SUPPLEMENT CD5

ALGORITHMS FOR NETWORK MODELS

In Chapter 6 we presented mathematical models that can be represented by networks. These include the transportation, capacitated transshipment, assignment, traveling salesman, shortest path, minimal spanning tree, and maximum flow models. We pointed out that, although each of these models can be formulated as a linear program and solved using standard linear programming techniques, because of their special network structure, more efficient input and solution procedures are available. Here we present an overview of the algorithms used in many general-purpose management science solution modules (including WINQSB) to determine the optimal solution to these network models.

I THE TRANSPORTATION PROBLEM—THE TRANSPORTATION ALGORITHM

In Section 6.2 of the text, we introduced the basic transportation problem for finding the minimum total shipping cost of a particular item from m sources, each with a different supply, to n destinations, each with a particular demand. Here we present a two-part method for solving such problems. First, we find an initial basic feasible solution using a particular "starting procedure"; then we generate subsequent basic feasible solutions using the *transportation algorithm* until we find an optimal solution. This algorithm is actually a compact version of the simplex method (discussed in Supplement CD3) which uses a simplified procedure for selecting the entering and leaving basic variables and generating new tableaus. Although knowledge of the simplex method is useful for understanding the "whys" of the transportation algorithm, the steps of the procedure can be performed without this background.

To use the transportation algorithm, the total supply must equal total demand. If total supply exceeds total demand, we create a *dummy destination*, whose demand equals the difference between the total supply and total demand; if total supply is less than total demand, we create a *dummy source*, whose supply equals the difference. All unit shipping costs into a dummy destination or out of a dummy source are 0.

To illustrate the transportation procedure, we consider the following problem which includes unit shipping costs (Cij), supplies (Si), and demands (Dj); the total supply = total demand = 40.

		DESTINATIONS				Supply
		D1	D2	D3	D4	
	S1	50	75	30	45	12
Sources	S2	65	80	40	60	17
	S3	40	70	50	55	11
Demand		10	10	10	10	

THE TRANSPORTATION TABLEAU

The decision variables of a transportation problem, Xij, represent the amount shipped from source i to destination j. We indicate these shipments in a tableau; the rows represent the sources, and the columns represent the destinations. As shown in Figure CD5.1, each variable in a transportation tableau is represented by a box. In the upper right corner of the box, we list the variable's unit shipping cost.

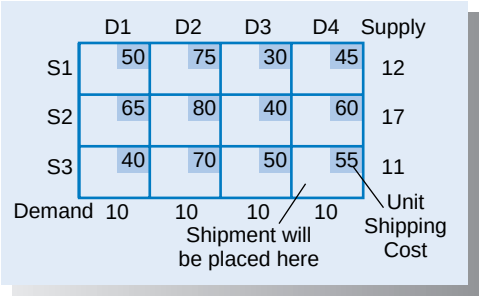

FIGURE CD5.1 The Transportation Tableau

In the simplex method, a variable is either nonbasic (i.e., it is assigned a 0 value) or basic, in which case it can have a positive value. Only nonbasic variables can have nonzero net marginal cost values, while the net marginal cost values for all basic variables are 0. For those who have studied the simplex method in Supplement CD3, the net marginal costs were referred to as CJ–ZJ values. Since the decision variables for this problem are denoted by double subscripts, i and j, we denote their CJ–ZJ values by Cij–Zij. Thus, for each variable in a transportation problem, we need only to keep track of its Cij–Zij value if it is a nonbasic variable (since Xij = 0), or its value, Xij, if it is a basic variable (since its Cij–Zij = 0).

Recall that in Section 6.2 of the text we showed for a problem with m sources and n destinations, there are m + n − 1 basic variables. The above problem, then, has 3 + 4 − 1 = 6 basic variables.

STARTING SOLUTION PROCEDURES

If we solved this problem using the standard simplex technique, we would have to add artificial variables to all constraints, thus requiring many iterations (at least six, in this case) before obtaining the first basic feasible solution. The structure of the transportation constraint coefficients, however, allows us to use one of several special *starting procedures* for transportation problems. Here we illustrate three of the more popular procedures.

Northwest Corner Starting Solution

The *Northwest Corner* method is a quick and efficient starting solution procedure that is easy to implement. It works as follows:

Northwest Corner Starting Procedure

1. Select the *remaining* variable in the upper left (northwest) corner and note the supply remaining in the row, s, and the demand remaining in the column, d.
2. Allocate the minimum of s or d to this variable. If this minimum is s, eliminate all variables in its row from future consideration and reduce the demand in its column by s; if the minimum is d, eliminate all variables in the column from future consideration and reduce the supply in its row by d.

REPEAT THESE STEPS UNTIL ALL SUPPLIES HAVE BEEN ALLOCATED.

For our problem, the sequence of steps for the Northwest Corner starting procedure is as follows:

	Northwest Corner	Remaining Supply in the Row (s)	Remaining Demand in the Column (d)	Allocate	Modifications
1.	X11	12	10	X11 = 10	Eliminate column 1; reduce row 1 supply to 12 − 10 = 2
2.	X12	2	10	X12 = 2	Eliminate row 1; reduce column 2 demand to 10 − 2 = 8
3.	X22	17	8	X22 = 8	Eliminate column 2; reduce row 2 supply to 17 − 8 = 9
4.	X23	9	10	X23 = 9	Eliminate row 2; reduce column 3 demand to 10 − 9 = 1

After the fourth step, only row 3 is left with a remaining supply of 11. Since the remaining demand is now 1 for column 3 and 10 for column 4, we make the allocations X33 = 1 and X34 = 10. These results are given by the transportation tableau in Figure CD5.2. While the northwest corner procedure is quick and easy to implement, unfortunately, it ignores costs altogether and frequently results in a starting solution that is not very close to the optimal solution.

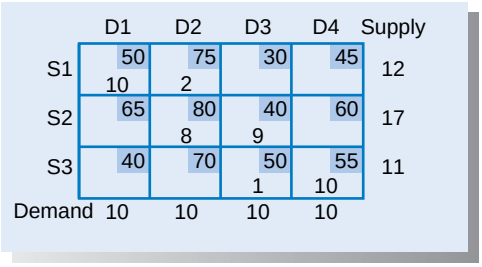

FIGURE CD5.2 Northwest Corner Starting Solution

Least Cost Starting Solution

Like the Northwest Corner method, the *least cost starting procedure* is also easy. It does, however, take costs into account as follows:

Least Cost Starting Procedure

1. For the remaining variable with the lowest unit cost, determine the remaining supply left in its row, s, and the remaining demand left in its column, d (break ties arbitrarily).
2. Allocate the minimum of s or d to this variable. If this minimum is s, eliminate all variables in its row from future consideration and reduce the demand in its column by s; if the minimum is d, eliminate all variables in the column from future consideration and reduce the supply in its row by d.

REPEAT THESE STEPS UNTIL ALL SUPPLIES HAVE BEEN ALLOCATED.

