CHAPTER 2
ROUTING OF VEHICLES

The basic routing problem is easy to state. We are given a set of nodes and/or arcs to be serviced by a fleet of vehicles. There are no restrictions on when or the order in which these entities must be serviced. The problem is to construct a low-cost, feasible set of routes—one for each vehicle. A route is a sequence of locations that a vehicle must visit along with an indication of the service it provides.

In Fig. 2.1, a set of vehicle routes that services the 13 demand points is presented. Each node has a demand of one unit, the vehicle capacity is three units, and each vehicle must return to the depot it originated from. Each route can be traversed in either direction.

The routing of vehicles is primarily a spatial problem. It is assumed that no temporal or other restrictions impact the routing decision except (possibly) maximum route-length constraints. This is in contrast to scheduling problems to be described in Chapter 3, where the movement of each vehicle must be traced through both time and space.

Due to the relatively unconstrained nature of these problems and their inherent complexity, they have tantalized and challenged combinatorial analysts and operations researchers for many years. (See Magnanti[455] for an extensive discussion.)

The basic vehicle routing problems discussed in this chapter are:

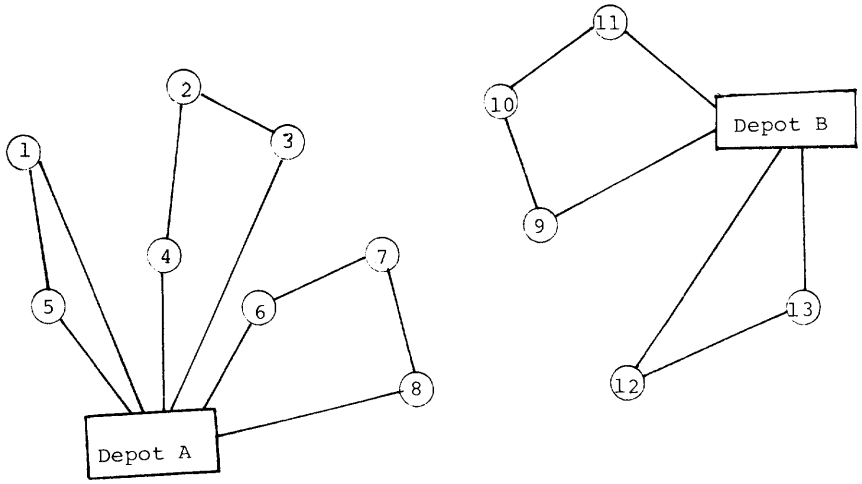
*The Traveling Salesman Problem
*The Chinese Postman Problem
*The M-Traveling Salesman Problem
*The Single Depot, Multiple Vehicle, Node Routing Problem
*The Multiple Depot, Multiple Vehicle, Node Routing Problem
*The Single Depot, Multiple Vehicle, Node Routing Problem with Stochastic Demands
*The Capacitated Chinese Postman Problem.

By way of introduction, we now discuss each of the preceding problems informally. A figure illustrating the spatial configuration of a feasible solution is provided where appropriate. In addition to these basic problems, a number of variants are also described in this chapter.

The *traveling salesman problem* requires the determination of a minimal cost cycle that passes through each node in the relevant graph exactly once. If costs are symmetric, that is, if the cost of traveling between two locations does not depend on the direction of travel, we have a symmetric traveling salesman problem; otherwise, we have an asymmetric or directed traveling salesman problem. A feasible solution to a symmetric traveling salesman problem is given in Fig. 2.2. Every feasible solution has two arcs incident to each node. A feasible solution to an asymmetric traveling salesman problem is given in Fig. 2.3. In this case, note that there is one arc into and one arc out of every node.

The *Chinese postman problem* requires the determination of the minimal cost cycle that passes through every arc of the graph at least one time. A Chinese postman problem is called directed or undirected, depending on whether arcs of the graph are directed or not. Both of these variations can be solved by polynomially-bounded algorithms. The mixed Chinese postman problem has some undirected and some directed arcs; this problem is *NP*-hard. An illustration of a feasible solution to an undirected Chinese postman problem is given in Fig. 2.4.

The *multiple traveling salesman problem* is a generalization of the traveling salesman problem where there is a need to account for more than one salesman (or vehicle). The M vehicles in the fleet are to leave from and return to a common depot. There are no restrictions on the number of nodes that each vehicle may visit except that each vehicle must visit at least one node. An illustration of a feasible solution to a multiple traveling salesman problem with $M = 3$ is given in Fig. 2.5.



Route 1: Depot A - 1 - 5 - Depot A

Route 2: Depot A - 3 - 2 - 4 - Depot A

Route 3: Depot A - 6 - 7 - 8 - Depot A

Route 4: Depot B - 9 - 10 - 11 - Depot B

Route 5: Depot B - 12 - 13 - Depot B

Fig. 2.1. Illustration of routes.

The *single depot, multiple vehicle, node routing problem* (classical vehicle routing problem) asks for a set of delivery routes for vehicles housed at a central depot, which services all the nodes and minimizes total distance traveled. The demand at each node is assumed to be deterministic and each vehicle has a known capacity. The multiple traveling salesman problem is a vehicle routing problem with a fleet of M vehicles each of which has, for all practical purposes, infinite capacity (so that no capacity problems arise). In the example in Fig. 2.6, the demand at each node is one, and vehicle capacity is 3.

The *multiple depot, multiple vehicle, node routing problem* is a generalization of the previous problem in that the fleet of vehicles now must serve D depots rather than just one. All other constraints from the classical VRP still apply. In addition, each vehicle must leave from and return to the same depot. A feasible solution to this problem with two depots using the same data as in Fig. 2.6 is given in Fig. 2.7. In this example, we assume that each depot can house up to two vehicles.

The *single depot, multiple vehicle, node routing problem with stochastic demands* is identical to the classical vehicle routing problem except that the demands are not known with certainty,

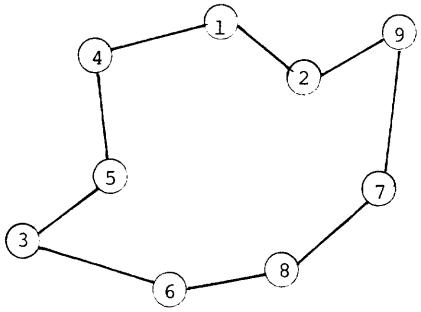


Fig. 2.2. Symmetric case.

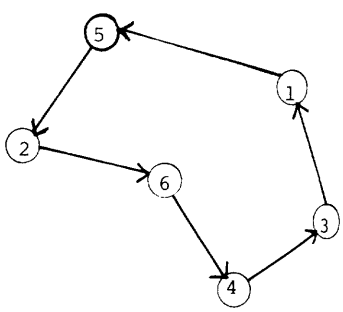


Fig. 2.3. Asymmetric case.

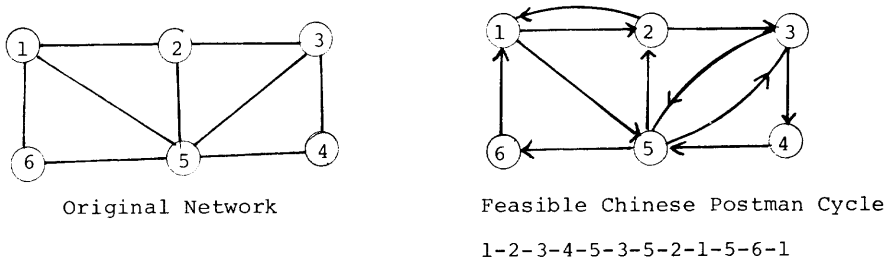


Fig. 2.4. Chinese postman problem.

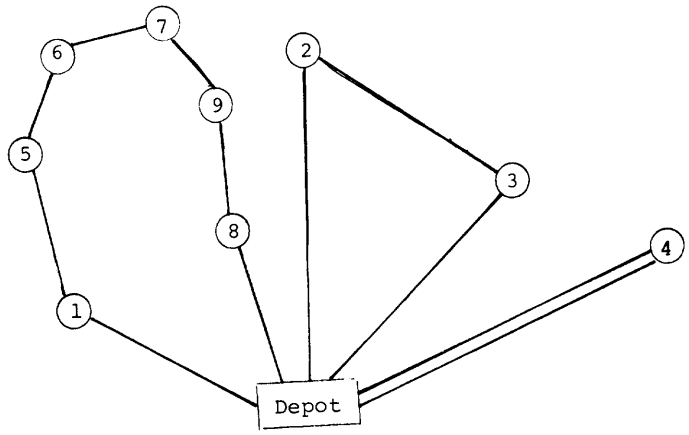


Fig. 2.5. The multiple traveling salesman problem.

but instead come from a specified probability distribution. For example, it may be that node i has demand described by a Poisson distribution with mean λ_i.

The *capacitated arc routing problem* is as follows: given an undirected network with arc demands $q_{ij} \geq 0$ for each arc which must be satisfied by one of a fleet of vehicles, each of capacity W, find the vehicle cycles, each passing through the depot, that satisfy all demands at minimal total cost. In Fig. 2.8, each arc has unit demand and $W = 3$.

In this chapter, we discuss many of the approaches that have been proposed for solving the routing problems already described and others as well. For some of these algorithms we present worst case bounds. Worst case bounds tell us that the worst value of the objective function resulting from the algorithm under consideration is guaranteed not to exceed $K(n)$ times the optimal objective value where $K(n)$ is a function of n (the number of nodes). For optimal algorithms, $K(n)$ is equal to 1 and for heuristic algorithms, $K(n) > 1$ and generally increases with n. Since the determination of worst case bounds usually requires sophisticated mathematical analysis, worst case results are stated and referenced in this chapter, but are not derived in any detail.

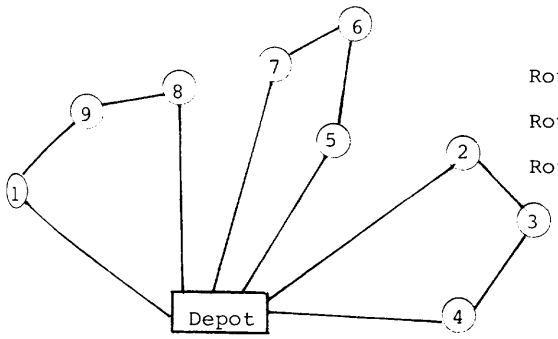
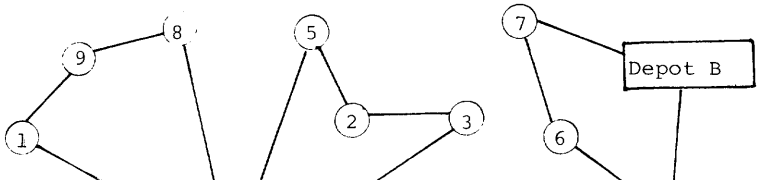


Fig. 2.6. Vehicle routing problem.

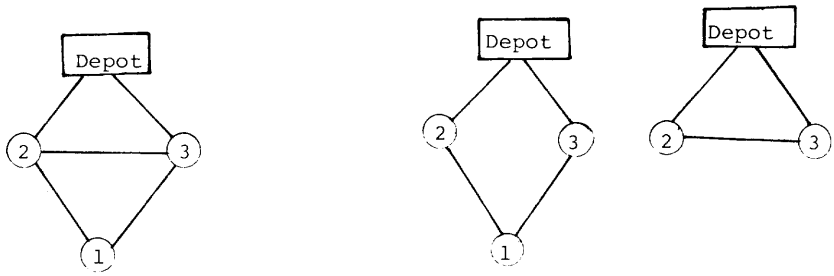


Route 1: Depot A - 1 - 9 - 8 - Depot A

Route 2: Depot A - 5 - 2 - 3 - Depot A

Route 3: Depot B - 7 - 6 - 4 - Depot B

Fig. 2.7. Multiple depot problem.



Original Network Feasible Solution

Fig. 2.8. Capacitated arc routing problem.

The next section discusses the celebrated traveling salesman problem (TSP) and focuses on the routing of a single vehicle. Sections 2.2 and 2.3 treat the solution methodology for this problem including both exact and heuristic techniques. Sections 2.4 and 2.5 discuss single depot multiple vehicle routing problems. Subsequent sections deal with extensions of this problem in the presence of multiple depots, arc routing, and various other scerarios.

2.1. THE TRAVELING SALESMAN PROBLEM (TSP)— INTRODUCTION

Let a network $G = [N, A, C]$ be defined with N the set of nodes, A the set of branches, and $C = [c_{ij}]$ the matrix of costs. That is, c_{ij} is the cost of moving or the distance from node i to node j. The traveling salesman problem requires the Hamiltonian cycle in G of minimal total cost (a Hamiltonian cycle is a cycle passing through each node $i \in N$ exactly once). Many interesting aspects of this problem have been discussed by Bellmore and Nemhauser[61] and Christofides[131]. In addition, connections between statistical mechanics (the study of how systems with many degrees of freedom behave in thermal equilibrium at a finite temperature) and the TSP are explored in a recent paper by Kirkpatrick *et al.*[389].

Karp[377] has shown that the traveling salesman problem (TSP) is *NP*-complete. This remains true even when additional assumptions such as the triangle inequality or Euclidean distances are invoked (see Garey *et al.*[241] and Papadimitriou[532]). These results imply that a polynomially bounded exact algorithm for the TSP is unlikely to exist. Although ingenious algorithms for the TSP have been proposed by Little *et al.*[447], Held and Karp[337], [338] Miliotis[480], Crowder and Padberg[155], and others, they all encounter problems with storage and running time for cases with more than about 100 nodes. An important exception is the recent work of Padberg and Hong[528].

Due to the difficulty of the TSP, many heuristic (approximate) procedures have been developed. These heuristics may be compared analytically, as in the ground-breaking paper of Rosenkrantz, Stearns, and Lewis[568] by studying their worst-case behavior. Alternatively,

computational experimentation may be used to compare the performance of these heuristics, as in the work of Golden *et al.*[302] and Stewart[627]. Golden *et al.*[302] consider a set of problems ranging in size from 25 to 150 nodes and give some guidelines as to which TSP procedure to use in a particular situation. If an approximate, but not an optimal solution is sought, they recommend that the user consider one of the quick tour construction procedures. For more accurate (but more costly) results, a composite procedure is suggested. Finding the optimal solution is feasible only for very small problems with, say, less than 100 nodes. Stewart[627] describes a number of new algorithms designed specifically to perform well on Euclidean problems.

We now proceed to review both optimal and heuristic approaches to the TSP. The next section focuses on an exact approach that is typical of a general methodology for solving combinatorial optimization problems.

2.2. TSP—OPTIMAL APPROACHES

Formulations. Optimal approaches to the TSP are based on mathematical programming formulations for this problem. Indeed, a particular way of formulating the TSP may provide special insights and motivate the exact approach to be used. The following discussion reviews some useful formulations of the TSP.

For the sake of simplicity, we assume that the costs are symmetric (that is, $c_{ij} = c_{ji}$) unless otherwise specified and set $c_{ii} = +\infty$ for $i = 1, 2, \ldots, n$. The problem is to form a tour over all the nodes beginning and ending at the origin, node 1, which gives the minimum total tour distance or cost. For notation, let

$$x_{ij} = \begin{cases} 1 & \text{if arc } i\text{–}j \text{ is in the tour} \\ 0 & \text{otherwise.} \end{cases}$$

An assignment-based formulation of the problem selects the matrix $X = (x_{ij})$ of decision variables so that exactly one arc (i, j) emanates from each node i and exactly one arc (i, j) is directed into each node j. This implies an assignment of each node to its successor node on the tour. It is well known that the assignment requirements, by themselves, do not ensure that the matrix X corresponds to a tour. Instead, the assignment may result in subtours as shown in Fig. 2.9.

