to the given knapsack instance with added constraints $x_i = 1$ if $i \in I1$ and $x_i = 0$ if $i \in I2$. The bound $lbb(I1, I2)$ is a lower bound under the constraints of $I1$ and $I2$. Algorithm `Reduce` needs no further explanation. It should be clear that $I1$ and $I2$ are such that from an optimal solution to (8.2) we can easily obtain an optimal solution to the original knapsack problem.

The time complexity of `Reduce` is $O(n^2)$. Because the reduction procedure is very much like the heuristics used in `DKnap1` and the knapsack algorithms of this chapter, the use of `Reduce` does not decrease the overall computing time by as much as may be expected by the reduction in number of objects. These algorithms do dynamically what `Reduce` does. The exercises explore the value of `Reduce` further.

EXERCISES

1. Work out Example 8.2 using the variable tuple size formulation.
2. Work out Example 8.3 using the variable tuple size formulation.
3. Draw the portion of the state space tree generated by LCBB for the following knapsack instances:
   (a) $n = 5$, $(p_1, p_2, \ldots, p_5) = (10, 15, 6, 8, 4)$, $(w_1, w_2, \ldots, w_5) = (4, 6, 3, 4, 2)$, and $m = 12$.
   (b) $n = 5$, $(p_1, p_2, p_3, p_4, p_5) = (w_1, w_2, w_3, w_4, w_5) = (4, 4, 5, 8, 9)$ and $m = 15$.
4. Do problem 3 using LCBB on a dynamic state space tree (see Section 7.6). Use the fixed tuple size formulation.
5. Write a LCBB algorithm for the knapsack problem using the ideas given in Example 8.2.
6. Write a LCBB algorithm for the knapsack problem using the fixed tuple size formulation and the dynamic state space tree of Section 7.6.
7. [Programming Project] Program in C++ the algorithms `DKnap` (Program 5.7), `DKnap1` (see page 398), LCBB for knapsack, and `Bknap` (Program 7.12). Compare these programs empirically using randomly generated data as below:
   (a) Random w_i and p_i, $w_i \in [1, 100]$, $p_i \in [1, 100]$, and $m = \sum_1^n w_i/2$.
   (b) Random w_i and p_i, $w_i \in [1, 100]$, $p_i \in [1, 100]$, and $m = 2 \max\{w_i\}$.
   (c) Random w_i, $w_i \in [1, 100]$, $p_i = w_i + 10$, and $m = \sum_1^n w_i/2$.
   (d) Same as (c) except $m = 2 \max\{w_i\}$.

   (e) Random p_i, $p_i \in [1, 100]$, $w_i = p_i + 10$, and $m = \sum_1^n w_i/2$.
   (f) Same as (e) except $m = 2 \max\{w_i\}$.

   Obtain computing times for $n = 5, 10, 20, 30, 40, \ldots$. For each n generate (say) ten problem instances from each of the above data sets. Report average and worst-case computing times for each of the above data sets. From these times can you say anything about the expected behavior of these algorithms?

   Now, generate problem instances with $p_i = w_i$, $1 \le i \le n$, $m = \sum w_i/2$, and $\sum w_i x_i \neq m$ for any 0, 1 assignment to the x_i's. Obtain computing times for your four programs for $n = 10$, 20, and 30. Now study the effect of changing the range to [1, 1000] in data sets (a) through (f). In sets (c) to (f) replace $p_i = w_i + 10$ by $p_i = w_i + 100$ and $w_i = p_i + 10$ by $w_i = p_i + 100$.
8. [Programming Project]
   (a) Program the reduction heuristic `Reduce` of Section 8.2. Generate several problem instances from the data sets of Exercise 7 and determine the size of the reduced problem instances. Use $n =$ 100, 200, 500, and 1000.
   (b) Program `DKnap` and the backtracking algorithm `Bknap` for the knapsack problem. Compare the effectiveness of `Reduce` by running several problem instances (as in Exercise 7). Obtain average and worst-case computing times for `DKnap` and `Bknap` for the generated problem instances and also for the reduced instances. To the times for the reduced problem instances, add the time required by `Reduce`. What conclusion can you draw from your experiments?

