell\) of \(G\), find the local center \(x_{\ell}\) of \(\ell\).

**STEP 2:** Among all the local centers \(x_{\ell}\), choose the one with the smallest \(m(x_{\ell})\). That local center is also the absolute center \(x^*\) of \(G\).

Unfortunately, Step 1 of this simple two-step algorithm is a time-consuming one. We illustrate this through our earlier example.

**Example 17 (continued)**

To find all the local centers on the graph of Figure 6.37, we examine each of the five network links separately. The procedure is described in pictorial form in Figure 6.38.