To eliminate the possibility of subtours, further restrictions are imposed on the choices for the arc selection matrix X to give the formulation:

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.1}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = b_j = 1 \qquad (j = 1, \ldots, n) \tag{2.2}$$

$$\sum_{j=1}^{n} x_{ij} = a_i = 1 \qquad (i = 1, \ldots, n) \tag{2.3}$$

$$X = (x_{ij}) \in S \tag{2.4}$$

$$x_{ij} = 0 \text{ or } 1 \qquad (i, j = 1, \ldots, n). \tag{2.5}$$

Fig. 2.9. Two subtours.

The set S can be any restrictions that prohibit subtour solutions satisfying the assignment constraints (2.2), (2.3), and (2.5). Such restrictions are called subtour-breaking constraints. Possible choices for S includes:

(1) $S = \{(x_{ij})\colon \sum_{i \in Q} \sum_{j \notin Q} x_{ij} \geq 1$ for every nonempty proper subset Q of $N\}$;

(2) $S = \{(x_{ij})\colon \sum_{i \in R} \sum_{j \in R} x_{ij} \leq |R| - 1$ for every nonempty subset R of $\{2, 3, \ldots, n\}\}$;

(3) $S = \{(x_{ij})\colon y_i - y_j + n x_{ij} \leq n - 1$ for $2 \leq i \neq j \leq n$ for some real numbers $y_i\}$.

Note that S contains nearly 2^n subtour-breaking constraints in (1) and (2), but only $n^2 - 3n + 2$ constraints in formulation (3). Observe that the configuration in Fig. 2.9 satisfies constraints (2.2), (2.3), and (2.5), but not (2.4). That is, it does not represent a tour. The first set of subtour-breaking constraints (1) states that every proper subset Q of nodes must be connected to the other nodes in the network in the solution X. They prohibit the solution in Fig. 2.9 when $Q = \{1, 2, 3\}$. The second set of subtour-breaking constraints (2) implies that the arcs selected in X contain no cycle, since a cycle on nodes R must contain $|R|$ arcs. It excludes the solution in Fig. 2.9 when $R = \{4, 5, 6\}$. The third set of subtour-breaking constraints is more complicated. Adding constraints (3) for arcs 4–5, 5–6, and 6–4 in Fig. 2.9 yields $3n \leq 3(n-1)$, a contradiction. A similar contradiction arises whenever the matrix X contains a subtour solution. Thus, each set of constraints (1), (2), and (3) prohibits subtours.

It is also easy to see that tours are feasible in each case. Constraints (1) and (2) present no difficulties. In constraints (3), let

$$y_i = \begin{cases} t & \text{if node } i \text{ is visited on the } t^{\text{th}} \text{ step in a tour} \\ 0 & \text{otherwise.} \end{cases}$$

Then, each constraint (3) associated with an arc in a tour states that

$$t - (t+1) + n \leq n - 1.$$

Other constraints in (3) only reveal that $y_i - y_j \leq n - 1$ whenever $y_i \leq n$ and $y_j \geq 1$.

A somewhat different formulation due to Gavish and Graves [257] introduces flow variables y_{ij} and assumes that $(n-1)$ units of a good are supplied at the origin, node 1, and that every other node in the network requires 1 of these units. The formulation is:

$$\text{Minimize} \quad \sum_{i=j}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.6}$$

subject to:

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad (j = 1, \ldots, n) \tag{2.7}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1, \ldots, n) \tag{2.8}$$

$$\sum_{j=1}^{n} y_{ij} - \sum_{j=1}^{n} y_{ji} = -1 \qquad (i = 2, \ldots, n) \tag{2.9}$$

$$y_{ij} \leq U x_{ij} \qquad (i, j = 1, \ldots, n) \tag{2.10}$$

$$x_{ij} = 0, 1, \; y_{ij} \geq 0 \qquad (i, j = 1, \ldots, n) \tag{2.11}$$

where $U \geq n - 1$. Constraints (2.7) and (2.8) define an assignment problem, as before. The

"forcing constraints" (2.10) insure that the flow y_{ij} on arc (i, j) is zero if that arc is not selected in the assignment matrix X. The formulation prohibits subtours, since, if node i lies on a subtour not containing the origin, it could not receive a unit of demand from the origin. Formally, suppose that there is a subtour $\{i_1, i_2, \ldots, i_r, i_1\}$ that does not include node 1. If we set $y_{i_1,i_2} = f$, then (2.9) implies that

$$y_{i_2,i_3} = f - 1$$

$$y_{i_3,i_4} = f - 2$$

$$\vdots$$

$$y_{i_r,i_1} = f - r + 1.$$

But this gives $\sum_{j=1}^{n} y_{i_1 j} - \sum_{j=1}^{n} y_{j i_1} = f - (f - r + 1) = r - 1 > -1$, which violates (2.9). Thus, no subtours are admissible. Any tour $\tau = \{i_0 = 1, i_1, i_2, \ldots, i_{n-1}, i_n = 1\}$ is feasible since setting y_{i_{j-1},i_j} for $j = 1, 2, \ldots, n$ and $y_{\alpha\beta} = 0$ otherwise, satisfies all constraints.

These various formulations are valuable for a number of reasons. They help provide insight onto the complexity of the TSP and its relationship to other routing and distribution problems. They suggest algorithmic strategies and heuristic procedures, some of which are obtained by dualizing with respect to certain constraints. Finally, they serve as compact mathematical problem representations.

An exact solution procedure. In this section, we demonstrate a procedure for computing sharp lower bounds on the TSP solution and we describe how to embed this procedure within a branch and bound framework in order to solve the TSP exactly. This approach was first introduced by Held and Karp [337, 338] in a formal sense, and by Christofides [126] informally. Extensions have been proposed by Hansen and Krarup [327] and by Bazaraa and Goode [49]. For a comprehensive treatment of exact approaches to the TSP, see Christofides [134].

Consider the formulation (2.1)–(2.5). Constraints (2.2) and (2.3) can be replaced by

$$\sum_j x_{1j} = 2 \tag{2.12}$$

$$\sum_j x_{ij} + \sum_k x_{ki} = 2 \quad \text{for} \quad i = 1,2,\ldots,n \tag{2.13}$$

$$\sum_i \sum_j x_{ij} = n. \tag{2.14}$$

Constraints (2.12) and (2.13) state that each node has a degree of two. Constraint (2.14) guarantees that a total of n arcs are in the solution. Note that under these new constraints a solution with arcs (1, 2), (2, 4), (4, 3) and (1, 3) is acceptable. Assuming a symmetric distance matrix, the direction on arc (1, 3) can be reversed without changing the objective value in order to obtain a tour. Let problem (P) be the problem represented by (2.1), (2.4), (2.5), (2.12), (2.13) and (2.14).

A lower bound on the solution to problem (P) can be computed if we solve the Lagrangian relaxation obtained from including constraints (2.13) in the objective function. This yields a new objective function of

$$\underset{x}{\text{Minimize}} \quad \sum_i \sum_j c_{ij} x_{ij} + \sum_i \lambda_1 \left\{ \sum_j x_{ij} + \sum_k x_{ki} - 2 \right\}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a vector of Lagrange multipliers. We call this problem (LR_λ).

Rearranging the objective function, we can rewrite it as:

$$(LR_\lambda): \quad \underset{x}{\text{Minimize}} \quad \sum_i \sum_j (c_{ij} + \lambda_i + \lambda_j) x_{ij} - 2 \sum_i \lambda_i$$

subject to: (2.4), (2.5), (2.12), and (2.14).

Let z and $z_D(\lambda)$ be the optimal objective values for problems (P) and (LR_λ), respectively. If x_{ij}^* is an optimal solution to (P), then

$$z_D(\lambda) \leq \sum_i \sum_j c_{ij} x_{ij}^* + \sum_i \lambda_i \left\{ \sum_j x_{ij}^* + \sum_k x_{ki}^* - 2 \right\} = z.$$

Thus, (LR_λ) provides an infinite family of lower bounds on the length of an optimum tour, one for each value of λ. As we will see, for a given λ, $z_D(\lambda)$ is easy to determine. The best of these lower bounds is given by

$$\max_\lambda z_D(\lambda).$$

A key observation, which motivates this approach, is that problem (LR_λ) is substantially easier to solve than problem (P). For any given λ, the arc costs are given by $c'_{ij} = c_{ij} + \lambda_i + \lambda_j$ and the problem is to find a least-cost 1-*tree*. A 1-*tree* is a tree having node set $\{2, 3, \ldots, n\}$ along with two distinct arcs incident to node 1. A least cost 1-tree can be determined in only $O(n^2)$ operations using a minimal spanning tree algorithm on nodes $\{2, 3, \ldots, n\}$ and attaching the two smallest arcs with respect to $C' = (c'_{ij})$ incident to node 1.

There are a number of effective iterative procedures for obtaining $\max_\lambda z_D(\lambda)$. The subgradient method has worked well in a variety of applications and is, perhaps, the most popular procedure for determining λ^*. Details may be found in papers by Held and Karp [337, 338].

In general, the bounds provided from Lagrangian relaxation and the subgradient method are extremely tight and, because of this fact, have been used successfully in branch and bound procedures to solve TSP's of moderate size efficiently. In this setting, we solve a Lagrangian relaxation in place of a linear programming relaxation at each node of the branch and bound tree. Each partitioning of a node in the branch and bound tree includes, and/or excludes, some arcs from the tour. That is, at each node, a set of arcs, A^1, has already been included and another set, A^2, has been excluded from the tour. If the least-cost 1-tree at a particular node is a tour, then it is the optimal tour for the given sets of included and excluded arcs and a feasible solution to problem (P). If, on the other hand, the lower bound at a node exceeds the value of a feasible solution (upper bound), then that node in the branch and bound tree may be discarded and it requires no further consideration.

The Lagrangian relaxation procedure outlined above typifies a general line of attack for combinatorial routing problems. For the TSP, it is possible to devise other Lagrangian relaxation schemes based on a different formulation and/or the relaxation of constraints other than the ones used above. For example, Christofides [134] discusses the assignment problem, matching problem, and shortest n-arc path TSP relaxation in his overview paper. In all cases the role of the relaxation is to provide tight bounds on the optimum value searched for. We will encounter this approach also in the context of the Vehicle Routing Problem discussed later in this chapter.

Perhaps, at this point, it is a good idea to indicate the size of the largest TSP's that have been solved to optimality. Crowder and Padberg [155], using a cutting plane algorithm that generates subtour-breaking and "comb" inequalities as needed, have solved a symmetric 318 node problem in under 50 minutes of computer time on an IBM 370/168. The above-mentioned comb inequalities are due to Chvatal [125] and Grötschel and Padberg [319]. When it comes to asymmetric TSP's the restricted Lagrangian approach of Balas and Christofides [29] seems to be the best available exact procedure known. This algorithm combines features of subtour-elimination and Lagrangian relaxation procedures and has been *remarkably* successful in solving up to 325 node problems. In particular, ten 325 node problems were solved in an

average of 49 seconds on a CDC 7600. The maximum time observed was only 82 seconds.

Since exact approaches to the TSP are, in general, computationally burdensome for large TSP's, a variety of heuristic approaches have found wide use. The next section reviews heuristic techniques for the TSP.

2.3. TSP—HEURISTIC APPROACHES

Heuristics examined. The heuristics we examine fall into three broad classes—tour construction procedures, tour improvement procedures, and composite procedures. *Tour construction procedures* generate an approximately optimal tour from the distance matrix. *Tour improvement procedures* attempt to find a better tour given an initial tour. *Composite procedures* construct a starting tour from one of the tour construction procedures and then attempt to find a better tour using one or more of the tour improvement procedures. Most of these procedures are described in the literature and, hence, will be sketched only briefly; but newer procedures will be studied in more detail. We assume for the sake of simplicity that the costs are symmetric, (i.e. $c_{ij} = c_{ji}$) satisfy the triangle inequality, and are defined for each (i, j) pair, unless otherwise specified.

2.3.1 *Tour construction procedures*

(a) *Nearest neighbor procedure* (Rosenkrantz, Stearns, and Lewis[568]).

Step 1. Start with any node as the beginning of a path.

Step 2. Find the node closest to the last node added to the path. Add this node to the path.

Step 3. Repeat step 2 until all nodes are contained in the path. Then, join the first and last nodes.

Worst case behavior:

$$\frac{\text{length of nearest neighbor tour}}{\text{length of optimal tour}} \leq \frac{1}{2}\left\lceil \lg(n) \right\rceil + \frac{1}{2}$$

where lg denotes the logarithm to the base 2, $\lceil X \rceil$ is the smallest integer $\geq X$, and n is the number of nodes in the network.

Number of computations. The nearest neighbor algorithm requires on the order of n^2 computations.

Comments. In a computational setting, the procedure outlined above may be repeated n times, each time with a new node selected as the starting node. The best solution obtained would then be listed as the answer. Notice that this strategy runs in an amount of time proportional to n^3.

(b) *Clark and Wright Savings* (Clark and Wright[145], Golden [291]).

Procedure

Step 1. Select any node as the central depot which we denote as node 1.

Step 2. Compute savings $s_{ij} = c_{1i} + c_{1j} - c_{ij}$ for $i, j = 2,3, \ldots, n$.

Step 3. Order the savings from largest to smallest.

Step 4. Starting at the top of the savings list and moving downwards, form larger subtours by linking appropriate nodes i and j. Repeat until a tour is formed.

Worst case behavior. The worst case behavior for this approach is known for both a sequential and concurrent version. Golden[289] demonstrates that for a sequential version of this algorithm where at each step we select the best savings from the last node added to the subtour, the worst case ratio is bounded by a linear function in $\lg(n)$. Ong[517] has derived a similar result for the concurrent version.

Number of computations. The calculation of the matrix $S = [s_{ij}]$ in step 2 requires about cn^2 operations for some constant c. Next, in step 3, savings can be sorted into nonincreasing order via the "Heapsort" method of Williams[674] and Floyd[226] in a maximum of $cn^2 \lg(n)$ comparisons and displacements. Step 4 involves at most n^2 operations since there are that many savings to consider. Thus, the Clark and Wright savings procedure requires on the order of $n^2 \lg(n)$ computations.

Comments. Each node may be selected as node 1 for the above procedure (yielding $n^3\lg(n)$ computations). The best solution obtained would be listed as the answer. In practice one can exploit geometric properties when distances are Euclidean; as a result, only a fraction of the almost n^2 saving are calculated which reduces running times by an order of magnitude (see Golden, Magnanti and Nguyen[306] for details).

(c) *Insertion procedures* (Rosenkrantz, Sterns and Lewis [568]).

An insertion procedure takes a subtour on k nodes at iteration k and attempts to determine which node (not already in the subtour) should join the subtour next (the selection step) and then determines where in the subtour it should be inserted (the insertion step). For the first four insertion procedures we discuss, each node in the network can be used as a starting node. Notice that when each node is used as a starting node, the complexity of the entire procedure increases by an order of magnitude (that is, the number of computations is multiplied by n).

(c1) *Nearest insertion*

Procedure

Step 1. Start with a subgraph consisting of node i only.

Step 2. Find node k such that c_{ik} is minimal and form the subtour i-k-i.

Step 3. *Selection step.* Given a subtour, find node k not in the subtour closest to any node in the subtour.

Step 4. *Insertion step.* Find the arc (i, j) in the subtour which minimizes $c_{ik} + c_{kj} - c_{ij}$. Insert k between i and j.

Step 5. Go to step 3 unless we have a Hamiltonian cycle.

Worst case behavior. $\dfrac{\text{length of nearest insertion tour}}{\text{length of optimal tour}} \leq 2.$

Number of Computations. The nearest insertion algorithm requires on the order of n^2 computations.