8.3 TRAVELING SALESPERSON (*)

An $O(n^2 2^n)$ dynamic programming algorithm for the traveling salesperson problem was arrived at in Section 5.9. We now investigate branch-and-bound algorithms for this problem. While the worst-case complexity of these algorithms will not be any better than $O(n^2 2^n)$, the use of good bounding functions will enable these branch-and-bound algorithms to solve some problem instances in much less time than required by the dynamic programming algorithm.

Let $G = (V, E)$ be a directed graph defining an instance of the traveling salesperson problem. Let c_{ij} be the cost of edge $\langle i, j \rangle$, $c_{ij} = \infty$ if $\langle i, j \rangle \notin E$, and let $|V| = n$. Without loss of generality, we can assume that every tour starts and ends at vertex 1. So, the solution space S is given by $S = \{1, \pi, 1 | \pi$ is a permutation of $(2, 3, \ldots, n)\}$. $|S| = (n-1)!$. The size of S can be

reduced by restricting S so that $(1, i_1, i_2, \ldots, i_{n-1}, 1) \in S$ iff $\langle i_j, i_{j+1} \rangle \in E$, $0 \leq j \leq n-1$, and $i_0 = i_n = 1$. S can be organized into a state space tree similar to that for the n-queens problem (see Figure 7.2). Figure 8.10 shows the tree organization for the case of a complete graph with $|V| = 4$. Each leaf node L is a solution node and represents the tour defined by the path from the root to L. Node 14 represents the tour $i_0 = 1, i_1 = 3, i_2 = 4, i_3 = 2$ and $i_4 = 1$.

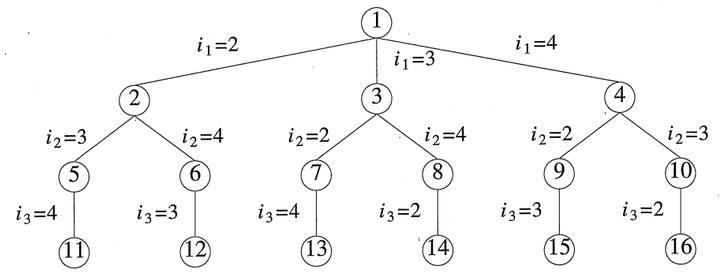


Figure 8.10 State space tree for the traveling salesperson problem with $n = 4$ and $i_0 = i_4 = 1$

To use LCBB to search the traveling salesperson state space tree, we need to define a cost function $c(\cdot)$ and two other functions $\hat{c}(\cdot)$ and $u(\cdot)$ such that $\hat{c}(r) \leq c(r) \leq u(r)$ for all nodes r. The cost $c(\cdot)$ is such that the solution node with least $c(\cdot)$ corresponds to a shortest tour in G. One choice for $c(\cdot)$ is

$$c(A) = \begin{cases} \text{length of tour defined by the path from the root to } A, & \text{if } A \text{ is a leaf.} \\ \text{cost of a minimum-cost leaf in the subtree } A, & \text{if } A \text{ is not a leaf.} \end{cases}$$

A simple $\hat{c}(\cdot)$ such that $\hat{c}(A) \leq c(A)$ for all A is obtained by defining $\hat{c}(A)$ to be the length of the path defined at node A. For example, the path defined at node 6 of Figure 8.10 is $i_0, i_1, i_2 = 1, 2, 4$. It consists of the edges $\langle 1, 2 \rangle$ and $\langle 2, 4 \rangle$. A better $\hat{c}(\cdot)$ can be obtained by using the reduced cost matrix corresponding to G. A row (column) is said to be *reduced* iff it contains at least one zero and all remaining entries are non-negative. A matrix is *reduced* iff every row and column is reduced. As an example of how to reduce the cost matrix of a given graph G, consider the matrix of Figure 8.11(a). This corresponds to a graph with five vertices. Since every tour on this graph includes exactly one edge $\langle i, j \rangle$ with $i = k$, $1 \leq k \leq 5$, and exactly one edge $\langle i, j \rangle$ with $j = k$, $1 \leq k \leq 5$, subtracting a constant t from every entry in