For our problem, the sequence of steps for the least cost starting procedure is as follows:

	Least Cost Cell	*Remaining Supply in the Row (s)*	*Remaining Demand in the Column (d)*	*Allocate*	*Modifications*
1.	X13 Cost = 30	12	10	X13 = 10	Eliminate column 3; reduce row 1 supply to 12 − 10 = 2
2.	X31 Cost = 40	11	10	X31 = 10	Eliminate column 1; reduce row 3 supply to 11 − 10 = 1
3.	X14 Cost = 45	2	10	X14 = 2	Eliminate row 1; reduce column 4 demand to 10 − 2 = 8
4.	X34 Cost = 55	1	8	X34 = 1	Eliminate row 3; reduce column 4 demand to 8 − 1 = 7

Since only row 2 is left with a remaining supply of 17 and the remaining demand for the only remaining columns is now 10 for column 2 and 7 for column 4, we make the final allocations of X22 = 10 and X24 = 7. This results in the transportation tableau shown in Figure CD5.3.

	D1	D2	D3	D4	Supply
S1	50	75	30 10	45 2	12
S2	65	80 10	40	60 7	17
S3	40 10	70	50	55 1	11
Demand	10	10	10	10	

FIGURE CD5.3 Least Cost Starting Solution

The major problem with the least cost approach is that, after initially allocating to one or two low-cost routes, the last few allocations may be to very high-cost routes; that is, the method does not take into account relative costs. We note, for example, that, in least cost allocation for our sample problem, X22 = 10; that is, 10 units are shipped to the most costly cell (with a unit cost of $80).

Vogel's Approximation Method (VAM)

Vogel's Approximation Method (VAM) introduces the concept of relative costs into the starting method. It requires more start-up calculations, but, once a starting solution is reached using this method, the optimal solution is typically attained more quickly than in the previous two methods. The VAM method proceeds as follows:

Vogel's Approximation Method Starting Procedure

1. For each remaining row and column, determine the difference between the lowest two *remaining* costs; these are called the *row and column penalties*.
2. Select the row or column with the largest penalty found in step 1 and note the supply remaining for its row, s, and the demand remaining in its column, d.
3. Allocate the minimum of s or d to the variable in the selected row or column with the lowest remaining unit cost. If this minimum is s, eliminate all variables in its row from future consideration and reduce the demand in its column by s; if the minimum is d, eliminate all variables in the column from future consideration and reduce the supply in its row by d.

   REPEAT THESE STEPS UNTIL ALL SUPPLIES HAVE BEEN ALLOCATED.

Applying the Vogel Approximation Method to our sample problem results in the following iterations:

Iteration 1: Row Penalties: (1) 45 − 30 = 15
(2) 60 − 40 = 20 <=== largest
(3) 50 − 40 = 10

Column Penalties: (1) 50 − 40 = 10
(2) 75 − 70 = 5
(3) 40 − 30 = 10
(4) 55 − 45 = 10

Since row 2 has the largest penalty, we allocate as many items as possible to the cell with the lowest cost of 40 (column 3). Since the supply in row 2 is 17 and the demand in column 3 is 10, we set X23 = 10, reduce the supply for row 2 to 17 − 10 = 7, and eliminate column 3 from future consideration.

Iteration 2: Eliminating column 3 has no effect on the remaining column penalties but causes us to reevaluate the row penalties:

New Row Penalties: (1) 50 − 45 = 5
(2) 65 − 60 = 5
(3) 55 − 40 = 15

The largest of these and the three remaining column penalties is the 15 for row 3. We allocate as much supply as possible to the least cost remaining in row 3; that is, to X31 (unit cost = 40). Since the remaining supply in row 3 is 11 and the remaining demand in column 1 is 10, we set X31 = 10, reduce the supply for row 3 to 11 − 10 = 1, and eliminate column 1 from future consideration.

Iteration 3: Eliminating column 1 has no effect on the remaining column penalties, but it does affect the row penalties:

New Row Penalties: (1) 75 − 45 = 30
(2) 80 − 60 = 20
(3) 70 − 55 = 15

The maximum of these and the two remaining column penalties (5 for column 2 and 10 for column 4) is the 30 for row 1. We allocate as much as possible to the least cost remaining in row 1—that is, X14 (unit cost = 45). Since the remaining supply in row 1 is 12 and the remaining demand column 4 is 10, we set X14 = 10, reduce the supply for row 1 to 12 − 10 = 2, and eliminate column 4.

Now since there is only one column left, column 2, with a remaining demand of 10, and the remaining supplies at rows 1, 2, and 3 are 2, 7, and 1, respectively, we make the allocations X12 = 2, X22 = 7, X23 = 1. The resulting basic feasible solution is shown in Figure CD5.4.

	D1	D2	D3	D4	Supply
S1	50	75 2	30	45 10	12
S2	65	80 7	40 10	60	17
S3	40 10	70 1	50	55	11
Demand	10	10	10	10	

FIGURE CD5.4 Vogel Approximation Method (VAM), Starting Solution

SOLVING FOR THE OPTIMAL SOLUTION—TRANSPORTATION SIMPLEX METHOD

Once we have determined a starting basic feasible solution, we use a compact form of the simplex method to move to an optimal solution. Since the basic transportation problem is a minimization problem, the simplex method proceeds as follows:

1. Find the current Cij–Zij values for each nonbasic variable and select the one with the most negative Cij–Zij value as the entering variable; if all Cij–Zij values are nonnegative, the current solution is optimal.
2. Determine which basic variable reaches 0 first when the entering variable is increased.
3. Determine a new basic solution and repeat the steps.

We will illustrate how the transportation simplex method performs these steps starting with the tableau generated by the VAM starting procedure.

STEP I: Determine The Cij–Zij Values for the Nonbasic Variables

We use a method based on duality in linear programming (see Supplement CD2), known as the *modified distribution approach* (MODI), to determine the marginal costs for the nonbasic variables. This method assumes that, if necessary, a dummy destination or a dummy source has been added so that total supply equals total demand. If that is the case, m supply constraints and all n demand constraints can be written as equalities. The approach, which is actually quite simple to implement, is developed using the following arguments.

1. If Ui is the dual variable associated with the i-th supply constraint, and Vj is the dual variable associated with the j-th demand constraint, then for shipments from node i to node j, we can find the corresponding Zij value by Zij = Ui − Vj. Thus the Cij–Zij value for variable Xij is found by

$$Cij - Zij = Cij - (Ui - Vj) = Cij - Ui + Vj$$

2. Given that there is a redundant equation among the m + n constraints (and any of the m + n constraints can be considered the redundant one), we can show that the Ui or Vj associated with the redundant equation is 0. Thus, we can select one Ui or Vj arbitrarily and set it to 0. We will set U1 = 0.
3. Since the Cij–Zij values for *basic* variables are 0 (i.e., Cij − Ui + Vj = 0 for basic variables), we can easily solve for the remaining values of the Uis and Vjs from the m + n − 1 equations for the basic variables.