(c2) *Cheapest insertion*

Procedure

Same as for nearest insertion except that steps 3 and 4 are replaced by the following step.

Step 3′. Find (i, j) in subtour and k not, such that $c_{ik} + c_{kj} - c_{ij}$ is minimal and, then, insert k between i and j.

Worst case behavior: Same as for nearest insertion.

Number of computations. The cheapest insertion algorithm requires on the order of $n^2\lg(n)$ computations.

(c3) *Arbitrary Insertion*

Procedure

Same as for nearest insertion except that in step 3, arbitrarily select node k not in the subtour to enter the subtour.

Worst case behavior.

$$\frac{\text{length of arbitrary insertion tour}}{\text{length of optimal tour}} \leq \lceil \lg(n) \rceil + 1.$$

Number of computations. The arbitrary insertion algorithm requires on the order of n^2 computations.

(c4) *Farthest Insertion*

Procedure

Same as for nearest insertion except that in step 3 replace "closest to" by "farthest from" and in step 2 replace "minimal" by "maximal".

Worst case behavior. Same as for arbitrary insertion.

Comments. Since the worst case result holds for any arbitrary ordering of the nodes, it holds in particular for the ordering induced by the farthest insertion procedure. It is, however, quite possible that a tighter worst case bound can be derived.

(c5) *Quick Insertion or Nearest Addition*

Procedure

Step 1. Pick any node as a starting circuit T, with one node (and 0 edges).

Step 2. Given the k-node circuit T_k, find the node z_k not on T_k that is closest to a node, call it y_k, on T_k.

Step 3. Let T_{k+1} be the $k+1$—node circuit obtained by inserting z_k immediately in front of y_k in T_k.

Step 4. Repeat Steps 2 and 3 until a Hamiltonian circuit (containing all nodes) is formed.

Worst case behavior. Same as for nearest insertion. *Number of Computations*: Same as for nearest insertion.

(c6) *Convex hull insertion*

It has been shown that if the costs c_{ij} represent Euclidean distance and H is the convex hull of the nodes in two-dimensional space, then the order in which the nodes on the boundary of H appear in the optimal tour will follow the order in which they appear in H (see Eilon, Watson–Gandy and Christofides [201] for a discussion of the implications of this result). This observation serves as impetus for the convex hull heuristic procedure (which is discussed along with a host of variants in Stewart's doctoral dissertation [627]). See also the work of Golden and Stewart [308].

Procedure

Step 1. Form the convex hull of the set of nodes. The hull gives an initial subtour.

Step 2. For each node k not yet contained in the subtour decide between which two nodes i and j on the subtour to insert node k. That is, for each such k, find (i, j) such that $c_{ik} + c_{kj} - c_{ij}$ is minimal.

Step 3. From all (i, k, j) found in Step 2, determine the (i^*, k^*, j^*) such that $(c_{i^*k^*} + c_{k^*j^*})/c_{i^*j^*}$ is minimal.

Step 4. Insert node k^* in subtour between nodes i^* and j^*.

Step 5. Repeat Steps 2 through 4 until a Hamiltonian cycle is obtained.

Worst case behavior. Unknown.

Number of computations. This procedure involves, in the worst case, about as much computational complexity as the cheapest insertion algorithm—on the order of n^2 lg(n) computations.

Comments. Wiorkowski and McElvain [685] introduced an insertion algorithm also based on convexity ideas a number of years ago. However, their procedure performed with rather poor accuracy. More recently, Norback and Love [514] have proposed two geometric methods closely related to algorithm (c6).

(c7) *Greatest Angle Insertion*

Procedure

Step 1. From the convex hull of the set of nodes. The hull gives an initial subtour.

Step 2. Choose the node k^* not in the subtour and the arc (i^*, j^*) in the subtour such that the angle formed by the two arcs (i^*, k^*) and (k^*, j^*) is the largest possible.

Step 3. Insert node k^* in subtour between nodes i^* and j^*.

Step 4. Repeat Steps 2 and 3 until a Hamiltonian cycle is obtained.

Worst case Behavior. Unknown. However, Yang[696] has demonstrated the negative result that there does not exist a constant that bounds the ratio of "greatest angle" tour length to optimal tour length.

Number of computations. Same as for cheapest insertion. *Comments*: Norback and Love[514] seem to have first suggested this approach as well as a related procedure known as the eccentric ellipse method.

(c8) *Difference × ratio insertion* (*Or* [518])

Procedure

Same as for greatest angle insertion except that Step 2 is replaced by the following step.

Step 2′. Choose the node k^* not in the subtour and the arc (i^*, j^*) in the subtour such that the product $\{c_{i^*k^*} + c_{k^*j^*} - c_{i^*j^*}\} \times \{(c_{i^*k^*} + c_{k^*j^*})/c_{i^*j^*}\}$ is smallest possible.

Worst case behavior. Unknown

Number of computations. Same as for cheapest insertion.

(d) *Minimal spanning tree approach* (Kim [387])

Procedure.

Step 1. Find a minimal spanning tree T of G.

Step 2. Double the edges in the minimal spanning tree (MST) to obtain an Euler cycle.

Step 3. Remove polygons over the nodes with degree greater than 2 and transform the Euler cycle into a Hamiltonian cycle.

Worst case behavior.

$$\frac{\text{length of MST approach tour}}{\text{length of optimal tour}} \leq 2.$$

Number of computations. This approach requires on the order of n^2 computations.

(e) *Christofides' heuristic* Christofides[132] recently proposed the following interesting technique for solving TSP's.

Procedure

Step 1. Find a minimal spanning tree T of G

Step 2. Identify all the odd degree nodes in T. Solve a minimum cost perfect matching on the odd degree nodes using the original cost matrix. Add the branches from the matching solution to the branches already in T, obtaining an Euler cycle. In this subgraph, every node is of even degree although some nodes may have degree greater than 2.

Step 3. Remove polygons over the nodes with degree greater than 2 and transform the Euler cycle into a Hamiltonian cycle.

Worst case behavior.

$$\frac{\text{length of Christofides' tour}}{\text{length of optimal tour}} \leq 1.5.$$

Cornuejols and Nemhauser[151] have improved this bound slightly (although not asymptotically) in obtaining a tight bound for every $n \geq 3$.

Number of computations. Since the most time-consuming component of this procedure is the minimum matching segment which requires $O(n^3)$ operations, this heuristic is $O(n^3)$. In most cases, the number of odd nodes will be considerably less than n.

(f) *Nearest merger* (Rosenkrantz *et al.* [568])

The nearest merger method when applied to a TSP on n nodes constructs a sequence $S_1, \ldots, S_n$ such that each S_i is a set of $n - i + 1$ disjoint subtours covering all the nodes.

Procedure

Step 1. S_1 consists of n subtours, each containing a single node.

Step 2. For each $i < n$, find an edge (a_i, b_i) such that $c_{a_i b_i} = \min \{c_{xy}$ for x and y in different subtours in $S_i\}$. Then, S_{i+1} is obtained from S_i by merging the subtours containing a_i and b_i. (At each step in the procedure, the two subtours closest to one another are merged.)

Worst case behavior.

$$\frac{\text{length of nearest tour}}{\text{length of optimal tour}} \leq 2.$$

Number of computations. This approach requires on the order of n^2 computations.

Comments. To merge two tours T_1 and T_2 when one or both consist of a single node is trivial. If T_1 and T_2 each contain at least two modes, let (a, b) be an edge in T_1 and (d, e) an edge in T_2 such that

$$c_{ad} + c_{be} - c_{ab} - c_{de}$$

is minimized. Then MERGE (T_1, T_2) is the tour derived from T_1 and T_2 by deleting (a, b) and (d, e) and adding (a, d) and (b, e).

Other tour construction algorithms have been proposed recently by Stinson[634] and Karp[376]. Stinson's heuristic is a modification of Vogel's approximation method which has been used extensively in obtaining an initial feasible solution to the transportation problem.

Karp[376] presents a partitioning algorithm for the TSP in the plane and performs a probabilistic analysis in order to obtain some interesting theoretical results. From a practical point of view, his procedure is a decomposition algorithm which should be capable of solving extremely large TSP's. The key idea is to partition a large rectangle into a number of subrectangles and solve a TSP in each subrectangle. The outcome is an Euler cycle which can be transformed into a tour over all the nodes with minimal effort. A simpler, but somewhat related approach, is described by Platzman and Bartholdi[546] they include a BASIC program listing in the appendix.

2.3.2 *Tour improvement procedures*

Perhaps the best known heuristics for the TSP are branch exchange heuristics. The 2-opt and 3-opt heuristics were introduced by Lin[444] in 1965 and the k-opt procedure, for $k \geq 3$, was presented by Lin and Kernighan[446] more recently. These branch exchange heuristics work as follows:

Step 1. Find an initial tour. Generally, this tour is chosen randomly (this need not, however, be the case) from the set of all possible tours.

Step 2. Improve the tour using one of the branch exchange heuristics.

Step 3. Continue Step 2 until no additional improvement can be made.

The branch exchange procedure terminates at a local optimum. For a given k, if we define a k-change of a tour as consisting of the deletion of k branches in a tour and their replacement by k other branches to form a new tour, then a tour is k-optimal if it is not possible to improve the tour via a k-change. Steps 2 and 3 would generate this k-optimal tour. Since the 2-opt exchange procedure is weaker than the 3-opt exchange procedure, it will generally terminate at an inferior local optimum. Similarly, a k-opt exchange procedure will generally terminate with a better local optimum than will a 3-opt exchange procedure for $k \geq 4$. Figure 2.10 illustrates a 2-opt exchange. Note that the exchange is well-defined at each step. Once we have decided to drop arcs (A, B) and (E, F) from the current solution in order to reduce tour length, we must introduce arcs (A, E) and (B, F), since introducing (A, F) leads to a subtour solution and introducing (A, B) leads to the original solution. Note that symmetry is required here since the direction on arcs (B, C), (C, D), and (D, E) is reversed.

These branch exchange procedures are important since they illustrate a general approach to heuristics for combinatorial optimization problems. In addition, they have been used to generate excellent solutions to large-scale traveling salseman problems in a reasonable amount of time.

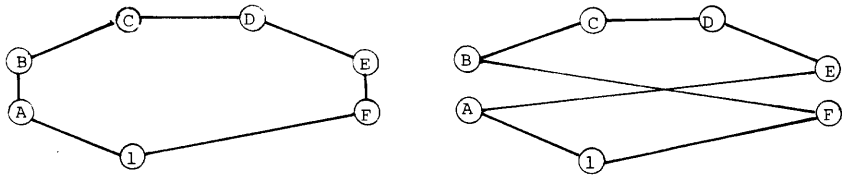


Fig. 2.10. Initial tour and feasible 2-opt swap.

In terms of worst case performance, Rosenkrantz, Stearns, and Lewis[568] give some partial results. For example, they show (assuming the triangle inequality) that a 2-optimal tour can be almost twice the length of an optimal tour.

Roughly speaking, a 3-opt procedure requires n times the work that a 2-opt procedure needs. To obtain close approximations to the length of the optimal tour using this strategy, one should repeat a 3-opt procedure for a number of starting tours[568]. Computationally, this approach becomes burdensome quickly. We, therefore, consider a class of heuristic methods which is computationally less expensive but as accurate as the repeated use of the 3–opt procedure. These are composite procedures.

2.3.3 *Composite procedures*

The basic composite procedure can be stated as follows:

Step 1. Obtain an initial tour using one of the tour construction procedures.

Step 2. Apply a 2-opt procedure to the tour found in Step 1.

Step 3. Apply a 3-opt procedure to the tour found in Step 2.

The composite procedure is relatively fast computationally and gives excellent results.

The idea behind the composite procedure is to get a good initial solution rapidly and hope that the 2-opt and 3-opt procedures will then find an almost-optimal solution. In this way, the 3-opt procedure which is the computationally most expensive step of the three, need only be used once.

The following are some possible variants of the composite procedure outlined above:

Variant 1. Run the composite procedure without Step 2.

Variant 2. Run the composite procedure without Step 3. This variant gives very fast running times and very accurate results but would not be expected to perform as accurately as the three-step composite procedure.

Variant 3. Run the composite procedure a few times, using different tour construction algorithms in Step 1. This variant, although more expensive computationally than the three-step composite procedure, is expected to give better results.

Variant 4. In Step 1 of the composite procedure, only run the tour construction procedure from a small number of starting nodes. Then perform Steps 2 and 3 from the best of these solutions.

All of the heuristic algorithms described in this section are intended for symmetric TSP's (although some can be applied to asymmetric problems as well). Algorithms specifically designed for asymmetric TSP's have been proposed by Akl[4], Frieze *et al.*[236], Karp[373], Van Der Cruyssen and Rijckaert[659], and Kanellakis and Papadimitriou[370]. We discuss only the first of these methods here.

2.3.4 *Akl's directed TSP approach*

The procedure described in this section was especially designed for directed (i.e. asymmetric) TSP's by Akl[4] and is closely related to Christofides' heuristic. Before we describe the procedure, we need to define a *minimal directed spanning graph* (MDSG). An MDSG is a subgraph spanning n nodes of a complete directed graph with weights $w(i, j)$, which has minimum weight and forms a tree when directions are ignored. In other words, when directions are ignored, the graph is a minimal spanning tree with arc lengths $l_{ij} = \min\{w_{ij}, w_{ji}\}$.

Given a directed TSP on graph G, the algorithm stated below computes a nearly optimal solution. First, we present the steps briefly and then we comment on them in more detail.

Table 2.1. Directed TSP distance matrix

	1	2	3	4	5	6
1	∞	7	65	68	34	81
2	19	∞	22	27	59	29
3	14	43	∞	62	77	65
4	76	53	64	∞	6	51
5	39	58	38	27	∞	13
6	46	67	27	11	38	∞

Procedure

Step 1. Find a MDSG of G.

Step 2. Add a set of arcs to the MDSG in order to make the directed graph thus obtained Eulerian.

Step 3. Find an Euler cycle in this directed graph.

Step 4. Transform the Euler cycle into a Hamiltonian cycle.

Step 1 involves solving a minimal spanning tree (MST) on the graph G which is now taken to be undirected with arc costs $l_{ij} = \min\{w_{ij}, w_{ji}\}$. If an arc (i, j) appears in the resulting MST it is given the direction (i, j) if $l_{ij} = w_{ij}$ and (j, i) if $l_{ij} = w_{ji}$. This directed subgraph is an MDSG of G. In Step 2, we identify nodes with excess in-degree as supply points and nodes with excess out-degree as demand points and solve an associated transportation problem in order to balance the graph. Step 3 is straightforward (see Nijenhuis and Wilf[512]). Step 4 is also quite easy: Assume the Euler cycle is n_1–n_2–. . .–n_{l-1}–n_l where the n_i's are not necessarily distinct nodes. A Hamiltonian cycle can be formed by starting at node n_1, moving to the right and introducing a node into the Hamiltonian cycle only if it appears for the first time. In order to generate all Hamiltonian cycles obtainable from the Euler cycle, two copies of the Euler cycle are placed contiguously, n_1–n_2–. . . .–n_{l-1}–n_l–n_1–n_2–. . .–n_{l-1}–n_l and the method just described is repeated l times, every time using the next node in $\{n_1, n_2, \ldots, n_l\}$ as a start node.

We now illustrate the procedure by quickly working through the small example shown in Table 2.1. Note that in this example, which is taken from Akl[4], the triangle inequality is not preserved.

The results of Steps 1, 2 and 4 are displayed in Fig. 2.11–2.13. The length of the resulting tour is 94 units.