one column or one row of the cost matrix reduces the length of every tour by exactly t. A minimum-cost tour remains a minimum-cost tour following this subtraction operation. If t is chosen to be the minimum entry in row i (column j), then subtracting it from all entries in row i (column j) introduces a zero into row i (column j). Repeating this as often as needed, the cost matrix can be reduced. The total amount subtracted from the columns and rows is a lower bound on the length of a minimum-cost tour and can be used as the $\hat{c}$ value for the root of the state space tree. Subtracting 10, 2, 2, 3, 4, 1, and 3 from rows 1, 2, 3, 4, and 5 and columns 1 and 3 respectively of the matrix of Figure 8.11(a) yields the reduced matrix of Figure 8.11(b). The total amount subtracted is 25. Hence, all tours in the original graph have a length at least 25.

With every node in the traveling salesperson state space tree we can associate a reduced cost matrix. Let A be the reduced cost matrix for node R. Let S be a child of R such that the tree edge (R, S) corresponds to including edge $\langle i, j \rangle$ in the tour. If S is not a leaf, then the reduced cost matrix for S may be obtained as follows: (1) Change all entries in row i and column j of A to ∞. This prevents the use of any more edges leaving vertex i or entering vertex j. (2) Set $A(j, 1)$ to ∞. This prevents the use of edge $\langle j, 1 \rangle$. (3) Reduce all rows and columns in the resulting matrix except for rows and columns containing only ∞. Let the resulting matrix be B. Steps (1) and (2) are valid as no tour in the subtree s can contain edges of the type $\langle i, k \rangle$ or $\langle k, j \rangle$ or $\langle j, 1 \rangle$ (except for edge $\langle i, j \rangle$). If r is the total amount subtracted in step (3) then $\hat{c}(S) = \hat{c}(R) + A(i, j) + r$. For leaf nodes, $\hat{c}(\cdot) = c()$ is easily computed as each leaf defines a unique tour. For the upper bound function u, we can use $u(R) = \infty$ for all nodes R.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

(a) Cost matrix

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

(b) Reduced cost matrix
L = 25

Figure 8.11 An example

Let us now trace the progress of the LCBB algorithm on the problem instance of Figure 8.11(a). We use $\hat{c}$ and u as above. The initial reduced matrix is that of Figure 8.11(b) and *upper* $= \infty$. The portion of the state space tree that gets generated is shown in Figure 8.12. Starting with the root node as the E-node, nodes 2, 3, 4, and 5 are generated (in that order). The reduced matrices corresponding to these nodes are shown in Figure 8.13.

The matrix of Figure 8.13(b) is obtained from that of 8.11(b) by (1) setting all entries in row 1 and column 3 to ∞, (2) setting the element at position (3, 1) to ∞, and (3) reducing column 1 by subtracting by 11. The $\hat{c}$ for node 3 is therefore 25 + 17 (the cost of edge $\langle 1,3\rangle$ in the reduced matrix) + 11 = 53. The matrices and $\hat{c}$ value for nodes 2, 4, and 5 are obtained similarly. The value of *upper* is unchanged and node 4 becomes the next E-node. Its children 6, 7, and 8 are generated. The live nodes at this time are nodes 2, 3, 5, 6, 7, and 8. Node 6 has least $\hat{c}$ value and becomes the next E-node. Nodes 9 and 10 are generated. Node 10 is the next E-node. The solution node, node 11, is generated. The tour length for this node is $\hat{c}(11) = 28$ and *upper* is updated to 28. For the next E-node, node 5, $\hat{c}(5) = 31 > upper$. Hence, LCBB terminates with 1, 4, 2, 5, 3, 1 as the shortest length tour.