4. Once we have determined Ui's and Vj's, we can find the Cij–Zij values for the nonbasic variables by

$$Cij - Zij = Cij - Ui + Vj$$

We illustrate this technique using the solution represented by the VAM tableau for our sample problem. The basic variables are in cells X12, X14, X22, X23, X31, and X32. We start by setting U1 = 0. The remaining Uis and Vjs are determined as follows:

Basic Cell	*Basic Cij–Zij Values = 0*	*Substitute*	*Implies*
X12	C12 − U1 + V2 = 0	75 − 0 + V2 = 0	V2 = −75
X14	C14 − U1 + V4 = 0	45 − 0 + V4 = 0	V4 = −45
X22	C22 − U2 + V2 = 0	80 − U2 + (−75) = 0	U2 = 5
X23	C23 − U2 + V3 = 0	40 − 5 + V3 = 0	V3 = −35
X32	C32 − U3 + V2 = 0	70 − U3 + (−75) = 0	U3 = − 5
X31	C31 − U3 + V1 = 0	40 − (−5) + V1 = 0	V1 = −45

We can now obtain the Cij–Zij values for the nonbasic variable cells:

Variable Cell	*Cij–Zij Calculation*
X11	C11 − U1 + V1 = 50 − 0 − 45 = 5
X13	C13 − U1 + V3 = 30 − 0 − 35 = −5
X21	C21 − U2 + V1 = 65 − 5 − 45 = 15
X24	C24 − U2 + V4 = 60 − 5 − 45 = 10
X33	C33 − U3 + V3 = 50 − (−5) − 35 = 20
X34	C34 − U3 + V4 = 55 − (−5) − 45 = 15

The most negative Cij–Zij value is − 5, which is associated with the nonbasic variable, X13; thus, we select X13 as the entering variable.

STEP 2—Determine Which Current Basic Variable Reaches 0 First

A negative Cij–Zij value for a nonbasic variable cell implies that a reduction in the value of the objective function is possible for every unit shipped from the cell's source to the cell's destination. We obtain this cost savings by altering the values of some of the other *basic* variables. The property that enables us to find out how much the entering variable can increase is called the *cycle property*. It states that, for every nonbasic variable, there is a unique cycle that joins this cell and the cells of some (not necessarily all) of the other *basic* variables using only horizontal and vertical lines on the transportation tableau.

By alternately modifying the variables in this cycle (an addition to one cell in the cycle is followed by a subtraction from the next cell in the cycle) we can maintain the conservation of supply and demand. For small problems, such as ours, the cycle is easy to determine by trial and error. For example, in step 1 we determined that we wish to increase the allocation to X13; each unit shipped here will save $5. As Figure CD5.5 indicates, a cycle that contains X13 and a subset of the basic variables is X13, X23, X22, and X12.

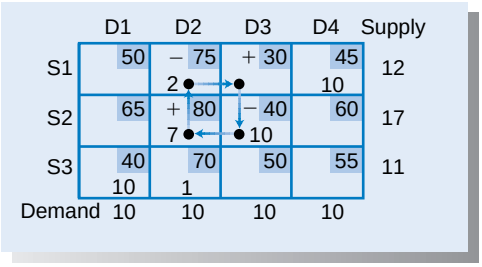

FIGURE CD5.5 Determining the Cycle of the Entering Variable with the Basic Variables

Now if we add one unit to X13, we must subtract a unit from X23 (leaving 9), add it back to X22 (giving 8), and subtract it from X12 (leaving 1); this saves \$5. If we add two units to X13, we must subtract two from X23 (leaving 8), add two to X22 (giving 9), and subtract two from X12 (leaving 0); this saves 2(\$5) = \$10.

Since X12 has now reached 0, no further reduction is possible for this variable; 2 is the most that presently can be allocated to X13. Note that this is the minimum amount on the cycle *where a subtraction is to be made.*

STEP 3: Determine the Next Transportation Tableau

The transition to the next tableau is very simple. We alter the variables along the cycle by the amount determined in step 2 (in our case, 2). The entering variable (X13) now becomes a new basic variable with a value of 2; X23 will now be 10 − 2 = 8, and X22 will be 7 + 2 = 9. X12, the basic variable that goes to 0, becomes the new nonbasic variable (at zero value). The result is shown in Figure CD5.6. We then repeat the three-step process.

	D1	D2	D3	D4	Supply
S1	50	75	30 2	45 10	12
S2	65	80 9	40 8	60	17
S3	40 10	70 1	50	55	11
Demand	10	10	10	10	

FIGURE CD5.6 The Transportation Tableau for Iteration 2

The steps of this method are summarized as follows:

Transportation Simplex Method

Find an initial basic feasible solution by some starting procedure. Then,

1. Set U1 = 0. Solve for the other Uis and Vjs by:

$$Cij - Ui + Vj = 0 \text{ for basic variables.}$$

Then calculate the Cij–Zij values for nonbasic variables by:

$$Cij - Zij = Cij - Ui + Vj$$

Choose the nonbasic variable with the most negative Cij–Zij value as the entering variable. If all Cij–Zij values are nonnegative, STOP; the current solution is optimal.

2. Find the cycle that includes the entering variable and some of the BASIC variables. Alternating positive and negative changes on the cycle, determine the "change amount" as the smallest allocation on the cycle at which a subtraction will be made.

3. Modify the allocations to the variables of the cycle found in step 2 by the "change amount" and return to step 1.

We now continue the algorithm on the new tableau. Applying step 1, we again start by setting U1 = 0. We have:

Basic Cell	Basic Cij–Zij Values = 0	Substitute	Implies
X13	C13 − U1 + V3 = 0	30 − 0 + V3 = 0	V3 = −30
X14	C14 − U1 + V4 = 0	45 − 0 + V4 = 0	V4 = −45
X23	C23 − U2 + V3 = 0	40 − U2 + (−30) = 0	U2 = 10
X22	C22 − U2 + V2 = 0	80 − 10 + V2 = 0	V2 = −70
X32	C32 − U3 + V2 = 0	70 − U3 + (−70) = 0	U3 = 0
X31	C31 − U3 + V1 = 0	40 − 0 + V1 = 0	V1 = −40

The Cij–Zij values for the nonbasic variables are now:

Variable Cell	Cij–Zij Calculation
X11	C11 − U1 + V1 = 50 − 0 − 40 = 10
X12	C12 − U1 + V2 = 75 − 0 − 70 = 5
X21	C21 − U2 + V1 = 65 − 10 − 40 = 15
X24	C24 − U2 + V4 = 60 − 10 − 45 = 5
X33	C33 − U3 + V3 = 50 − 0 − 30 = 20
X34	C34 − U3 + V4 = 55 − 0 − 45 = 10

Since all the Cij–Zij values are now nonnegative, the solution shown in Figure CD5.2 is optimal. This solution is summarized as follows:

From	To	Amount	Unit Cost	Transportation Cost
S1	D3	2	$30	$ 60
S1	D4	10	$45	$ 450
S2	D2	9	$80	$ 720
S2	D3	8	$40	$ 320
S3	D1	10	$40	$ 400
S3	D2	1	$70	$ 70
				Total = $2020

A NOTE ON TIES IN THE PROCESS: DEGENERACY

When performing a starting procedure, we may find that a supply is depleted and a demand is satisfied simultaneously; thus, we can eliminate both a row and a column at the same time. Now, when we finish the procedure, fewer than m + n − 1 basic variables remain. This is called *degeneracy*. But there must be m + n − 1 basic variables for the transportation simplex method to work!