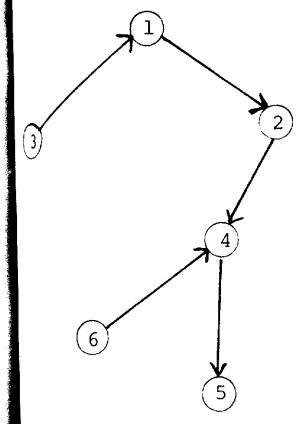


Fig. 2.11. Directed TSP step 1.

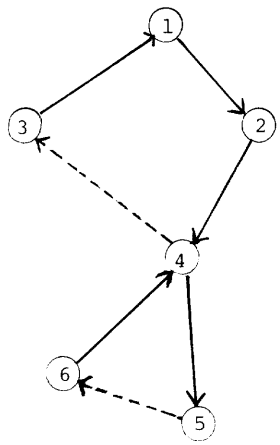


Fig. 2.12. Directed TSP step 2.

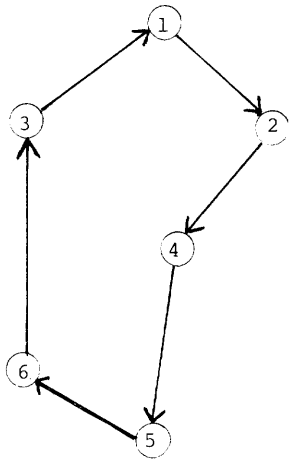


Fig. 2.13. Directed TSP step 4.

2.3.5 *Heuristic approaches to the time-constrained traveling salesman problem*

Suppose that each ordered pair (i, j) of nodes has associated with it a net profit and a travel time. In the time-constrained TSP, the objective is to find a subtour that begins and ends at the origin (node 1), which maximizes profit while requiring less than τ units of time. This is a much more realistic statement of the problem a traveling salesman faces and it has been the focus of research attention by Cloonan[146], Gensch[264], and, more recently, Golden *et al.*[304].

In particular, let $p = [p(i, j)]$ be the net profit matrix and $t = [t(i, j)]$ be the time matrix and let both of these be $N \times N$. We now present an iterative heuristic solution procedure. At each iteration l, we let P and T denote the total profit and total time of the subtour just generated. Also, ΔP and ΔT are the changes in total profit and time associated with each permissible insertion. Our initial idea was to use the ratio $\Delta P/\Delta T$ as a measure of the attractiveness of an insertion. However, since these changes might be negative, we sought an alternative approach.

Procedure TCTSP

Step 0. Set l, P, and T to 0. Initialize R_l. Choose α.

Step 1. For each node i, compute

$$\Delta T = t(1, i) + t(i, 1),$$

and

$$\Delta P - R_0 \Delta T = \{p(1, i) + p(i, 1)\} - R_0\{t(1, i) + t(i, 1)\}.$$

Find the node i^* such that $\Delta T \leq \tau$ and $\Delta P - R_0 \Delta T$ is maximized. If no such node exists, stop. Otherwise, form the subtour $1 - i^* - 1$ and record it. Set P to $P + \Delta P$ and T to $T + \Delta T$.

Step 2. $l \leftarrow l + 1$

$$R_l = \alpha(P/T) + (1 - \alpha)\, R_{l-1}.$$

Step 3. For each node k not in the present subtour and adjacent nodes i and j in the subtour, compute

$$\Delta T = t(i, k) + t(k, j) - t(i, j),$$

and

$$\Delta P - R_l \Delta T = \{p(i, k) + p(k, j) - p(i, j)\} - R_l\{t(i, k) + t(k, j) - t(i, j)\}.$$

Find the i, k, j triple i^*, k^*, j^* such that $T + \Delta T \leq \tau$ and $\Delta P - R_l \Delta T$ is maximized. If no such triple exists, go to Step 5. Otherwise, insert k^* between i^* and j^* in the subtour and record the subtour. Set P to $P + \Delta P$ and T to $T + \Delta T$.

Step 4. Go to Step 2.

Step 5. From all the subtours recorded, select the one with the largest P.

Several points still need clarification. First, the ratio P/T is the worth of a unit of time in the current subtour. R_l is the best estimate of the worth of a unit of time at iteration l. That is, R_l is an estimate which takes into account all previous ratios P/T but weights the more recent ones more heavily. R_o should represent an educated guess (possibly based on preliminary analysis) of the profit to time ratio for the optimal subtour. The expression for R_l is the standard exponential smoothing model that is used in a variety of forecasting situations. Finally, in order to promote flexibility and generate a number of heuristic solutions, one may vary α between 0 and 1 and select different initial values for R_0. We remark that each application of this insertion procedure requires on the order of N^3 operations in the worst case.

Consider a traditional traveling salesman problem with arc lengths $l(i, j)$, in which we seek a minimum-length Hamiltonian tour over the points. Procedure TCTSP will generate a near-optimal solution to the TSP if we set

$$p(i, j) = \tau - l(i, j), \text{ and}$$

$$t(i, j) = 0 \text{ for all } i \text{ and } j,$$

Table 2.2. TCTSP example

p(i,j)

	1	2	3	4	5
1	-	160	125	35	53
2	-48	-	86	2	58
3	-35	106	-	80	65
4	-50	97	152	-	60
5	-37	148	125	55	-

t(i,j)

	1	2	3	4	5
1	-	26	26	15	14
2	8	-	30	19	16
3	6	28	-	12	16
4	8	30	25	-	16
5	4	24	26	13	-

where $\tau \geq 2 \max\{l(i, j)\}$. In this case, the heuristic becomes the well-known cheapest-insertion algorithm. The transformation from $l(i, j)$ to $p(i, j)$ is needed in order to ensure that each node will be included in the subtour obtained.

An example of a time-constrained TSP with $\tau = 72$ is given in Table 2.2.
If we let $R_0 = 1$ and $\alpha = 1$, Procedure TCTSP determines the subtour 1–2–5–4–1 with a total profit of 223 and a total time of 63. The intermediate subtours are 1–2–1 and 1–2–5–1 with total profits of 112 and 181 and total travel times of 34 and 46 units. The *optimal* solution is 1–3–4–5–1 with a total profit of 228 and a total travel time of 58.

2.4. SINGLE DEPOT/MULTIPLE VEHICLE ROUTING

2.4.1 *Introduction*

In addition to its practical relevance, which has already been illustrated, the routing and scheduling of vehicles is an area of importance to operations research theoreticians as well. Recent research in this field has provided significant breakthroughs in problem formulations and in the design and analysis of algorithms. These advances have an important bearing on future research in routing and scheduling, as well as in related combinatorial and computational research endeavors.

In this section we focus primarily on *node* routing problems in which demands at *nodes* must be satisfied by a *fleet* of vehicles that is domiciled at a central depot. In a later section, we study multi-vehicle *arc* routing problems.

2.4.2 *The multiple traveling salesmen problem*

The multiple traveling salesmen problem (MTSP) is a generalization of the traveling salesman problem (TSP) that comes closer to accommodating real world problems where there is a need to account for more than one salesman (vehicle). Multiple traveling salesmen problems arise in various scheduling and sequencing applications. For example, this framework could be used to develop the basic route structure for a pickup or delivery service (perhaps a school bus or rural bus service—see Chapter 4). It has already proved to be an appropriate model for the problem of bank messenger scheduling, where a crew of messengers picks up deposits at branch banks and returns them to the central office for processing (see [638]).

In the multiple traveling salesmen problem, M salesmen are to visit the nodes of a given n node network in such a way that the total distance traveled by all M salesmen is a minimum. Each salesman must travel along a subtour of the nodes, which includes a common depot, and every node except the depot must be visited exactly once by exactly one salesman. The mathematical programming formulation of the (MTSP) displayed below is a natural extension of the assignment-based formulation of the traveling salesman problem.

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.15}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = b_j = \begin{cases} M & \text{if } j = 1 \\ 1 & \text{if } j = 2,3,\ldots,n \end{cases} \tag{2.16}$$

$$\sum_{j=1}^{n} x_{ij} = a_i = \begin{cases} M & \text{if } i = 1 \\ 1 & \text{if } i = 2,3,\ldots,n \end{cases} \tag{2.17}$$

$$X = (x_{ij}) \in S \tag{2.18}$$

$$x_{ij} = 0 \text{ or } 1 \quad (i, j = 1), \ldots, n) \tag{2.19}$$

for any choice of the set S that breaks subtours which do not include the origin (node 1). In particular, any of the choices for S given earlier can be used. Note that (2.16) requires that *all* M salesman be used.

The apparent generalization that the MTSP provides is somewhat illusory, since any problem of this type can be converted into an equivalent TSP. Equivalent TSP formulations of the MTSP were derived independently by Bellmore and Hong[59], Orloff[524], Svestka and Huckfeldt[638], and Eilon *et al.*[201]. The equivalence is obtained by creating M copies of the depot denoted $D_1, D_2, \ldots, D_M$, each connected to the other nodes exactly as was the original depot (with the same lengths). The M copies are not connected, or are connected by arcs each with a length exceeding

$$\sum_{i=1}^{n} \sum_{j=1}^{n} |c_{ij}|.$$

In this way, an optimal single salesman tour in the expanded network (over node set $\{D_1, D_2, \ldots, D_M, 2, 3, \ldots, n\}$) will never use an arc connecting two copies of the depot. Then, by coalescing the copies back into a single node, the single salesman tour decomposes into M subtours as required in the MTSP.

For example, consider a four-node problem with two salesmen. Originally, we have nodes 1, 2, 3 and 4 with node 1 as the origin or central depot. Our expanded network is comprised of nodes D_1, D_2, 2, 3 and 4 where nodes D_1 and D_2 represent the two salesmen. In Fig. 2.14, we see that tour D_1–3–D_2–4–2–D_1 in the expanded network is equivalent in the two-salesmen interpretation to subtours 1–3–1 and 1–4–2–1.

This useful transformation therefore allows one to solve MTSP's by using any of the algorithms for the TSP described in Sections 2.2 and 2.3. We should remark, however, that Laporte and Nobert[414] report that it is more efficient to solve the MTSP directly using a constraint relaxation approach.

2.3.4 *Vehicle routing formulations*

The vehicle routing problem requires a set of delivery routes from a central depot to various demand points, each having service requirements, in order to minimize the total distance covered by the entire fleet. Vehicles have capacities and, possibly, maximum route time constraints. All vehicles start and finish at the central depot. We will refer to the following formulation for this problem, which is due to Golden *et al.* [306], as the generic vehicle routing problem. If the maximum route time constraints are omitted, we obtain the standard vehicle routing problem (VRP). The problem as stated is a pure delivery problem. In the case where only pick-ups are made instead, we have an equivalent problem. When both pick-ups and deliveries are present simultaneously, we face a more complex problem that we shall discuss later in Chapter 4.

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{v=1}^{NV} c_{ij} x_{ij}^{v} \tag{2.20}$$

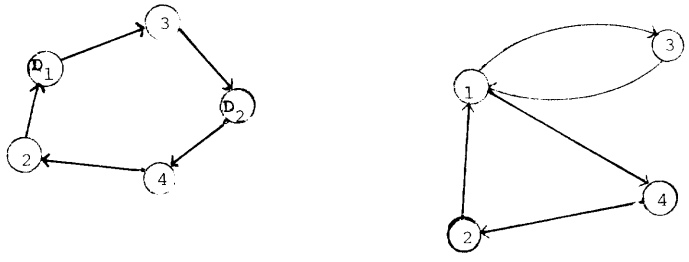


Fig. 2.14. A two-salesmen problem.

$$\text{subject to} \quad \sum_{i=1}^{n} \sum_{v=1}^{NV} x_{ij}^{v} = 1 \qquad (j = 2, \ldots, n) \tag{2.21}$$

$$\sum_{j=1}^{n} \sum_{v=1}^{NV} x_{ij}^{v} = 1 \qquad (i = 2, \ldots, n) \tag{2.22}$$

$$\sum_{i=1}^{n} x_{ip}^{v} - \sum_{j=1}^{n} x_{pj}^{v} = 0 \qquad \begin{matrix}(v = 1, \ldots, NV;\\ p = 1, \ldots, n)\end{matrix} \tag{2.23}$$

$$\sum_{i=1}^{n} d_i \left(\sum_{j=1}^{n} x_{ij}^{v} \right) \leq K_v \qquad (v = 1, \ldots, NV) \tag{2.24}$$

$$\sum_{i=1}^{n} t_i^{v} \sum_{j=1}^{n} x_{ij}^{v} + \sum_{i=1}^{n} \sum_{j=1}^{n} t_{ij}^{v} x_{ij}^{v} \leq T_v \qquad (v = 1, \ldots, NV) \tag{2.25}$$

$$\sum_{j=2}^{n} x_{1j}^{v} \leq 1 \qquad (v = 1, \ldots, NV) \tag{2.26}$$

$$\sum_{i=2}^{n} x_{i1}^{v} \leq 1 \qquad (v = 1, \ldots, NV) \tag{2.27}$$

$$X \in S \tag{2.28}$$

$$x_{ij}^{v} = 0 \text{ or } 1 \qquad \text{for all } i, j, v, \tag{2.29}$$

where n = number of nodes; NV = number of vehicles; K_v = capacity of vehicle v; Tv = maximum time allowed for a route of vehicle v; d_i = demand at node i ($d_1 = 0$); t_1^{v} = time required for vehicle v to deliver or collect at node i($t_1^{v} = 0$); t_{ij}^{v} = travel time for vehicle v from node i to node j($t_{ii}^{v} = \infty$); c_{ij} = cost of travel from node i to node j; $x_{ij}^{v} = 1$ if arc $i-j$ is traversed by vehicle v, 0 otherwise; X = matrix with components $x_{ij} \equiv \sum_{v=1}^{NV} x_{ij}^{v}$, specifying connections regardless of vehicle type.

Equation (2.20) states that total distance is to be minimized. Alternatively, we could minimize costs by replacing c_{ij} by a cost coefficient c_{ij}^{v} which depends upon the vehicle type. Equations (2.21) and (2.22) ensure that each demand node is served by exactly one vehicle. Route continuity is represented by equations (2.23), i.e. if a vehicle enters a demand node, it must exit from that node. Equations (2.24) are the vehicle capacity constraints; similarly, equations (2.25) are the total elapsed route time constraints. For instance, a newspaper delivery truck may be restricted from spending more than one hour on a tour in order that the maximum time interval from press to street be made as short as possible. Equations (2.26) and (2.27) guarantee that vehicle availability is not exceeded. Finally, the subtour-breaking constraints (2.28) can be, with slight modifications, any of the equations specified previously. Since (2.21) and (2.23) imply (2.22), and (2.23) and (2.26) imply (2.27), from now on we consider the model to include (2.20)–(2.29) excluding (2.22) and (2.27), which are redundant. We assume that $\max_{1 \leq i \leq n} d_i < \min_{1 \leq v \leq NV} K_v$. That is, the demand at each node does not exceed the capacity of any truck. In our model we have assumed that when a demand node is serviced, its requirements are satisfied completely, that is, partial servicing of demand is excluded.

Observe that when vehicle capacity constraints (2.24) and route time constraints (2.25) are nonbinding, and can be ignored, this model reduces to a MTSP as can be seen by eliminating constraints (2.23) and substituting $x_{ij} = \sum_{v=1}^{NV} x_{ij}^{v}$ in the objective function and remaining constraints.

Note that since $\sum_{j=2}^{n} x_{1j}^{v}$ is 1 or 0 depending upon whether or not the v^{th} vehicle is used, a fixed acquisition cost $f_v \sum_{j=2}^{n} x_{1j}^{v}$ can be added to the model when it is formulated with an objective function to investigate trade-offs between routing and vehicle acquisition (or fixed) costs.