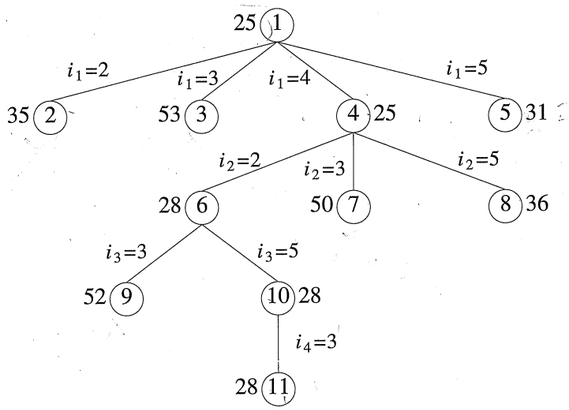


Figure 8.12 State space tree generated by procedure LCBB

An exercise examines the implementation considerations for the algorithm described above. A different LCBB algorithm can be arrived at by considering a different tree organization for the solution space. This organization is reached by regarding a tour as a collection of n edges. If $G = (V, E)$ has e edges, then every tour contains exactly n of the e edges. However, for each $i, 1 \leq i \leq n$, there is exactly one edge of the form $\langle i, j\rangle$ and one of the form $\langle k, i\rangle$ in every tour. A possible organization for the state space is a binary tree in which a left branch represents the inclusion of a particular edge while

the right branch represents the exclusion of that edge. Figure 8.14(b) and (c) represents the first two levels of two possible state space trees for the three vertex graph of Figure 8.14(a). As is true of all problems, many state space trees are possible for a given problem formulation. Different trees differ in the order in which decisions are made. Thus, in Figure 8.14(c) we first decide the fate of edge $\langle 1, 2\rangle$. Rather than use a static state space tree, we shall now consider a dynamic state space tree (see Section 7.1). This is also a binary tree. However, the order in which edges are considered depends on the particular problem instance being solved. We compute $\hat{c}$ in the same way as we did using the earlier state space tree formulation.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

(a) Path 1,2; node 2

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

(b) Path 1,3; node 3

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

(c) Path 1,4; node 4

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

(d) Path 1,5; node 5

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

(e) Path 1,4,2; node 6

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$$

(f) Path 1,4,3; node 7

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$$

(g) Path 1,4,5; node 8

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

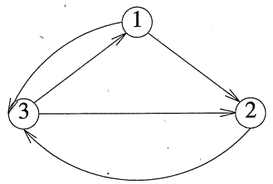
(h) Path 1,4,2,3; node 9

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

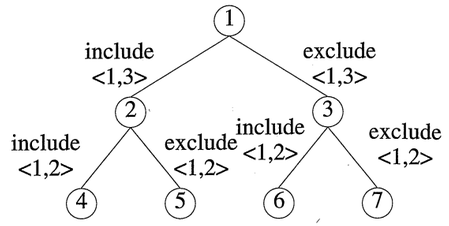
(i) Path 1,4,2,5; node 10

Figure 8.13 Reduced cost matrices corresponding to nodes in Figure 8.12

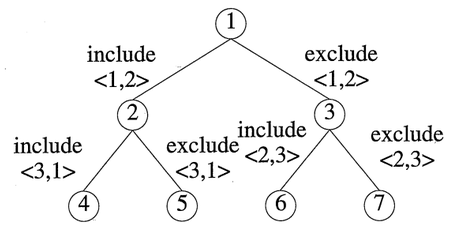
As an example of how LCBB would work on the dynamic binary tree formulation, consider the cost matrix of Figure 8.11(a). Since a total of 25 needs to be subtracted form the rows and columns of this matrix to obtain the reduced matrix of Figure 8.11(b), all tours have a length at least 25. This fact is represented by the root of the state space tree of Figure 8.15. Now, we must decide which edge to use to partition the solution space into two subsets. If edge $\langle i, j\rangle$ is used, then the left subtree of the root represents all tours including edge $\langle i, j\rangle$ and the right subtree represents all tours that do not include edge $\langle i, j\rangle$. If an optimal tour is included in the left subtree,