When this happens, we arbitrarily select variables to be *basic* variables at 0 value until we get m + n − 1 basic variables. The only restriction on the choice of variables to be made basic is that, when these variables are collectively included with the other basic variables, they must not form any cycles with any subset of the other basic variables (recall that cycles are formed in a transportation tableau by connecting cells using only horizontal and vertical lines).

Degeneracy also occurs when there is a tie for the leaving variable (two or more variables reach 0 at the same time) in step 2 of the algorithm. In this case, we select only one of these variables to be nonbasic; the others remain basic but at 0 value.

II THE CAPACITATED TRANSSHIPMENT PROBLEM—THE OUT-OF-KILTER ALGORITHM

Another streamlined form of the simplex method, the *out-of-kilter algorithm*, can be used to solve a network problem with sources, destinations, intermediate nodes, unit shipping costs, and maximum capacities between nodes. Shipments may or may not be

allowed between supply nodes or between demand nodes.[1] In the most general case, the one we illustrate here, such shipments are possible.

The concepts behind this algorithm follow those of the transportation problem. Once we find a basic feasible solution, we calculate Cij–Zij values for the flow along each nonbasic shipping route that is not currently being used to see if increasing flow along the arc will reduce the overall total cost. I*ncreasing* the flow along an arc with a *negative* Cij–Zij value reduces the total cost.

The new wrinkle in this problem is that arcs with flows at their upper bound are also considered nonbasic. We calculate the Cij–Zij values for these arcs as well to see if a *reduction* in the flow on these arcs can possibly save money. D*ecreasing* the flow along an arc with a *positive* Cij–Zij value reduces the total cost.

If increasing the flow along an arc that is not being used or reducing the flow along an arc used to capacity results in cost savings, the system is said to be *out of kilter*. The out-of-kilter algorithm uses a simple technique for calculating the Cij–Zij values for nonbasic arcs. Once those values are calculated, if we determine that a modification to the flow will improve the objective function value, the algorithm provides a way to make this change quite easily. We then recalculate the Cij–Zij values for the new solution and repeat the process.

The out-of-kilter method assumes that the total supply at the supply nodes equals the total demand at the demand nodes. If not, we create a dummy supply node with zero-cost arcs from the dummy node to all demand nodes, or a dummy demand node with zero-cost arcs from all supply nodes to the dummy node to balance the problem.

As stated above, there are two types of *nonbasic variables* in a capacitated transshipment problem: (1) those at their lower bound (0); and (2) those at their upper bound (m). Variables that are at neither their lower bound nor their upper bound are the *basic variables*. There are n − 1 basic variables (where n is the total number of nodes).[2] The arcs corresponding to these basic variables form a spanning tree in the network. The following are the steps of the out-of-kilter algorithm:

The Out-of-Kilter Algorithm

I*nitialization*: Find an initial basic feasible solution.

I*terative Steps*:

1. Calculate value of the dual variables:
(1) Set U1 = 0
(2) Solve for the other Uis by Cij − Ui + Uj = 0 for basic variable arcs. (The flow is FROM node i TO node j.)

2. Calculate the Cij–Zij values for the nonbasic arcs:

$$Cij - Zij = Cij - Ui + Uj$$

3. For nonbasic variables at their lower bound (0), variables with negative Cij–Zij values are out of kilter: Determine the variable with the most negative Cij–Zij value.
For nonbasic variables at their upper bound (Mij), variables with positive Cij–Zij values are out of kilter: Determine the variable with the most positive Cij–Zij value.

(*continued*)

[1]The most general form of this problem allows for lower bounds on the arcs other than 0. For simplicity, we simply assume that all arcs have a minimum capacity of 0.

[2]If the number of variables that are at neither their upper nor lower bound is less than n − 1, the problem is degenerate and any arc at its lower bound or any arc at its upper bound can be made basic as long as the arc does not form a cycle with any of the other basic variables.

Of these two, select the one with the larger absolute Cij–Zij value as the entering variable. If this largest value is 0, STOP; the current flow is optimal.

4. The arc for this variable forms a cycle in the network with a set of other basic variables. If the entering variable is at its lower bound, increase the flow on the arc; if it is at its upper bound, decrease the flow on the arc. In either case, the flow along the other arcs in the cycle must be adjusted to maintain a feasible flow. Adjust the flow until some arc reaches either 0 or its upper bound (Mij). This is the new nonbasic variable.

GO TO STEP 1.

We illustrate this approach for the capacitated transshipment problem discussed in Section 6.2 of the text and shown in Figure CD5.7.

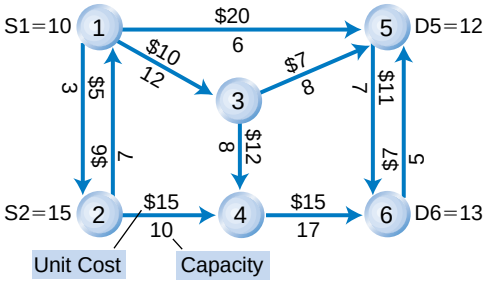

FIGURE CD5.7 Capacitated Transshipment Problem

INITIALIZATION STEP

We found the following basic feasible flow by trial and error:

		Lower Bound		Upper Bound	
Basic Variables	**Flow**	**Nonbasic Variables**	**Flow**	**Nonbasic Variables**	**Flow**
X13	9	X12	0	X15	6
X21	5	X56	0	X24	10
X34	3	X65	0		
X35	6				
X46	13				

The basic variable tree for this solution is highlighted in Figure CD5.8; the flows are shown in the circles below the arcs.