We can, without difficulty, generalize the generic model to the multicommodity case where several different types of products must be routed simultaneously over a network in order to satisfy demands at delivery points for the various products.

Furthermore, we can incorporate timing restrictions (time windows) into the vehicle dispatching model. If we define a_j as the arrival time at node j, then restrictions on delivery deadlines $\bar{a}_j$ and earliest delivery times $\underline{a}_j$ can be represented by the following nonlinear equations:

$$a_j = \sum_v \sum_i \left(a_i + t_i^v + t_{ij}^v \right) x_{ij}^v \qquad (j = 1, \ldots, n)$$

$$a_i = 0$$

$$\underline{a}_j \le a_j \le \bar{a}_j. \qquad (j = 2, \ldots, n).$$

Alternatively, the above nonlinear constraints can be replaced by the linear constraints

$$\left.\begin{aligned} a_j &\ge (a_i + t_i^v + t_{ij}^v) - (1 - x_{ij}^v)\, T \\ a_j &\le (a_i + t_i^v + t_{ij}^v) + (1 - x_{ij}^v)\, T \end{aligned}\right\} \text{for all } i, j, v.$$

where $T = \max_{1 \le v \le \mathrm{NV}} T_v$. When $x_{ij}^v = 0$, these constraints are redundant. When $x_{ij}^v = 1$, they determine a_j in terms of the arrival time a_i at node i preceding node j on a tour, the delivery time t_i^v at node i, and the travel time t_{ij}^v between nodes i and j. Of course, these constraints add considerably to the size of the model (2.20)–(2.29). Their role is to introduce a vehicle scheduling component into the routing problem. Time windows are more fully discussed in Chapter 4.

2.4.4 *Classification of solution strategies in vehicle routing*

Most solution strategies for vehicle routing problems can be classified as one of the following approaches: (1) cluster first—route second; (2) route first—cluster second; (3) savings/insertion; (4) improvement/exchange; (5) mathematical-programming-based; (6) interactive optimization; (7) exact procedures. The first four approaches as well as the last one have been used extensively in the past. The other two approaches represent relatively recently developed ideas. A more general framework for heuristic algorithms is given by Ball and Magazine [39].

Cluster first-route second procedures group or cluster demand nodes and/or arcs first and then design economical routes over each cluster as a second step. Examples of this idea are given by Gillett and Miller [274], Gillett and Johnson [273], Chapleau *et al.* [120], and Karp [376] for the standard single depot vehicle routing problem.

Route first-cluster second procedures work in the reverse sequence. First, a large (usually infeasible) route or cycle is constructed which includes all of the demand entities (that is, nodes and/or arcs). Next, the large route is partitioned into a number of smaller, but feasible, routes. Golden *et al.* [296] provide an algorithm that typifies this approach for a heterogenous fleet size vehicle routing problem. Newton and Thomas [506] and Bodin and Berman [82] use this approach for routing school buses to and from a single school, and Bodin and Kursh [87, 88] utilize this approach for routing street sweepers. See also the work of Stern and Dror [623]. These applications are described in greater detail in Chapter 4.

Savings or *insertion* procedures build a solution in such a way that at each step of the procedure (up to and including the penultimate step) a current configuration that is possibly infeasible is compared with an alternative configuration that may also be infeasible. The alternative configuration is one that yields the largest savings in terms of some criterion function, such as total cost, or that inserts least expensively a demand entity not in the current configuration into the existing route or routes. The procedure eventually concludes with a feasible configuration. Examples of savings/insertion procedures for single depot node and arc routing problems are described by Clarke and Wright [145], Golden *et al.* [306], Norback and

Love[514], and Golden and Wong[310]. Hinson and Mulherkar[350] use a variation of this procedure for routing airplanes.

Improvement or *exchange* procedures such as the well-known branch exchange heuristic developed by Lin[444] and Lin and Kernighan[446] and extended by Christofides and Eilon[139] and Russell[572] always maintain feasibility and strive towards optimality. At each step, one feasible solution is altered to yield another feasible solution with a reduced overall cost. This procedure continues until no additional cost reductions are possible. Bodin and Sexton[94] modify this approach in order to schedule mini-buses for the subscriber dial-a-ride problem.

Mathematical programming approaches include algorithms that are directly based on a mathematical programming formulation of the underlying routing problem. An excellent example of mathematical-programming-based procedures is given by Fisher and Jaikumar[219]. They formulate the Dantzig-Ramser vehicle routing problem as a mathematical program in which two interrelated components are identified. One component is a traveling salesman (routing) problem and the other is a generalized assignment (packing) problem. Their heuristic attempts to take advantage of the fact that these two problems have been studied extensively and powerful mathematical programming approaches for their solution have already been devised. We discuss their heuristic in detail later. The algorithm due to Krolak and Nelson[404] is similar in spirit. Christofides *et al.*[143] and Stewart and Golden[628] discuss Lagrangian relaxation procedures for the routing of vehicles (see also Section 2.2). In addition, the article by Christofides, Mingozzi, and Toth[144]] represents a mathematical-programming-based (in particular, dynamic programming) approach for obtaining lower bounds in a variety of combinatorial optimization problems related to vehicles routing. Further discussion on this topic can be found in the article by Magnanti[455].

Interactive optimization is a general-purpose approach in which a high degree of human interaction is incorporated into the problem-solving process. The idea is that the experienced decision-maker should have the capability of setting the revising parameters and injecting subjective assessments based on knowledge and intuition into the optimization model. This almost always increases the likelihood that the model will eventually be implemented and used. Some early adaptations of this approach to vehicle routing problems are presented by Krolak, Felts and Marble[402] and Krolak, Felts and Nelson[403]. The paper by Cullen, Jarvis and Ratliff[156] introduces several rather novel interactive optimization heuristics.

Exact procedures for solving vehicle routing problems include specialized branch and bound, dynamic programming and cutting plane algorithms. Some of the more effective TSP approaches are described by Held and Karp[337, 338], Hansen and Krarup[327], Miliotis[480, 481], Balas and Christofides[29] and Crowder and Padberg[155]. Christofides *et al.*[143] discuss exact algorithms for the VRP. As noted before, any relaxation procedure that can provide good lower bounds on the optimal value of the vehicle routing problem can be embedded within a branch and bound approach to yield an exact procedure.

2.5. SELECTED SOLUTION TECHNIQUES FOR SINGLE DEPOT VRP's

As pointed out in the previous section, there are exact algorithms that solve the vehicle routing problem optimally by branch and bound techniques and there are heuristic algorithms that solve the problem approximately. Since the exact algorithms have been viable only for very small problems, we concentrate on heuristic algorithms. To our knowledge, the largest vehicle routing problem of any complexity that has been solved exactly involved about 25 customers[143]. We will now discuss several vehicle routing heuristic methods for a homogeneous fleet that have been used for problems up to 1000 customers.

The savings algorithm. The savings algorithm due to Clarke and Wright[111] is an "exchange" procedure in the sense that at each step one set of tours is exchanged for a better set. Initially, we suppose that every demand point is supplied individually by a separate vehicle (refer to Fig. 2.15 a).

Now, if instead of using two vehicles to service nodes i and j, we use only one, then we obtain a savings s_{ij} in travel distance of

$$(2c_{1i} + 2c_{1j}) - (c_{1i} + c_{1j} + c_{ij}) = c_{1i} + c_{1j} - c_{ij}$$

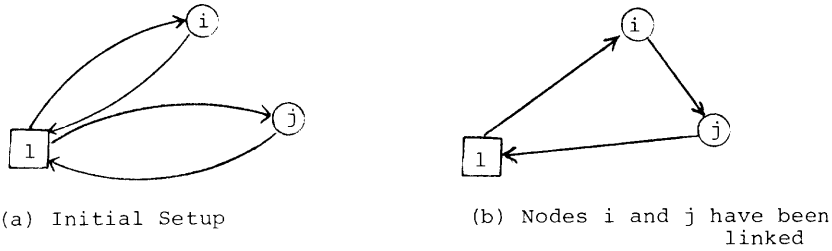


Fig. 2.15. Tour aggregation

(see Fig. 2.15 b). We note that if distances are asymmetric then $s_{ij} = c_{i1} + c_{1j} - c_{ij}$. For every possible pair of tour end points i and j there is a corresponding savings s_{ij}. We order these savings from largest to smallest and starting from the top of the list we link nodes i and j with maximum savings s_{ij} unless the problem constraints are violated.

The sweep approach. The sweep approach was originally devised by Gillett and Miller[274]. This approach, which is an efficient algorithm for problems of up to about 250 nodes, constructs a solution in two stages. First, it assigns nodes to vehicles and then it sequences the order in which each vehicle visits the nodes assigned to it. Rectangular coordinates for each demand point are required, from which we may calculate polar coordinates. We select a "seed" node randomly. With the central depot as the pivot, we start sweeping (clockwise or counterclockwise) the ray from the central depot to the seed. Demand nodes are added to a route as they are swept. If the polar coordinate indicating angle is ordered for the demand points from smallest to largest (with seed's angle 0) we enlarge routes as we increase the angle until capacity restricts us from enlarging a route by including an additional demand node (note that the procedure does not explicitly account for timing constraints); this demand point becomes the seed for the following route. Once we have the routes, we can apply TSP algorithms such as the 3-opt heuristic to improve tours. In addition, we can vary the seed and select the best solution. Figure 2.16 illustrates this procedure.

A penalty algorithm. Stewart and Golden[628] and Stewart[627] present a heuristic algorithm that treats the capacity constraints by moving them into the objective function and applying a multiplier in order to impose a penalty when demand on a route exceeds capacity. This explicit consideration of customer demand tends to provide better results.

We will assume that the vehicle fleet is homogeneous and that no timing restrictions are imposed. In this instance, the mathematical programming formulation of the general VRP given earlier becomes:

$$\text{Minimize} \quad \sum_{v} \sum_{i,j} c_{ij} x_{ij}^{v} \tag{2.30}$$

subject to

$$\sum_{i,j} d_i x_{ij}^{v} \leq K \quad \text{for all} \quad v = 1, \ldots, M \tag{2.31}$$

$$X = [x_{ij}^{v}] \in S^* \tag{2.32}$$

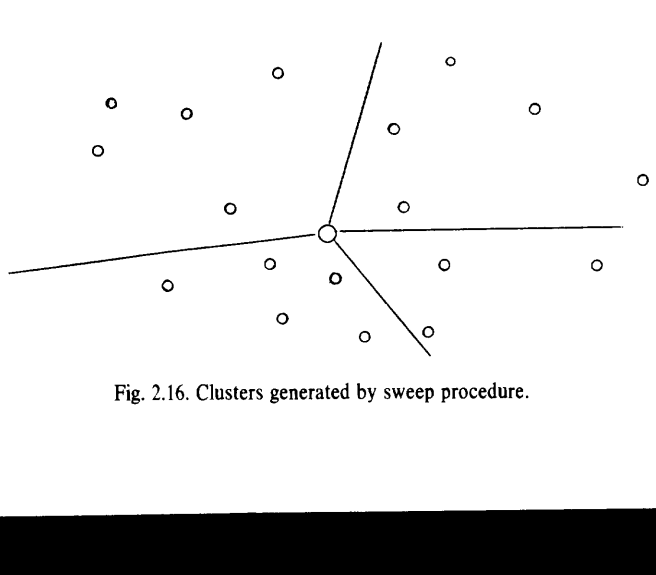


Fig. 2.16. Clusters generated by sweep procedure.

where c_{ij} = the distance or cost of traveling from i to j; $x_{ij}^v = 1$ if i and j are joined on route v, 0 otherwise; d_i = the demand at point i; K = the vehicle capacity; M = the number of vehicles; S^* = the set of all M-traveling salesman solutions. Without loss of generality, we can impose the additional redundant constraints

$$\sum_{i,j} d_i x_{ij}^k \geq \sum_{i,j} d_i x_{ij}^l \text{ for } l = k+1 \text{ and } k = 1, \ldots, M-1 \tag{2.33}$$

to the previous mathematical (integer) program. The effect is merely to assign the largest demand route—number 1, the second largest—number 2, and so on.

A Lagrangean problem associated with the VRP given in (2.30)–(2.33) is of the form

$$\text{Minimize} \quad \sum_v \sum_{i,j} c_{ij} x_{ij}^v + \lambda \sum_{i,j} d_i x_{ij}^l \tag{2.34}$$

subject to

$$\sum_{i,j} d_i x_{ij}^k \geq \sum_{i,j} d_i x_{ij}^l \text{ for } l = k+1 \text{ and } k = 1, \ldots, M\text{-}1 \tag{2.35}$$

$$X = [x_{ij}^v] \in S^* \tag{2.36}$$

where $\lambda \geq 0$ may be interpreted as a penalty for route failure.

We next present an algorithm that exploits this formulation. The idea is to start with a value of λ and solve problem (2.34)–(2.36) for $x(\lambda)$. Next, the algorithm provides a mechanism for varying λ in order to approach the optimal solution to the VRP. Each time λ is varied, problem (2.34)–(2.36) is solved for $x(\lambda)$ until "near optimality" is attained for the VRP. The procedure is heuristic due to the fact that (2.34)–(2.36) is solved approximately and not exactly. Moreover, even if (2.34)–(2.36) were solved exactly, the result might not be an optimal solution to the VRP since there may be a duality gap between the objective value for the "best" $x(\lambda)$ and the optimal solution to the VRP. The steps are described below:

Step 1. Set $\lambda = 0$. Solve the resulting MTSP. If the solution satisfies the capacity constraints, terminate. Otherwise, continue.

Step 2. Select $\lambda > 0$ and $\Delta > 0$.

Step 3. Solve the problem given in (2.34)–(2.36). This is an MTSP in which the largest demand route has penalty λ. If $x(\lambda)$ is feasible for the VRP, go to Step 5. Otherwise, continue.

Step 4. Route 1 is above vehicle capacity and has route demand y_1. Set $\lambda \leftarrow \lambda + \Delta$ and go to step 3.

Step 5. Search for the smallest value of λ, call it λ^*, between $\lambda - \Delta$ and λ such that $x(\lambda^*)$ is a feasible solution to the VRP. Then terminate with solution $x(\lambda^*)$.

The motivation underlying the penalty algorithm is that if λ is increased, the largest demand route is made smaller—thus driving infeasible solutions feasible. Conversely, if λ is decreased, the largest demand route becomes cheaper and is loaded more fully, thus, taking better advantage of vehicle capacity. This obvious relationship between λ and feasibility indicates why the redundant constraints (2.33) were introduced in the first place.