(a) Graph



(b) Part of a state space tree



(c) Part of a state space tree

Figure 8.14 An example

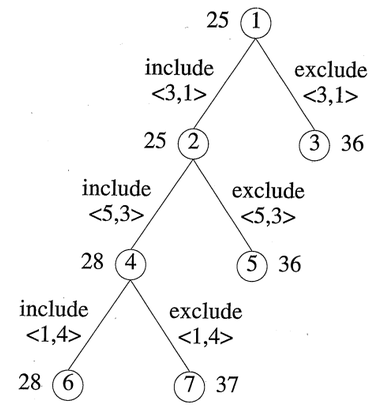


Figure 8.15 State space tree for Figure 8.11(a)

then only $n-1$ edges remain to be selected. If all optimal tours lie in the right subtree, then we have still to select n edges. Since the left subtree selects fewer edges, it should be easier to find an optimal solution in it than to find one in the right subtree. Consequently, we would like to choose as the partitioning edge an edge $\langle i, j\rangle$ that has the highest probability of being in an optimal tour. Several heuristics for determining such an edge can be formulated. A selection rule that is commonly used is select that edge which results in a right subtree that has highest $\hat{c}$ value. The logic behind this is that we soon have right subtrees (perhaps at lower levels) for which the $\hat{c}$ value is higher than the length of an optimal tour. Another possibility is to choose an edge such that the difference in the $\hat{c}$ values for the left and right subtrees is maximum. Other selection rules are also possible.

When LCBB is used with the first of the two selection rules stated above and the cost matrix of Figure 8.11(a), the tree of Figure 8.15 is generated. At the root node, we have to determine an edge $\langle i, j\rangle$ that will maximize the $\hat{c}$ value of the right subtree. If we select an edge $\langle i, j\rangle$ whose cost in the reduced matrix (Figure 8.11(b)) is positive, then the $\hat{c}$ value of the right subtree will remain 25. This is so as the reduced matrix for the right subtree will have $B(i, j) = \infty$ and all other entries will be identical to those in Figure 8.11(b). Hence B will be reduced and $\hat{c}$ cannot increase. So, we must choose an edge with reduced cost 0. If we choose $\langle 1, 4\rangle$, then $B(1, 4) = \infty$ and we need to subtract 1 from row 1 to obtain a reduced matrix. In this

$$\begin{bmatrix} \infty & 10 & \infty & 0 & 1 \\ \infty & \infty & 11 & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 3 & 12 & \infty & 0 \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

(a) Node 2

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 1 & \infty & 11 & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & 12 & \infty & 0 \\ 0 & 0 & 0 & 12 & \infty \end{bmatrix}$$

(b) Node 3

$$\begin{bmatrix} \infty & 7 & \infty & 0 & \infty \\ \infty & \infty & \infty & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

(c) Node 4

$$\begin{bmatrix} \infty & 10 & \infty & 0 & 1 \\ \infty & \infty & 0 & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 3 & 1 & \infty & 0 \\ \infty & 0 & \infty & 12 & \infty \end{bmatrix}$$

(d) Node 5

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

(e) Node 6

$$\begin{bmatrix} \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

(f) Node 7

Figure 8.16 Reduced cost matrices for Figure 8.15

case $\hat{c}$ will be 26. If $\langle 3,1\rangle$ is selected, then 11 needs to be subtracted from column 1 to obtain the reduced matrix for the right subtree. So, $\hat{c}$ will be 36. If A is the reduced cost matrix for node R, then the selection of edge $\langle i,j\rangle$ $(A(i,j) = 0)$ as the next partitioning edge will increase the $\hat{c}$ of the right subtree by $\triangle = \min_{k\neq j}\{A(i,k)\} + \min_{k\neq i}\{A(k,j)\}$ as this much needs to be subtracted from row i and column j to introduce a zero into both. For edges $\langle 1,4\rangle, \langle 2,5\rangle, \langle 3,1\rangle$ $\langle 3,4\rangle, \langle 4,5\rangle, \langle 5,2\rangle$, and $\langle 5,3\rangle, \triangle = 1$, 2, 11, 0, 3, 3, and 11 respectively. So, either of the edges $\langle 3,1\rangle$ or $\langle 5,3\rangle$ can be used. Let us assume that LCBB selects edge $\langle 3,1\rangle$. The $\hat{c}(2)$ (Figure 8.15) can be computed in a manner similar to that for the state space tree of Figure 8.12. In the corresponding reduced cost matrix all entries in row 3 and column 1 will be ∞. Moreover the entry (1, 3) will also be ∞ as inclusion of this edge will result in a cycle. The reduced matrices corresponding to nodes 2 and 3 are given in Figure 8.16(a) and (b). The $\hat{c}$ values for nodes 2 and 3 (as well as for all other nodes) appear outside the respective nodes.