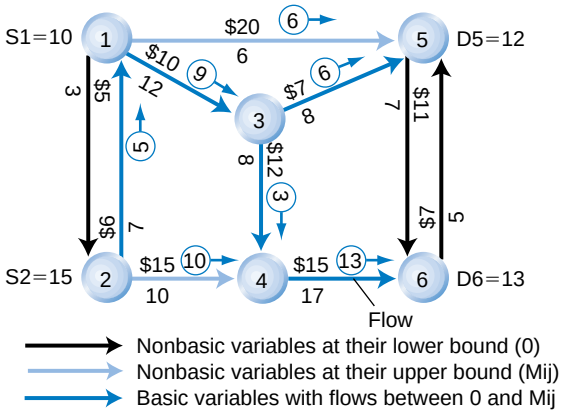

FIGURE CD5.8 Transshipment Problem: A First Basic Feasible Solution

ITERATION 1

Step 1: *Determine the Uis*

Set U1 to 0. Then,

Basic Variable	Cij–Zij = 0	Substitute	Implies
X13	C13 − U1 + U3 = 0	10 − 0 + U3 = 0	U3 = −10
X21	C21 − U2 + U1 = 0	6 − U2 + 0 = 0	U2 = 6
X34	C34 − U3 + U4 = 0	12 − (−10) + U4 = 0	U4 = −22
X35	C35 − U3 + U5 = 0	7 − (−10) + U5 = 0	U5 = −17
X46	C46 − U4 + U6 = 0	15 − (−22) + U6 = 0	U6 = −37

Steps 2 and 3: *Determine Cij–Zij Values for Nonbasic Variables and the Entering Variable*

Lower Bound Variables

Variable	Cij–Zij Calculation	Out of Kilter?
X12	C12 − U1 + U2 = 5 − 0 + 6 = 11	No
X56	C56 − U5 + U6 = 11 − (−17) + (−37) = −9	Yes
X65	C65 − U6 + U5 = 7 − (−37) + (−17) = 27	No

Upper Bound Variables

Variable	Cij–Zij Calculation	Out of Kilter?
X15	C15 − U1 + U5 = 20 − 0 + (−17) = 3	Yes
X24	C24 − U2 + U4 = 15 − 6 + (−22) = −1	No

Since X56 is the variable that is most out of kilter, ($|-9| > +3$) it becomes the entering variable.

Step 4: *Determine the Change to the Basic Variables*

As Figure CD5.8 shows, X56 forms a cycle with the basic variables X35, X34, and X46. Since X56 is at its lower bound (0), it is increased. As Figure CD5.9 shows, when X56 is increased, X35 must be increased and X46 decreased. Hence, X34 must also be decreased.

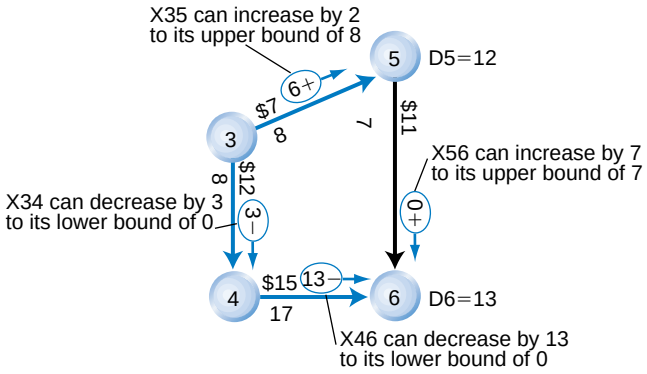

FIGURE CD5.9 Cycle Formed by X56 and Current Basic Variables

The calculations for the maximum amount of change in these four variables before reaching a bound shown in Figure CD5.9 are as follows:

Variables Increased	Current Value	Upper Bound	Maximum Increase
X56	0	7	7 − 0 = 7
X35	6	8	8 − 6 = 2
Variables Decreased	**Current Value**	**Lower Bound**	**Maximum Decrease**
X34	3	0	3 − 0 = 3
X46	13	0	13 − 0 = 13

The minimum of these maximum changes is 2, determined by increasing X35 to its upper bound. Thus, we add 2 to the flows on arcs X56 and X35 and subtract 2 from the flows along the arcs X34 and X46 to get the network flow diagram depicted in Figure CD5.10.

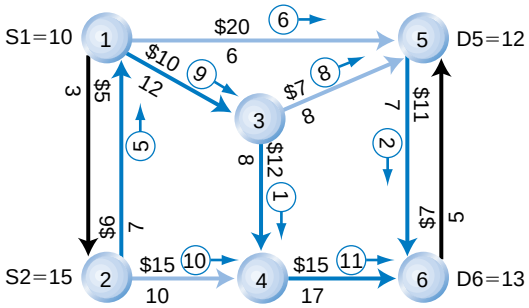

FIGURE CD5.10 Augmenting Flow by 2 on X56, X35, X34, and X45 Gives the Next Iteration

ITERATION 2

Basic Variable	Flow	Lower Bound Nonbasic Variables	Flow	Upper Bound Nonbasic Variables	Flow
X13	9	X12	0	X15	6
X21	5	X65	0	X24	10
X34	1			X35	8
X46	11				
X56	2				

Step 1: *Determine the Uis*

Set U1 to 0. Then,

Basic Variable	Opportunity Cost = 0	Substitute	Implies
X13	C13 − U1 + U3 = 0	10 − 0 + U3 = 0	U3 = −10
X21	C21 − U2 + U1 = 0	6 − U2 + 0 = 0	U2 = 6
X34	C34 − U3 + U4 = 0	12 − (−10) + U4 = 0	U4 = −22
X46	C46 − U4 + U6 = 0	15 − (−22) + U6 = 0	U6 = −37
X56	C56 − U5 + U6 = 0	11 − U5 + (−37) = 0	U5 = −26

Steps 2: *Determine Cij–Zij Values for Nonbasic Variables*

Lower Bound Variables

Variable	Cij–Zij Calculation	Out of Kilter?
X12	C12 − U1 + U2 = 5 − 0 + 6 = 11	No
X65	C65 − U6 + U5 = 7 − (−37) + (−26) = 18	No

Upper Bound Variables

Variable	Cij–Zij Calculation	Out of Kilter?
X15	C15 − U1 + U5 = 20 − 0 + (−26) = −6	No
X24	C24 − U2 + U4 = 15 − 6 + (−22) = −1	No
X35	C35 − U3 + U5 = 7 − (−10) + (−26) = −9	No

Since no arcs are out of kilter, we have found the optimal solution:

From	To	Amount	Unit Cost	Transportation Cost
Node 1	Node 3	9	$10	$ 90
Node 1	Node 5	6	$20	$120
Node 2	Node 1	5	$ 6	$ 30
Node 2	Node 4	10	$15	$150
Node 3	Node 4	1	$12	$ 12
Node 3	Node 5	8	$ 7	$ 56
Node 4	Node 6	11	$15	$165
Node 5	Node 6	2	$11	$ 22
				Total = $645

A NOTE ON TIES FOR THE NEW NONBASIC VARIABLE: DEGENERACY

In step 3 of the algorithm, two or more basic variables may reach 0 or their upper bounds at the same time. This is a case of degeneracy. We select only one of these variables to be nonbasic at the next iteration; the others remain basic at either 0 value or their upper bound.

III THE ASSIGNMENT PROBLEM—THE HUNGARIAN ALGORITHM

The H*ungarian algorithm* for solving the assignment problem of a least cost assignment of m workers to m jobs is based on the observation that a number can be added to or subtracted from every number in a row or a column of an assignment cost matrix without affecting the optimal solution. A number subtracted from all numbers in a row or column is considered a minimum amount each worker will charge or that each job will cost; the resulting numbers can then be thought of as a premium above this fixed amount. Similarly, adding a number to a row or column can be regarded as an increase in the fixed price that each worker will charge or that each job will cost. This process of adding or subtracting a number from all numbers in a row or column is known as *matrix reduction*.