Some specific comments concerning the nature of the algorithm still need to be made. Step 1 can be implemented in a number of ways. One possibility is to solve the VRP using the savings procedure and then to combine small demand routes so as to form exactly M routes. These need not all be feasible. An alternative is to convert the MTSP into a TSP and apply one of the numerous heuristics for the TSP discussed previously. In step 3, a modified 3-opt procedure is used to solve the MTSP in which the largest demand route is penalized. The modification entails recalculating total demand on each route at each step, and associating λ with the largest demand route. In other words, the problem given in (2.34)–(2.36) is solved approximately for each different value of λ. Figure 2.17 describes the nature of step 3. The solid lines in Fig. 2.17 (a) and (b) define a feasible solution to a 3-salesman problem of the form (2.34)–(2.36). In (b) the three nodes D_1, D_2 and D_3 corresponding to the salesmen in (a) are coalesced. In (a) the dashed

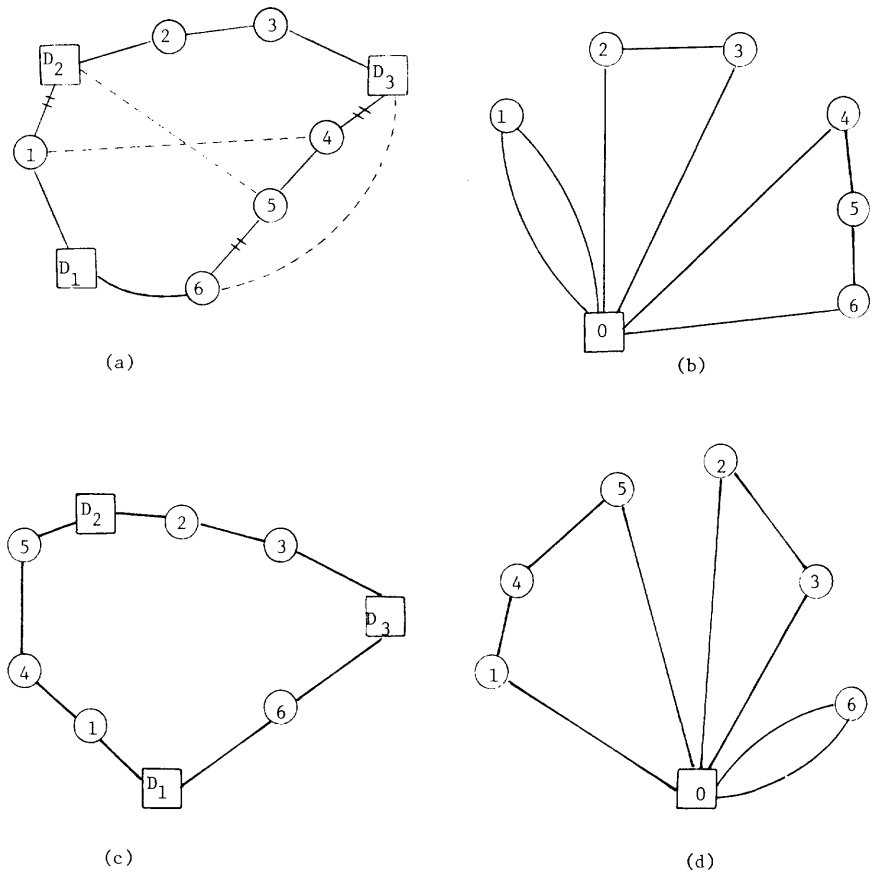


Fig. 2.17. Step 3—the swap operation

lines indicate a potential 3-swap. If the cost including the largest route penalty of λ times total demand associated with the new configuration (as shown in (c) and (d)) is less than the cost of the configuration in (a) and (b), the swap is performed. In step 3, we continue until it is no longer possible to reduce the cost (as measured by objective function (2.34)) via 3-swaps.

Step 4 is intended to help drive towards feasibility and in step 5 we strive for optimality. Finally, we point out that if different starting solutions are used in step 1, a number of different VRP solutions may be obtained, from which the best could then be selected. In particular, the savings algorithm can be reapplied t times using capacities $Q+\epsilon$, $Q+2\epsilon$, . . . , $Q+t\epsilon$.

M-Tour approach. This approach, due to Russell[572], is very similar to step 3 of the Penalty algorithm. The essential difference is that the M-tour algorithm requires a feasible solution to the VRP (with M vehicles) as input. This M-tour solution is expressed as a traveling salesman tour on an expanded network as in (a) and (b) of Fig. 2.17. Then, a modified 3-opt procedure or any other improvement scheme is used to reduce total cost. Before performing a profitable swap, however, a check for feasibility must be carried out. In that way, feasibility is preserved and total cost is improved at each step of the modified 3-opt procedure.

A generalized assignment heuristic. Now, we discuss a heuristic which views the VRP as consisting of two interrelated components. One component is a traveling salesman (routing) problem and the other is a generalized assignment (packing) problem. This heuristic was developed by Fisher and Jaikumar[219].

Before we discuss the mechanics of the heuristics algorithm, we formulate the VRP in such a way that the two components become self-evident. The formulation is given below.

$$\text{Minimize} \quad \sum_{i,j,v} c_{ij} x_{ij}^{v} \tag{2.37}$$

subject to

$$\sum_i d_i y_{iv} \le K_v, \qquad v = 1, \ldots, \mathrm{NV} \tag{2.38}$$

$$\sum_v y_{iv} = \begin{cases} \mathrm{NV}, & i = 1 \\ 1, & i = 2, \ldots, n \end{cases} \tag{2.39}$$

$$y_{iv} = 0 \text{ or } 1, \quad \begin{matrix} i = 1, \ldots, n \\ v = 1, \ldots, \mathrm{NV} \end{matrix} \tag{2.40}$$

$$\left.\begin{aligned} \sum_i x_{ij}^v &= y_{jv}, \quad j = 1, \ldots, n && (2.41)\\ \sum_j x_{ij}^v &= y_{iv}, \quad i = 1, \ldots, n && (2.42)\\ \sum_{i,j \in L} x_{ij}^v &\le |L| - 1, \quad \begin{matrix} L \subseteq \{2, \ldots, n\} \\ 2 \le |L| \le n-2 \end{matrix} && (2.43)\\ x_{ij}^v &= 0 \text{ or } 1, \quad \begin{matrix} i = 1, \ldots, n \\ j = 1, \ldots, n \end{matrix} && (2.44) \end{aligned}\right\} v = 1, \ldots, \mathrm{NV}$$

where

$$y_{iv} = \begin{cases} 1 \text{ if customer } i \text{ is serviced by vehicle } v \\ 0 \text{ otherwise} \end{cases}$$

Constraints (2.38)–(2.40), which define a generalized assignment problem, guarantee that each customer i is assigned to a vehicle, that no vehicle load $\sum_i d_i y_{iv}$ exceeds its capacity and that each vehicle route contains the central depot (node 1). Whenever the variables y_{iv} are fixed to satisfy (2.38)–(2.40), the constraints (2.41)–(2.44) for each vehicle v define a TSP over the customers assigned to that vehicle.

An advantage of this formulation is that since the TSP and the generalized assignment problem have been studied extensively in the literature, this formulation of the VRP can benefit from the use of theory and algorithms already devised for the two component problems. In addition, the heuristic that we are to describe based upon this formulation will always find a feasible solution if one exists.

Now consider the following reformulation of the VRP as a nonlinear generalized assignment problem.

$$\text{Minimize} \quad \sum_v f(y_v) \tag{2.45}$$

subject to

$$\sum_i d_i y_{iv} \le K_v, \qquad v = 1, \ldots, \mathrm{NV} \tag{2.46}$$

$$\sum_v y_{iv} = \begin{cases} \mathrm{NV}, & i = 1 \\ 1, & 1 = 2, \ldots, n \end{cases} \tag{2.47}$$

$$y_{iv} = 0 \text{ or } 1, \quad \begin{matrix} i = 1, \ldots, n \\ v = 1, \ldots, \mathrm{NV} \end{matrix} \tag{2.48}$$

where $f(y_v)$ is the cost of an optimal TSP tour over all customers in $N(y_v) = \{i | y_{iv} = 1\}$. Since $f(y_v) = \text{Minimum } \{\sum_{i,j} c_{ij} x_{ij}^v | \text{ constraints (2.41)–(2.44) hold for fixed } v\}$, it is clear that $f(y_v)$ is

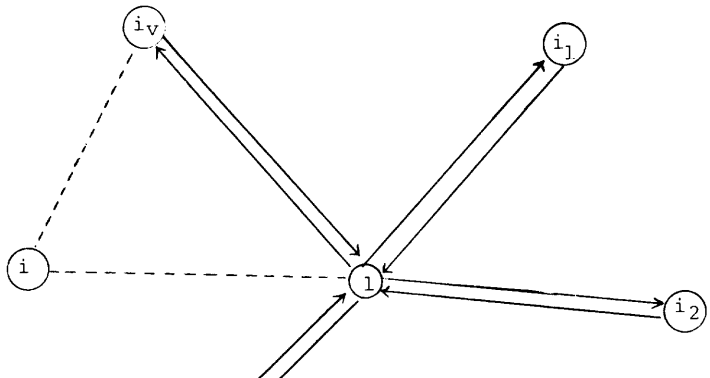


Fig. 2.18. Computation of $\tilde{c}_{iv}$.

indeed a complicated function. The generalized assignment heuristic replaces $f(y_v)$ in (2.45) with a linear approximation $\sum_i \tilde{c}_{iv} Y_{iv}$ and then solves (2.45)–(2.48).

The solution to the resulting linear generalized assignment problem defines a feasible assignment of customers to vehicles. Then, for each vehicle, we need to solve a TSP using either exact or approximate techniques. What remains to be shown is how one constructs an effective linear approximation of $f(y_v)$. We present one straightforward approach; there are a number of others that we do not pursue here.

Suppose we judiciously choose a set of "seed" customers $i_1, \ldots, i_{\mathrm{NV}}$ to be assigned to vehicles $1, \ldots, \mathrm{NV}$, respectively. We can set $\tilde{c}_{iv}$ equal to the cost incurred upon inserting customer i into route 1–i_v–1, as illustrated in Fig. 2.18. That is,

$$\tilde{c}_{iv} = \min\{c_{1,i} + c_{i,i_v} + c_{i_v,1},\ c_{1,i_v} + c_{i_v,i} + c_{i,1}\} - \{c_{1,i_v} + c_{i_v,1}\}.$$

Note that we are not assuming symmetry here. The seed customer may be varied and the procedure repeated. Also, the selection of seeds may be automated. Despite the simplicity of this linear approximation, the computational results have been extremely impressive.

2.6. MULTI-DEPOT VEHICLE ROUTING

Formulations. Minor alterations in the integer programming formulation of the vehicle routing problem given in (2.20)–(2.29) permit the modeling of multiple depot problems. Letting nodes $1, 2, \ldots, M$ denote the depots, we obtain the formulation by changing the index in constraints (2.21)–(2.22) to $(j = M+1, \ldots, n)$ and $(i = M+1, \ldots, n)$, respectively, by changing constraints (2.26) and (2.27) to

$$\sum_{i=1}^{M} \sum_{j=M+1}^{n} x_{ij}^{v} \le 1 \qquad (v = 1, \ldots, \mathrm{NV}) \tag{2.26'}$$

$$\sum_{p=1}^{M} \sum_{i=M+1}^{n} x_{ip}^{v} \le 1 \qquad (v = 1, \ldots, \mathrm{NV}), \tag{2.27'}$$

and be redefining our previous choices for the subtour breaking set S to be:

(1′) $S = \{(x_{ij})$: $\sum_{i \in Q} \sum_{J \notin Q} x_{ij} \ge 1$ for every proper subset Q of $\{1, 2, \ldots, n\}$ containing nodes $1, 2, \ldots, M\}$;

(2′) $S = \{(x_{ij})$: $\sum_{i \in R} \sum_{j \in R} x_{ij} \le |R| - 1$ for every nonempty subset R of $\{M+1, M+2, \ldots, n\}\}$;

(3′) $S = \{(x_{ij})\colon\ y_i - y_j + nx_{ij} \le n-1 \quad \text{for } M+1 \le i \ne j \le n$
for some real numbers $y_i\}$.

Heuristic solution techniques for multi-depot VRP's. Whereas the single depot VRP has been studied widely, the multi-depot problem has attracted less attention. The relevant literature is represented by only a few papers. However, a number of relatively successful heuristic approaches, which we will discuss in this section, have been developed for the multi-depot problem.

The assignment— sweep approach. This method, proposed by Gillett and Johnson[273], is an extension of the sweep heuristic that solves the multi-depot problem in two stages. First, locations are assigned to depots. Then, several single depot VRP's are solved using the sweep heuristic. Each stage is treated separately.

Initially, all locations are in an unassigned state.

For any node i, let

$t'(i)$ be the closest depot to i, and $t''(i)$ be the second closest depot to i.

For each node i, the ratio

$$r(i) = c_{i,t'(i)}/c_{i,t''(i)}$$

is computed and all nodes are ranked by increasing values of $r(i)$. The arrangement determines the order in which the nodes are assigned to a depot: those that are relatively close to a depot are considered first. After a certain number of nodes have been assigned from the list of $r(i)$, a small cluster is formed around every depot.

Locations i such that the ratio $r(i)$ is close to 1, are assigned as follows. If two adjacent nodes j and k are already assigned to a depot, inserting i between j and k on a route linked to t creates an additional cost equal to $c_{ji} + c_{ik} - c_{jk}$ which represents a part of the total cost (or distance). In other words, the algorithm assigns location i to a depot t by inserting i between two nodes already assigned to depot t, in the least costly manner.

The sweep algorithm is used to construct and sequence routes in the cluster about each depot independently. A number of refinements are then made to improve the resulting solution.

A multi-depot savings approach. This procedure, due to Tillman and Cain[648] starts with the initial solution consisting of servicing each node exclusively by one route from the closest depot. Let c_{ij} = cost or distance between demand nodes i and j; and c_i^k = cost or distance between node i and depot k. The total initial cost of all routes is then $D = \sum_{i=1}^{N} 2 \min\{c_i^k\}$ where N is the number of demand points. The method successively links pairs of nodes in order to decrease the total cost. One basic rule is assumed in the algorithm: the initial assignment of nodes to the nearest terminal is temporary, but once two or more nodes have been assigned to a common route from a terminal, the nodes are not reassigned to another terminal. In addition, as in the original savings algorithm, nodes i and j can be linked only if neither i nor j are interior to an existing tour.

At each step, the choice of linking a pair of nodes i and j on a route from terminal k is made in terms of the savings when linking i and j at k. Nodes i and j can be linked only if no constraints are violated.

The computation of savings is not as straightforward as in the case of a single depot. The savings s_{ij}^k are associated with every combination of demand nodes i and j and depot k, and represent the decrease in total cost or distance traveled when linking i and j at k. The formula is given by

$$s_{ij}^k = \tilde{c}_i^k + \tilde{c}_j^k - c_{ij}$$

where

$$\tilde{c}_i^k = \begin{cases} 2 \min_t\{c_i^t\} - c_i^k & \text{if } i \text{ has not yet been given a permanent assignment} \\ c_i^k & \text{otherwise.} \end{cases}$$

In the first case, node i is being reassigned from its closest depot and the previously assigned cost $2 \min_t \{c_i^t\}$ is saved; in the second case, node j is being inserted between depot k and node i and the link from i to k is dropped from the current solution.

The savings s_{ij}^k are computed for $i, j = 1, \ldots, N (i \neq j)$ and $k = 1, \ldots, M$ at each step. They can be stored in M matrices, each N by N. At each step, the link $i-j$ at k is chosen that maximizes s_{ij}^k and that yields a feasible solution (with regard to capacity and other restrictions).

A multi-depot savings algorithm for large problems. The multi-depot VRP can be viewed as being solved in two stages: first, nodes must be allocated to depots; then routes must be built that link nodes assigned to the same depot. Ideally, it is more efficient to deal with the two steps simultaneously as in the multi-depot savings algorithm just discussed. When faced with larger problems, with say 1000 nodes, however, this method is no longer tractable computationally. A reasonable approach would be to divide the problem into as many subproblems as there are depots and to solve each subproblem separately.

The algorithm presented in this section attempts to implement this strategy while using the ratios $r(i)$ introduced in the discussion of the assignment-sweep algorithm. The approach is due to Golden *et al.*[306]. If a given node i is much closer to one depot than any other depot, i will be served from its closest depot. When node i is equidistant from several depots, the assignment of i to a depot becomes more difficult. For every node i, we determine the closest depot k_1 and the second closest depot k_2. If the ratios $r(i) = c_i^{k_1}/c_i^{k_2}$ is less than a certain chosen parameter $(0 \leq \delta \leq 1)$, we assign i to k_1; in the case $r(i) \geq \delta$ we say that node i is a border node.