Node 2 is the next E-node. For edges $\langle 1,4\rangle, \langle 2,5\rangle, \langle 4,5\rangle, \langle 5,2\rangle$, and $\langle 5,3\rangle$, $\triangle = 3, 2, 3, 3$, and 11 respectively. Edge $\langle 5,3\rangle$ is selected and nodes 4 and 5 generated. The corresponding reduced matrices are given in Figure 8.16(c) and (d). Then $\hat{c}(4)$ becomes 28 as we need to subtract 3 from column 2 to reduce this column. Note that entry (1, 5) has been set to ∞ in Figure 8.16(c). This is necessary as the inclusion of edge $\langle 1,5\rangle$ to the collection $\{\langle 3,1\rangle, \langle 5,3\rangle\}$ will result in a cycle. In addition, entries in column 3 and row 5 are set to ∞. Node 4 is the next E-node. The $\triangle$ values corresponding to edges $\langle 1,4\rangle, \langle 2,5\rangle$, and $\langle 4,2\rangle$ are 9, 2, and 0 respectively. Edge $\langle 1,4\rangle$ is selected and nodes 6 and 7 generated. The edge selection at node 6 is $\{\langle 3,1\rangle, \langle 5,3\rangle, \langle 1,4\rangle\}$. This corresponds to the path 5, 3, 1, 4. So, entry (4,

5) is set to ∞ in Figure 8.16(e). In general if edge $\langle i,j\rangle$ is selected, then the entries in row i and column j are set to ∞ in the left subtree. In addition, one more entry needs to be set to ∞. This is an entry whose inclusion in the set of edges would create a cycle (Exercise 4 examines how to determine this). The next E-node is node 6. At this time three of the five edges have already been selected. The remaining two may be selected directly. The only possibility is $\{\langle 4,2\rangle, \langle 2,5\rangle\}$. This gives the path 5, 3, 1, 4, 2, 5 with length 28. So *upper* is updated to 28. Node 3 is the next E-node. LCBB terminates now as $\hat{c}(3) = 36 > upper$.

In the preceding example, LCBB was modified slightly to handle nodes close to a solution node differently from other nodes. Node 6 is only two levels from a solution node. Rather than evaluate $\hat{c}$ at the children of 6 and then obtain their grandchildren, we just obtained an optimal solution for that subtree by a complete search with no bounding. We could have done something similar when generating the tree of Figure 8.12. Since node 6 is only two levels from the leaf nodes, we can simply skip computing $\hat{c}$ for the children and grandchildren of 6, generate all of them, and pick the best. This works out to be quite efficient as it is easier to generate a subtree with a small number of nodes and evaluate all the solution nodes in it than it is to compute $\hat{c}$ for one of the children of 6. This latter statement is true of many applications of branch-and-bound. Branch-and-bound is used on large subtrees. Once a small subtree is reached (say one with 4 or 6 nodes in it), then that subtree is fully evaluated without using the bounding functions.

We have now seen several branch-and-bound strategies for the traveling salesperson problem. It is not possible to determine analytically which of these is the best. The exercises describe computer experiments that determine empirically the relative performance of the strategies suggested.

EXERCISES

1. Consider the traveling salesperson instance defined by the cost matrix

$$\begin{bmatrix} \infty & 7 & 3 & 12 & 8 \\ 3 & \infty & 6 & 14 & 9 \\ 5 & 8 & \infty & 6 & 18 \\ 9 & 3 & 5 & \infty & 11 \\ 18 & 14 & 9 & 8 & \infty \end{bmatrix}$$

   (a) Obtain the reduced cost matrix

   (b) Using a state space tree formulation similar to that of Figure 8.10 and $\hat{c}$ as described in Section 8.3, obtain the portion of the state space tree that will be generated by LCBB. Label each node by its $\hat{c}$ value. Write out the reduced matrices corresponding to each of these nodes.