The idea behind the Hungarian algorithm is to reduce the matrix so that only nonnegative costs exist and at least one 0 remains in each row and column; then we attempt to make a complete assignment using only the 0 costs. If this is unsuccessful, we apply a systematic matrix reduction procedure that creates a new matrix of costs, in which we reduce to 0 an entry that previously had a positive cost. Then we make another attempt to find a complete 0 cost assignment. We repeat the process until we find an assignment of only 0 entries.

The following description of the Hungarian algorithm assumes that:

1. There is a cost assignment matrix for m "people" to be assigned to m "tasks." (If necessary we add dummy rows or columns consisting of all 0's so that the numbers of people and tasks are the same.)
2. All costs are nonnegative.
3. The problem is a minimization problem.

The Hungarian Algorithm

Initialization

1. For each row, subtract the minimum number from all numbers in that row.
2. In the resulting matrix, subtract the minimum number in each column from all numbers in the column.

(*continued*)

Iterative Steps

1. Make as many 0 cost assignments as possible. If all workers are assigned, STOP; this is the minimum cost assignment. Otherwise draw the minimum number of horizontal and vertical lines to cover all 0's in the matrix. (A method for making the maximum number of 0 cost assignments and drawing the minimum number of lines to cover all 0's follows.)
2. Find the smallest value not covered by the lines; this number is the *reduction value*.
3. Subtract the reduction value from all numbers not covered by any lines. Add the reduction value to any number covered by both a horizontal and vertical line.

GO TO STEP 1.

For small problems, we can usually determine the maximum number of zero-cost assignments by observation. For larger problems, we can use the following procedure:

Determining the Maximum Number of Zero-Cost Assignments

1. For each row, if only one 0 remains in the row, make that assignment and eliminate the *row* and *column* from consideration in the steps below.
2. For each column, if only one 0 remains, make that assignment and eliminate that *row* and *column* from consideration.
3. Repeat steps 1 and 2 until no more assignments can be made. (If 0's remain, this means that there are at least two 0's in each remaining row and column. Make an arbitrary assignment to one of these 0's and repeat steps 1 and 2.)

Again, for small problems, we can determine the minimum number of lines required to cover all the 0's by observation. The following procedure, based on network flow arguments, is used for larger problems:

Drawing the Minimum Number of Lines to Cover All 0's

1. Mark all rows with no *assignments* (with a "•").
2. For each row just marked, mark each column that *has a* 0 in that row (with a "•").
3. For each column just marked, mark each row that *has an assignment* in that column (with a "•").
4. Repeat steps 2 and 3 until no more marks can be made.
5. Draw lines through *unmarked rows* and *marked columns*.

Figure CD5.11 shows the sequence of matrix reductions required to solve the Ballston Electronics problem introduced in Section 6.3 of the text. In the third, fourth, and fifth matrices, we place a box around each 0 cost assignment and use a "•" to mark the rows and columns required to draw the minimum covering set of lines using the procedures outlined above. The minimum uncovered value is circled.

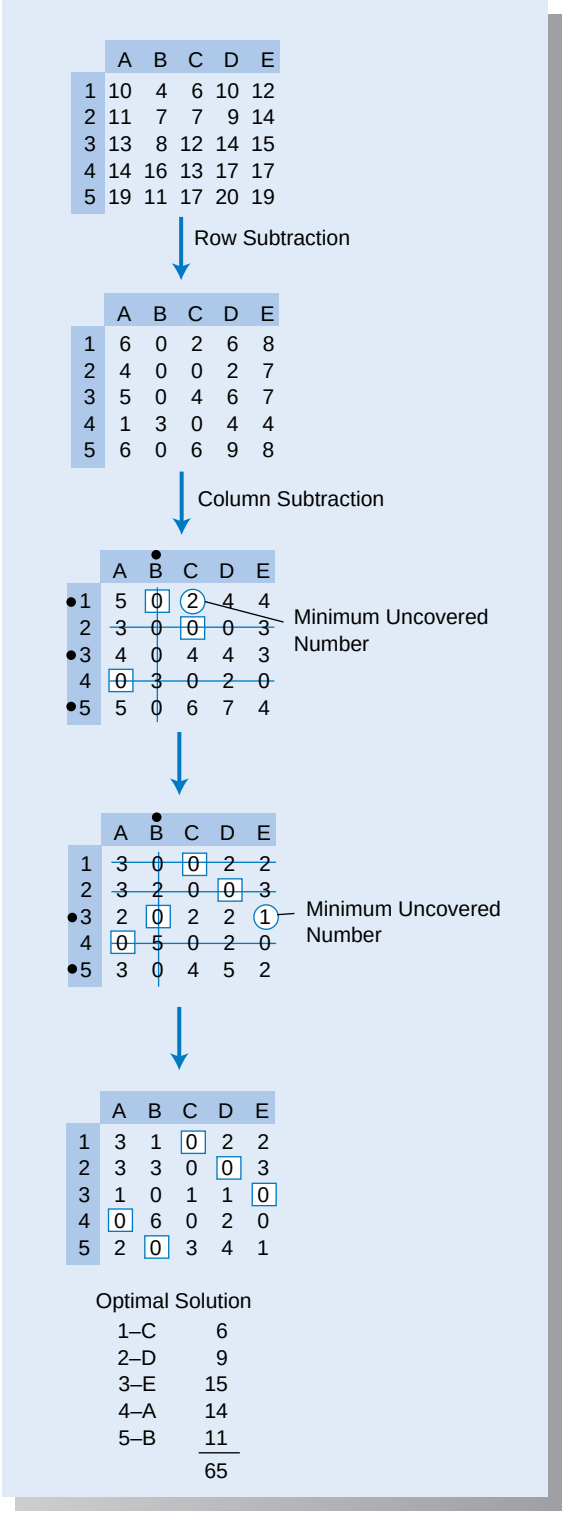

FIGURE CD5.11 Solution for the Ballston Electronics Assignment Problem

CONVERSION OF A MAXIMIZATION PROBLEM TO A MINIMIZATION PROBLEM

The Hungarian algorithm works only if the matrix is a cost matrix. In Section 6.3 of the text we mentioned that a maximization assignment problem can be converted to a minimization problem by creating a lost opportunity matrix. The problem then is to minimize the total lost opportunity.

After adding dummy rows or columns as necessary so that the number of jobs equals the number of workers, we can think of the lost opportunities for each job as the regret we suffer if the job is not performed by the worker who gives the highest return for the job. We find these by replacing each entry in the column by the difference between the entry and the largest entry in the column.

For example, suppose the following is a profit matrix for an assignment problem with three workers and four jobs. We first add a dummy row of 0's so that there are four rows and columns:

	J1	J2	J3	J4
W1	67	58	90	55
W2	58	88	89	56
W3	74	99	80	22
DUMMY	0	0	0	0

We find the lost opportunity matrix given below by subtracting each number in the J1 column from 74, each number in the J2 column from 99, each number in the J3 column from 90, and each number in the J4 from 56.