Clearly, if $\delta = 0$, all nodes are declared border nodes and if $\delta = 1$, all nodes are assigned to their closest depot. For a given problem, we can fix the number of border nodes as we wish, by varying δ.

The method proceeds as follows. In the first step, we ignore the nonborder nodes and the multi-depot savings algorithm is applied to the set of border nodes. The algorithm allocates the border nodes to depots and simultaneously builds segments of routes connecting these nodes. At the end of this first step, all nodes of our problem are assigned to some depot and all border nodes are linked on some routes. The solution to the VRP is produced depot by depot using single depot VRP techniques. The segments of routes that are built on border nodes are extended to the remaining nodes.

2.7. THE FLEET SIZE AND MIX PROBLEM

The goal of this section is to consider a situation in which the distributor decides, for the purposes of convenience and reliability, to *lease*, rather than *acquire*, vehicles for a certain period of time. The number of vehicles of each type needed to handle the system demand effectively must be determined together with a coherent routing policy for such vehicles. We assume that each vehicle type has a fixed leasing cost and a variable routing cost that is proportional to the distance traveled. The variable cost component encompasses the costs of fuel, maintenance, and manpower. This problem is formulated as a mathematical program by Golden *et al.*[296]. See Ball *et al.*[38] for a related study.

Golden *et al.*[296] also develop heuristics for solving this complex routing problem. We discuss some of these here. The most obvious approach for solving the fleet size and mix problem is to adapt the Clarke and Wright savings technique, discussed in Section 2.5. Remember that two distinct subtours containing customers i and j can be joined if: (a) i and j are not in the same subtour; (b) neither i nor j is an interior point in a subtour; and (c) the combined subtour containing both i and j has a total demand not in excess of vehicle capacity. The savings due to such a combination is $s_{ij} = c_{0i} + c_{0j} - c_{ij}$ and is computed for all $1 \leq i, j \leq n$ $(i \neq j)$. The Clarke and Wright procedure makes combinations in the order indicated by the savings (from largest to smallest), until no further feasible combinations exist.

This procedure is deficient for the fleet size and mix problem due to the fact that it ignores fixed vehicle costs. Since an infinite number of vehicles is assumed to be available, feasibility is certainly no problem. As a result, the savings method will tend to combine subtours until the capacity of the largest vehicle is reached. Unfortunately, a fleet composed almost entirely of largest vehicles may not be the least expensive. We denote this naive application of the Clarke and Wright approach to the fleet size and mix problem by CW.

It should now be clear that we need to extend the concept of savings to include fixed vehicle costs. Every subtour has a total cost equal to the sum of its routing costs and the cost of the vehicle servicing its customers. When two subtours combine to form a larger subtour, the total cost of this larger subtour can be calculated in a similar fashion. To determine which two subtours should combine at any point in the procedure, we need to find the subtours which generate the greatest total savings when combined. These savings must be calculated anew at every step, but all the other attractive features of the CW approach remain. This method, the Combined Savings (CS) approach, is the basis for the remaining savings algorithms that we consider in this section.

In order to give a precise representation of this algorithm, some additional notation is necessary. Let F be a real-valued function such that if z is the demand of a subtour, $F(z)$ is the fixed cost of the smallest vehicle that can service that demand. Clearly, F is a step function. Now consider a subtour I which has node i as one of its endpoints and a subtour J which has customer j as one of its endpoints. If the total demands on these two subtours are z_i and z_j, then the combined savings of joining customers i and j is

$$\bar{s}_{ij} = s_{ij} + F(z_i) + F(z_j) - F(z_i + z_j).$$

The algorithm terminates when $\bar{s}_{ij}$ is negative for all eligible i, j pairs.

The CS approach is a logical extension of the CW approach to the fleet size and mix problem. There is, however, and element of imprecision to the algorithm since $\bar{s}_{ij}$ represents only the *immediate* savings gained by joining customers i and j. Consider the following example. Suppose that subtours I, J and K each have total demand of 100 and are each serviced by a vehicle with a capacity of 100. Suppose further that the next largest vehicle has a capacity of 300 and is therefore considerably more expensive than the 100 capacity. It might be that having one 300 capacity vehicle service the combination of the I, J and K subtours is cheaper than having these routes individually serviced by three 100 capacity vehicles. But will this combined route form under the CS approach? Not necessarily. Since subtours can only combine two at a time, it is quite possible that the combined savings for, say, customers i and j (endpoints of subtours I and J) will be negative, due to the large fixed cost of the 300 capacity vehicle. In this situation, the CS approach ignores the fact that the 300 capacity vehicle would have an unused capacity of 100 units which could be used to absorb additional subtours later on.

These considerations motivated Golden *et al.*[296] to develop variants of the CS approach, which may be termed *opportunity* savings algorithms. Basically, these algorithms consider total savings to include savings in routing costs, savings in fixed vehicle costs, and opportunity savings. The first two savings are determined in the same manner as in the CS algorithm. Opportunity savings is a function of the unused capacity of the vehicle servicing the combined subtours.

The first opportunity savings algorithm is the Optimistic Opportunity Savings (OOS). In this algorithm, an opportunity savings is somewhat arbitrarily defined to be the cost of the smallest vehicle that can service the entire unused capacity of the new vehicle. For example, if the available vehicles have capacities of 50, 100 and 200 units, the total savings incurred by combining two subtours with total demands of 40 and 80 units into one with a total demand of 120 units would be $\bar{s}_{ij}$ plus the opportunity savings (the cost of a 100 capacity vehicle—the smallest vehicle that can service the unused capacity of 80 units). The OOS algorithm optimistically assumes that a subtour S with a total demand equal to the unused capacity on subtour R always exists and that at some future time subtour S will combine with subtour R and be absorbed by the vehicle servicing subtour R. The opportunity savings represents the fixed vehicle cost savings that would occur if this future combination materialized. The savings in routing cost from this future combination cannot be estimated as easily and is therefore ignored. Clearly, the OOS algorithm encourages subtours to combine more readily than the CS algorithm.

We now provide a formal description of the OOS algorithm. Let $P(z)$ be the capacity of the smallest vehicle that can service a subtour with a total demand of z. Thus, in the previous example, $P(40) = 50$, $P(80) = 100$, and $P(120) = 200$. The OOS algorithm is identical to the CS

approach, except that the savings upon linking customers i and j becomes

$$s_{ij}^{*} = s_{ij} + F(z_i) + F(z_j) - F(z_i + z_j) + F(P(z_i + z_j) - z_i - z_j).$$

Is the optimism of the OOS algorithm warranted? It would seem not. Overall, the CS approach outperforms the OOS algorithm. Actually, both methods are flawed. Essentially, the CS approach under-combines and the OOS algorithm over-combines subtours.

A remedial algorithm that comes to mind is the method of Realistic Opportunity Savings (ROS). This method is motivated by two basic assumptions. *First*, the sole purpose of opportunity savings should be to encourage the use of larger vehicles when it seems profitable to do so. Thus, opportunity savings should not be included in the savings formula unless combining two subtours forces the use of a larger vehicle. If the vehicle required to service the combined subtours is larger than each of the vehicles presently used, we say that a vehicle threshold has been crossed. Opportunity savings should be included only if a vehicle threshold is crossed. *Second*, based on empirical observations, the total demand of a subtour tends to be close to a vehicle capacity. Therefore, instead of assuming that the vehicle which might be absorbed in some future combination is the smallest vehicle that can service the unused capacity, it is perhaps more logical to use the largest vehicle that can be squeezed into the unused capacity. Note that this is the vehicle just smaller than the OOS vehicle, except when the unused capacity equals a vehicle capacity, in which case both algorithms use the same vehicle.

A full description of the ROS algorithm follows. Let $F'(z)$ be a function analogous to $F(z)$, except that $F'(z)$ represents the fixed cost of the largest vehicle whose capacity is less than or equal to z. If $z < a_1$, $F'(z) = 0$. Let I and J denote two subtours with total demands of z_i and z_j and let i and j be endpoints of subtours I and J. Assume, without loss of generality, that $z_i \geq z_j$. If $F(z_i + z_j) = F(z_i)$, then $\bar{s}_{ij}$ is the total savings from linking customers i and j as in the CS approach. If, however, $F(z_i + z_j) > F(z_i)$, then a vehicle threshold has been crossed and opportunity savings need to be considered. In this case, the total savings would be

$$s'_{ij} = \bar{s}_{ij} + F'(P(z_i + z_j) - z_i - z_j).$$

After total savings has been computed, the ROS algorithm proceeds in the same way as the CS approach.

Table 2.3. Summary of the savings algorithms

Algorithm	Savings	Savings Formula
CW	s_{ij}	$c_{0i} + c_{0j} - c_{ij}$
CS	$\bar{s}_{ij}$	$s_{ij} + F(z_i) + F(z_j) - F(z_i + z_j)$
OOS	s^{*}_{ij}	$\bar{s}_{ij} + F(P(z_i + z_j) - z_i - z_j)$
ROS	s'_{ij}	$\bar{s}_{ij} + \delta(w)\, F'(P(z_i + z_j) - z_i - z_j)$
ROS-γ	s''_{ij}	$s'_{ij} + (1-\gamma)\, c_{ij}$

where $F(z)$ = the fixed cost of the smallest vehicle that can service a demand of z

$P(z)$ = the capacity of the smallest vehicle that can service a demand of z

$F'(z)$ = the fixed cost of the largest vehicle that has a capacity less than or equal to z

$w = P(z_i + z_j) - P(\max\{z_i, z_j\})$

$$\delta(w) = \begin{cases} 0 & \text{if } w = 0 \\ 1 & \text{if } w > 0 \end{cases}$$

The computational results presented by Golden *et al.* [296] supported the claim that the ROS algorithm is superior to the OOS and CS algorithms. In seven of twelve problems tested, the ROS method performed at least as well as the other two algorithms. Its average and worst case behavior on the twelve problems was also superior to all other savings methods tested.

Still, none of the above methods are methods are powerful enough that single application will consistently generate good solutions. It is therefore necessary to vary these algorithms so that they produce a number of different solutions for each problem, with the least costly of these being selected as a final solution. A method used which introduces variety into the ROS algorithm is ROS-γ.

The ROS-γ algorithm uses a route shape parameter that changes the Clarke-Wright savings expression to $s_{ij} = c_{0i} + c_{0j} - \gamma \cdot c_{ij}$. In tests, Golden *et al.*[296] varied γ from 0.0 to 3.0, by tenths, for a total of 31 trials.

The logic behind the ROS-γ approach is fairly straightforward. A major flaw of the savings algorithms is that once a node is assigned to a tour, it cannot be reassigned to another at a later point. Two linkings may seem almost equivalent at an early stage of the algorithm and yet the choice between these can have enormous impact upon the final solution. Varying γ allows the approach to incorporate the variety which is essential to finding a good solution. The Realistic Opportunity Savings algorithm with route shape parameter seems to be the best of the savings algorithms that Golden *et al.* [296] have developed and tested. Additional heuristics and other aspects of the fleet size and mix problem are discussed by Golden *et al.*[296].

The savings algorithms described in this section are conveniently summarized in Table 2.3.

2.8. STOCHASTIC VEHICLE ROUTING

The vehicle routing problem (VRP), already discussed in detail, has been studied by many authors in recent years. This section treats a version of the stochastic vehicle routing problem (SVRP) which is a generalization of the deterministic VRP. The following modifications to the VRP are required:

(1) Customer demand is a random variable with a known probability distribution.

(2) Routes must be designed before the actual demands become known.

(3) The objective is to minimize expected travel distance, but other costs might be incurred if some customers cannot be serviced on a particular trip.

The SVRP is at least as difficult as the VRP. The presence of nonlinearities in the constraints and/or the objective function caused by the probabilistic demands may, in fact, make it considerably more difficult. For this reason, only heuristic solution methods have been considered for this problem.

Relatively little research has been conducted on the SVRP. Exceptions include the articles by Tillman[646], Stewart[625], Golden and Stewart[307], Golden and Yee[311], Yee and Golden[697], Stewart and Golden[631] and Cook and Russell[149]. Stewart's thesis[627] is the best source of material on this topic. Also see the recent work of Federgruen and Zipkin[208].

Some research that we describe has concentrated on transforming the SVRP to a VRP and then applying one of the existing VRP algorithms to this transformed problem. In what follows, artificial capacity will refer to a capacity to be used in a VRP algorithm in place of the true vehicle capacity. This artificial capacity will be less than the true capacity. Routing will be performed by viewing mean customer demands as artificial demands and filling vehicles up to their artificial capacity.

A mathematical formulation of the SVRP is now presented. When the demand at each customer location is a random variable, the problem becomes a stochastic vehicle routing problem and must be treated accordingly. In the formulation below, the demands d_i are random variables from a known probability distribution. The parameter α is the maximum allowable probability that any route fails.

Stochastic Vehicle Routing Problem (SVRP):

$$\text{Minimize} \sum_k \sum_{i,j} c_{ij} x_{ijk} \tag{2.49}$$

$$\text{Subject to Prob}\left\{\sum_{i,j} d_i x_{ijk} \le Q\right\} \ge 1-\alpha \quad \text{for } k=1,\ldots,m \tag{2.50}$$

$$x=[x_{ijk}] \in S \tag{2.51}$$

where c_{ij} = the distance or cost of traveling from i to j; Q = the vehicle capacity; m = the number of vehicles; S = the set of all m-traveling salesman solutions; $x_{ijk}=1$ if i and j are joined on route k, 0 otherwise. Constraint (2.50) is referred to as a chance-constraint.

The chance-constrained model is useful when a penalty for violation of the capacity constraint cannot be explicitly defined. In this case, some subjective estimate of customer service in terms of the probability of route failure may be all that is obtainable.

Solution procedures. The solution procedures discussed in this section were designed so that existing heuristic algorithms for the VRP could be used to solve the SVRP with only slight modification. This is an important consideration since the existing VRP algorithms have been so well-studied in the literature. Other heuristics have been studied and are discussed in Stewart's thesis [627].

In the arguments that follow, we demonstrate that a chance constraint can be transformed into an equivalent deterministic constraint. In many cases, a closed-form expression for the artificial capacity can be derived from this equivalent constraint.

Consider a constraint of the form

$$\text{Prob}\left\{\sum_i a_i y_i \le b\right\} \ge 1-\alpha. \tag{2.52}$$

If the a_i are independent and identically distributed random variables with mean $\bar{a}_i$ and standard deviation s_i and the y_i are decision variables, then the mean of $\sum_i a_i y_i$ is $M=\sum_i \bar{a}_i y_i$ and the standard deviation is given by $S=(\sum_i s_i^2 y_i^2)^{1/2}$. We now seek the constant T such that

$$\text{Prob}\left\{\left(\sum_i a_i y_i - M\right)/S \le T\right\} = 1-\alpha. \tag{2.53}$$

If $b \ge M+TS$, then the l.h.s. of inequality (2.52) is also not less than $1-\alpha$. Hence, constraint (2.52) can be replaced by the deterministic constraint

$$M+TS \le b. \tag{2.54}$$

This constraint is generally nonlinear in nature as seen below when we rewrite the constraint in terms of y_i:

$$\sum_i \bar{a}_i y_i + T\left(\sum_i s_i^2 y_i^2\right)^{1/2} \le b \tag{2.55}$$

Under special conditions, however, this constraint can assume a linear form. We note that in the context of the SVRP the a_i's are probabilistic demands and the y_i's are 0–1 variables. Of course, b is the vehicle capacity. If the probability distributions are such that $S^2=\lambda M$ for some constant λ, which is indeed the case for the gamma, binomial, Poisson, and negative binomial distributions discussed by Golden and Yee [311], then (2.55) can be rewritten as

$$\sum_i \bar{a}_i y_i + T\left(\lambda \sum_i \bar{a}_i y_i\right)^{1/2} \le b \tag{2.56}$$

or

$$M+T\sqrt{(\lambda M)} \le b. \tag{2.57}$$

If this inequality is manipulated properly, it results in a simple expression of the form

$$M \le \bar{b}. \tag{2.58}$$

Define $w = T\sqrt{\lambda}$ and note that

$$b - M \geq wM^{1/2}$$

or

$$(b - M)^2 \geq w^2 M$$

which yields

$$M \leq \frac{2b + w^2 - \sqrt{(w^4 + 4bw^2)}}{2} = \bar{b}. \tag{2.59}$$

If T is determined by Prob $(z \leq T) = 1 - \alpha$ where z is a unit normal variate, then this result is identical to the general expression derived by Golden and Yee [311].