	J1	J2	J3	J4
W1	7	41	0	1
W2	16	11	1	0
W3	0	0	10	34
DUMMY	74	99	90	56

We can now apply the Hungarian algorithm to this lost opportunity matrix to determine the maximum profit set of assignments.

IV A TRAVELING SALESMAN ALGORITHM

The traveling salesman problem seeks a minimum cost cycle that includes all the nodes in a network. Numerous algorithms have been proposed, but few are efficient when the number of nodes is large. Here we present an approach that has been successful for problems with a small number of nodes (say less than 20).

THE BRANCH AND BOUND ALGORITHM

We begin by constructing an assignment-type matrix; the workers are the "FROM" nodes, and the jobs are the "TO" nodes. The matrix entries are the costs (or distances) between the FROM nodes to the TO nodes. The cost assigned FROM a node TO itself is the huge cost "+M." The idea behind the algorithm is to try to "assign" one city to follow another city using the assignment algorithm.

The difficulty with this approach is that the optimal solution found using the assignment algorithm may not be a cycle, or "tour," of all the cities, but rather an assignment consisting of two or more cycles, or "subtours." To eliminate the possibility of a particular subtour occurring, we apply a branch and bound algorithm that parallels that introduced in Supplement CD4 for mixed integer linear programs.

We can find an initial upper bound for the optimal objective function value, U, by calculating the cost of any tour. Initially, we set U to the objective function value of the tour 1–2–3– . . . –n–1. The branches consist of problems that have been revised by assigning a high cost (+M) to one of the arcs on a subtour (thus making that subtour

infeasible). We then apply the assignment algorithm to the revised problem.

Once we find the solution, we compare its objective function value to the current upper bound, U; if it is not less than U, it is fathomed (eliminated from consideration). If it is less than U and the solution is a tour, we reset U to the total cost of the tour; if it is not a tour, we must conduct additional branching from this problem. We continue the branch and bound process until we have fathomed all the branches and found the optimal solution.

The branch and bound steps are summarized below. (Remember that "fathoming" means ceasing further consideration of the branch.)

Branch and Bound Algorithm for the Traveling Salesman Problem

Initialization Step

Solve the problem as an assignment problem. If the optimal solution is a tour, the minimum distance has been found. If the optimal solution is not a tour, make this solution the first active branch of the branch and bound tree and set U = the objective function value of the tour 1–2–3–4–5 . . . –n.

Branch and Bound Steps

1. Choose an unfathomed branch. Consider the subtour consisting of the minimum number of arcs (say, there are k arcs on this subtour). Create k branches from this node, each corresponding to an assignment problem that assigns a very large value +M to one of the arcs of the subtour, making the subtour infeasible. For each branch, re-solve the assignment problem, giving an objective function value Z. (If there are no unfathomed branches, STOP; the optimal solution is the one that gives the value of U.)

2. For each of these branches:

If	***Then***
The problem is infeasible	• Fathom the branch
$Z \geq U$	• Fathom the branch
The solution is a tour and $Z < U$	• Fathom the branch • Reset $U = Z$ • Fathom all branches with $Z \geq U$
The solution is not a tour and $Z < U$	• This is a new active subproblem.

GO TO STEP 1

We illustrate this approach in the following problem.

A TRAVELING SALESMAN PROBLEM

FROM \ TO	1	2	3	4	5	6
1	M	63	22	20	18	10
2	60	M	56	40	54	58
3	28	54	M	20	21	15
4	40	39	37	M	24	41
5	30	40	26	20	M	18
6	30	56	24	40	26	M

Initialization

When we solve this problem as an assignment problem, the solution consists of two subtours: (1–6–5–3–1) and (2–4–2), giving Z = 164. Since we did not find a tour, we calculate the value of the tour (1–2–3–4–5–6–1) (= 63 + 56 + 20 + 24 + 18 + 30 = 221) and use this value as our first upper bound: U = 221.

Iteration 1
U = 221

The subtour with the smallest number of arcs is (2–4–2). We now create two new branches, one blocking the route from 2–4 (by changing the entry for arc (2,4) in the assignment matrix to +M), the other blocking the route from 4–2 (by changing the entry for arc (4,2) in the assignment matrix to +M.) We then re-solve each as an assignment problem.

Branch	Z	*Result*	*Action*
Block 2–4	174	Tour: 1–6–3–5–4–2–1	Fathom this branch. Reset U to 174 (since 174 < U = 221).
Block 4–2	166	Two subtours: (1–6–3–1) (2–4–5–2)	Continue branching on one of these subtours.

These results are depicted in the tree shown in Figure CD5.12.

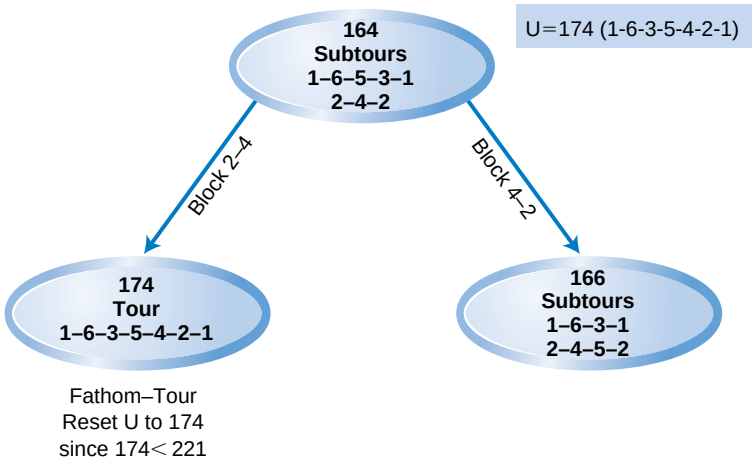

FIGURE CD5.12 The Traveling Salesman Approach: Iteration 1

Iteration 2
U = 174

Because the number of arcs on each subtour created by blocking the route from 4–2 is the same, we arbitrarily choose the subtour (2–4–5–2) to branch on. We create three new subproblems, which in addition to blocking the route from 4–2, block the routes from 2–4, 4–5, and 5–2 respectively. Re-solving each of these as an assignment problem gives the following:

Branch	Z	*Result*	*Action*
Block 2–4	178	Tour: 1–6–3–4–5–2–1	Fathom this branch. Do not reset U since 178 > U = 174.
Block 4–5	175	Tour: 1–6–3–5–2–4–1	Fathom this branch. Do not reset U since 175 > U = 174.
Block 5–2	193	Two subtours: (1–6–2–4–1) (3–5–3)	Fathom this branch since 193 > U = 174.

These results are summarized by the tree in Figure CD5.13. Since all branches are now fathomed, the optimal solution is the tour (1–6–3–5–4–2–1) giving Z = 174.