Unfortunately, closed-form expressions for $\bar{b}$ (the artificial capacity) like eqn (2.59) do not exist for very many probability distributions. With this is mind, a more general algorithm is often required. Several such algorithms including a penalty function procedure are developed by Stewart [627] and Stewart and Golden [629].

2.9. THE CHINESE POSTMAN PROBLEM

Introduction. The problem of finding an optimal route for a single vehicle over a network is, as we have seen, a very difficult combinatorial problem. Routing problems can be classified as node routing problems, arc routing problems, or general routing problems. Bodin [79] and Bodin and Golden [85] provide a convenient taxonomy for these problems. The problem of visiting all nodes in a network in the minimal amount of time (node routing) is the classical Traveling Salesman Problem (TSP). The problem of covering all arcs of a network while minimizing the total distance traveled (arc routing) is the Chinese Postman Problem (CPP). The General Routing Problem (GRP) on network $G = G(N; A)$ (N is the set of all nodes, A the set of all arcs) is a generalization which includes the TSP and the CPP as special cases. Here we seek the minimum cost cycle which visits every node in subset $Q \subseteq N$ and covers every arc in subset $R \subseteq A$. Orloff introduces the GRP and later extends it to handle more realistic problem situations [523–525]. In this section, we concentrate primarily on the Chinese Postman Problem and arc routing problems in general. Applications include routing of street sweepers, snow plows, household refuse collection vehicles, postmen, the spraying of roads with salt-grit to prevent ice formation, the inspection of electric power lines, gas or oil pipelines for faults, etc. Since in many vehicle routing situations customers are so numerous that identifying them individually becomes cumbersome, we can consider the CPP to be the continuous counterpart to the discrete TSP. Here an arc replaces a number of customers.

Several variations of the CPP which have been studied are: the undirected postman problem (where all arcs are undirected), the directed postman problem (where all arcs are directed), and the mixed postman problem (where some arcs are directed and others are not). Historically, the CPP has been thought of as an undirected postman problem and in this chapter "CPP" will refer to the undirected case. Stricker [635], Christofides [128], Golden and Wong [310], and Golden *et al.* [303] consider an extension which we will call "the Capacitated Chinese Postman Problem" (CCPP), which reflects real-life situations more directly. Here, we are given arc demands $Q = [q_{ij}]$ which must be satisfied by vehicles of capacity W. A number of cycles must be formed which traverse every arc, satisfy demands, and yield minimum distance. This is an arc routing version of the VRP.

The undirected Chinese postman problem. The solution of the undirected Chinese postman problem relies on the notion of Euler tours. An Euler tour is a continuous path (tour) through a network that traverses each arc exactly once. Conditions for the existence of Euler tours in directed and undirected graphs were first given by Euler in 1736 in connection with the celebrated Konigsberg Bridge problem. Euler proved that such a tour exists in an undirected graph if and only if the degree (number of incident arcs) of every node is even. If a graph

contains odd nodes (nodes of odd degree) we must cover some arcs more than once in order to form a cycle covering all arcs.

Mei-Ko[477], in 1962, addressed the problem of minimizing the distance required to cover all arcs when odd nodes do exist (see also [476]). Because this work appeared in *Chinese Mathematics* the problem of finding the best postman tour on an undirected network has become known as the Chinese Postman Problem.

We present below the mathematical programming formulation for the CPP which has been given by Stricker[635]. Since Edmonds' matching theory results (discussed later) provide us with a graph-theoretic algorithm for solving this problem, the integer programming formulation is not solved by classical IP techniques. In fact, for most applications, these programs are too large for solution by anything but special purpose algorithms. However, the formulation is valuable in that it provides a framework for many similar problems, such as the CCPP, which are more complex.

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=i}^{n} c_{ij} x_{ij} \tag{2.49}$$

$$\text{subject to} \quad \sum_{k=1}^{n} x_{ki} - \sum_{k=1}^{n} x_{ik} = 0 \quad \text{for } i = 1, \ldots, n \tag{2.50}$$

$$x_{ij} + x_{ji} \geq 1 \quad \text{for all arcs } (i, j) \in A \tag{2.51}$$

$$x_{ij} \geq 0 \text{ and integer} \tag{2.52}$$

where n = the number of nodes in the network; A = the set of all arcs in the network; x_{ij} = the number of times arc (i, j) is traversed from i to j; c_{ij} = the length of arc (i, j).

We first observe that every graph must have an even number of odd nodes. If d_i is the degree of node i and m is the number of arcs, then $\sum_i d_i = \sum_{i \text{even}} d_i + \sum_{i \text{odd}} d_i = 2m$ because each arc adds unity to the degrees of its two end nodes. Since $\sum_{i \text{ even}} d_i$ is even, $\sum_{i \text{ odd}} d_i$ must also be even. But each $d_i (i \text{ odd})$ is odd so the number of odd nodes must be even.

Let n_0 be the set of odd nodes, N_e the set of even nodes, and N the set of all nodes. Since the number of odd nodes $|N_0|$ is even, we can partition these nodes into two groups and form $k = (1/2)|N_o|$ paths between disjoint pairs of nodes. The arcs $\bar{A}$ contained in these paths are added to the original graph G as artificial arcs to obtain the augmented graph $G_0(\bar{A})$. The augmented graph will be a multigraph (more than one arc can join two nodes) since the artificial arcs are duplicates of arcs which were in G to begin with.

A feasible solution to the CPP therefore stems from k paths which connect the k pairs of odd nodes. The problem then reduces to finding the best k paths.

Glover[278] and Murty[499] presented algorithms to solve the CPP in 1967; these were evaluated by Stricker[635]. Bellman and Cooke[56] have developed a dynamic programming algorithm which performs well only when there are fewer than about 15 arcs. The matching application to the CPP pointed out by Edmonds and Johnson[197] and Christofides[128], however, yields a good (polynomial) algorithm. We next discuss this computationally efficient approach. Christofides[128] proves the following theorem which formalizes the notion that we need only find the best set of arcs to traverse more than once.

THEOREM 2.1 (Christofides) For any cycle covering G, there is some choice of $\bar{A}$ for which $G_0(\bar{A})$ possess an Euler tour corresponding to the cycle of G. The correspondence is such that if a cycle traverses an arc (i, j) of G γ times, there are γ arcs (one real and γ-1 artificial) between i and j in $G_0(\bar{A})$ each of which is traversed exactly once by the Euler tour of $G_0(\bar{A})$, and conversely.

The CPP algorithm follows directly now, assuming we can solve a matching problem. All that is necessary is to find the set of arc $\bar{A}$ in paths between nodes of odd degree which produces the least additional cost. It should be noted that γ in Theorem 2.1 is never greater than 2. In other words, the optimal cycle never traverses any arc of G more than twice since

no artificial arc can be included in more than one path between odd nodes. If it were, we could do better by dropping that arc from both paths. The steps of the algorithm are displayed below.

CPP Algorithm

Step 1. Using a shortest path algorithm on the graph G with cost matrix $[c_{ij}]$ form the $|N_o|$ by $|N_o|$ matrix $D = [d_{ij}]$ where d_{ij} is the cost of the least-cost path from node $i \in N_o$ to another node $j \in N_o$.

Step 2. Use a polynomial-time minimum 1-matching algorithm to find $\bar{A}_{\min}$ according to the cost matrix D.

Step 3. If node α is matched to another node β, identify the least-cost path $\mu_{\alpha\beta}$ corresponding to the cost $d_{\alpha\beta}$ of step 1. Insert artificial arcs to obtain $G_0(\bar{A}_{\min})$.

Step 4. The sum of the costs from matrix $[c_{ij}]$ of all arcs in $G_0(\bar{A}_{\min})$ is the minimum cost of a cycle covering G.

A 1-matching on the complete graph formed from the odd nodes is a set of arcs E which cover all the odd nodes in such a way that no two arcs in E are adjacent (share a node). That is, each odd node is of degree 1 relative to the set E. A minimum weighted 1-matching is a 1-matching of minimum total length. Edmonds has, almost singly-handedly, developed the elegant combinatorial theory behind matching. In fact, Edmonds and Johnson[196, 197] demonstrate that the CPP can also be solved directly on G using a generalized matching algorithm; this saves the computation of all shortest paths between odd nodes. The basic references on matching theory results are Edmonds[193–195], and Edmonds and Johnson [196, 197]. Edmonds' matching algorithm requires on the order of n^4 computations in the worst case; Gabow[239] and Lawler[425] later improved the matching algorithm to perform on the order of n^3 computations in the worst case. Recent improvements in computational complexity and empirical running times are due to Derigs and Kazakides[177] and to Christofides[140].

Once the graph $G_0(\bar{A}_{\min})$ has been determined, an Euler tour on G_o corresponding to the minimum length cycle covering G can be found by starting with any node and traversing and then erasing an incident arc such that the removal of the arc does not divide the graph into two disjoint components. Subsequent nodes are treated similarly one at a time.

The directed Chinese postman problem. For directed connected networks, a sufficient condition for the existence of an Euler tour is that for each node the number of directed arcs into the node is the same as the number out of the node.

Beltrami and Bodin[63] provide an optimal algorithm for solving the postman problem over a directed network and a heuristic algorithm for solving the directed capacited capacitated Chinese Postman Problem. The second algorithm involves an application of the first algorithm. Then, the resulting giant tour is broken into a set of smaller tours which obey the capacity restrictions. It is fundamentally very similar to the approach suggested by Stricker[635].

In order to solve the postman problem on a directed network find all nodes n_j and m_k where the number of arcs into n_j exceeds the number out by s_j and the number of arcs out of m_k exceeds the number entering by d_k. It can easily be shown that $\sum_j s_j = \sum_k d_k$.

We can construct a bipartite graph between nodes n_j and m_k. Each distance c_{jk} is the shortest path distance from n_j to m_k and may be found by applying a shortest path algorithm to the original graph G. If there is a node n_j, such that no path exists to any node m_k then the directed postman problem does not have a feasible solution. In order for an Euler tour to exist each node n_j must have additional arcs directed outward and each node m_k must have additional inward arcs. The s_j's can be viewed as supplies and the d_k's as demands. The following transportation problem naturally arises.

$$\text{Minimize} \quad \sum_{j,k} c_{jk} x_{jk}$$

subject to

$$\sum_k x_{jk} = s_j \quad \text{for all } j$$

$$\sum_j x_{jk} = d_k \quad \text{for all } k$$

$$x_{jk} \geq 0.$$

Now if we augment the original graph G with the arcs implied by the solution of the transportation problem we have an Euler tour which is the solution to the directed postman problem.

2.10. CAPACITATED ARC ROUTING PROBLEMS

Uncapacitated routing problems can be classified as node problems, arc routing problems, or general routing problems (see Chapter 1). We have studied the TSP and CPP in depth and mentioned the GRP. In each case we have assumed that all arcs are undirected and that there is a vehicle of unlimited capacity. We point out that the terms *cost* and *distance* are used interchangeably in this section.

Capacitated variations, of course, reflect real-life situations more directly. For example, the vehicle routing problem has been the focus of much research attention. On the other hand, capacitated arc routing problems, also extremely practical, have been comparatively neglected (the recent work by Stern and Dror[623], Golden and Wong[310], and Golden *et al.*[303] are exceptions).

The capacitated arc routing problem (CARP) that we focus on is as follows: given an undirected network $G(N, E, C)$ with arc demands $q_{ij} \geq 0$ for each arc (i, j) which must be satisfied by one of a fleet of vehicles each of capacity W, find cycles each of which passes through the domicile (node 1) which satisfy demands as minimal total cost. Several related problems are posed below.

(1) The Chinese postman problem (CPP).

(2) The rural postman problem (RPP). Find a minimum-cost cycle which traverses each arc in a given subset $R \subseteq E$ at least once. See Orloff[523, 525] for more discussion and Lenstra and Rinnooy Kan[432] for an NP-completeness proof.

(3) The capacitated Chinese postman problem (CCPP).

(4) The traveling salesman problem (TSP).

(5) The vehicle routing problem (VRP).

(6) The general routing problem (GRP).

Now we examine the relationship between the CARP and these other problems. If we assume $q_{ij} > 0$ for all $(i, j) \in E$, the CARP reduces to the CCPP. If, in addition, we have only one vehicle of capacity $W \geq \sum_i \sum_j q_{ij}$, we obtain the CPP. With one vehicle of capacity $W \geq \sum_i \sum_j q_{ij}$ and $q_{ij} > 0$ for all arcs $(i, j) \in R \subseteq E$, we have the RPP. Thus, the CCPP, the CPP, and the RPP are special cases of the CARP. Since we can incorporate the constraint that a node must be serviced by splitting that node into two nodes joined by a zero-length arc with a demand equal to the original node demand, the TSP, VRP, and GRP are also special cases of the CARP.

Golden and Wong[310] show that even an approximate, restricted version of the CARP is NP-hard. They also show how to obtain, in polynomial time, a valid lower bound for the CARP by considering a particular matching problem derived from the original Carp.

A general purpose heuristic for the CARP has been suggested by Golden and Wong[310] and modified and tested by Golden *et al.*[303]. Its steps are outlined below.

Step 1. INITIALIZE—All demand arcs are serviced by a separate cycle.

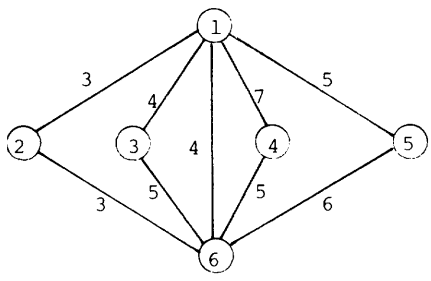


Fig. 2.19. Illustration of algorithm.

Step 2. AUGMENT—Starting with greatest length cycle available, see if a demand arc on a smaller cycle can be serviced on the larger cycle.

Step 3. MERGE—Subject to capacity constraints, evaluate the merging of any two cycles (possibly subject to additional restrictions). Merge the two cycles which yield the largest positive savings.

Step 4. ITERATE—Repeat Step 3 until finished.

Note that Steps 3 and 4 are closely related to the well-known Clarke–Wright vehicle routing algorithm. The details of the algorithm (in particular, the MERGE step) are explained by Golden *et al.*[303].

We demonstrate the algorithm on the small example given in Fig. 2.19. Here, vehicle capacity is 4 and each arc has unit demand. Initially, there are nine cycles—one for each demand arc. The augmentation step leaves the four cycles 1261, 1361, 1641, 1651. Step 3 merges 1261 and 1651 to obtain 12651 reducing the number of cycles to three and we are finished. Note that merging 1261 and 1361 to obtain 12631 or 1361 and 1651 to obtain 13651 would have given the same total distance.

Much additional information regarding the CARP and the CCPP is contained in the papers by Christofides[128], Golden and Wong[310], and Golden *et al.*[303].