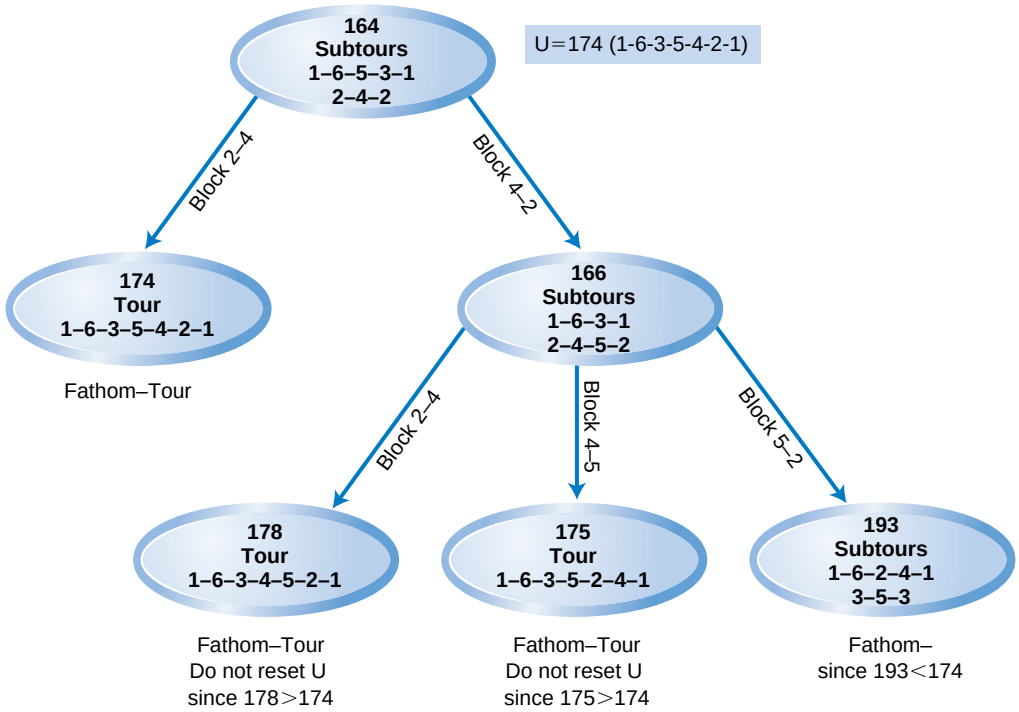

FIGURE CD5.13 The Traveling Salesman Approach: Iteration 2

IV The Shortest Path Problem: The Dijkstra Algorithm

Several algorithms have been proposed for finding the shortest path from a start node to a terminal node in a network. Here we present the *Dijkstra algorithm*, which is one of the easiest and most efficient algorithms for solving shortest path problems.

We begin the Dijkstra algorithm by labeling each node with the minimum distance found from the start node to the node "so far." The node is temporarily labeled with this distance. At each iteration of the process, we select the node with the smallest of the temporary labels; that label becomes a permanent label. This permanent label is the true minimum distance from the source node to that node because any further labels at this node include traveling through another node with a larger temporary label value. We repeat the process until all the nodes (or at least the terminal node) have been assigned permanent labels.

The following is a formal statement of the Dijkstra algorithm:

The Dijkstra Algorithm

Initialization Step

Assign a temporary label of 0 to the start node and $+\infty$ to all other nodes. (These are the minimum distances found thus far from the start node to all other nodes; we do not put the $+\infty$ values on the network.) *(continued)*

Iterative Steps

1. Find the node with the smallest temporary label and make it permanent. This node is the *assigned node*. If all nodes have permanent labels, STOP; the minimum distances have been found.
2. From the assigned node, consider all arcs to its adjacent nodes with temporary labels. For these adjacent nodes calculate:

   D = (Permanent label at assigned node) + (Arc distance)

   Replace the temporary label at the adjacent node by D *only if* the current label at the adjacent node is greater than D. If the label is replaced, record the assigned node that generated the label (shown next to the label valve).

   GO TO STEP 1.

We determine the shortest path by retracing the path of assigned nodes from the terminal node back to the start node. Figure CD5.14 illustrates the shortest path approach on a network.

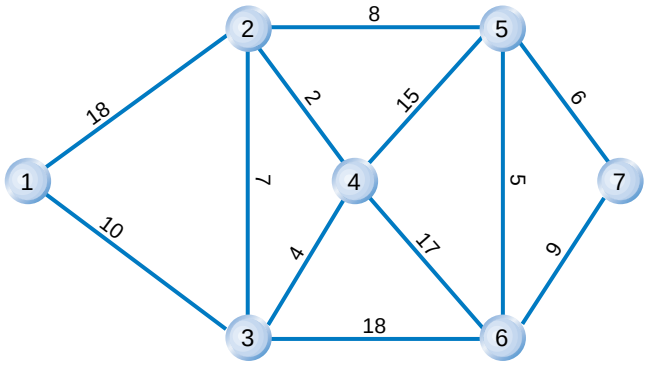

FIGURE CD5.14 Shortest Path: Find the Minimum Distance from Node 1 to Node 7

INITIALIZATION

All nodes have temporary labels of $+\infty$ (not shown), except for node 1 which has a temporary label of 0.

ITERATION 1

According to Figure CD5.14:

Minimum Temporary Label (Made Permanent): 0 at *Node* 1
Temporary Nodes Adjacent to Node 1: Nodes 2 and 3

Adjacent Node	*Distance from Assigned Node*	*New Temporary Label at Adjacent Node?*
2	$0 + 18 = 18 < \infty$	Yes—18 (1)
3	$0 + 10 = 10 < \infty$	Yes—10 (1)

This gives Figure CD5.15*a*.

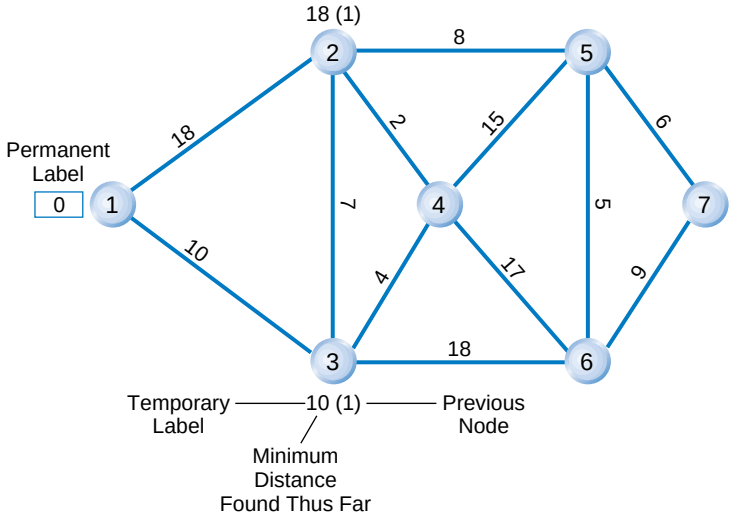

FIGURE CD5.15*a* Shortest Path Solution: Iteration 1

ITERATION 2

According to Figure CD5.15*a*:

Minimum Temporary Label (Made Permanent): 10 at N*ode* 3

Temporary Nodes Adjacent to Node 3: Nodes 2, 4, and 6

Adjacent Node	***Distance from Assigned Node***	***New Temporary Label at Adjacent Node?***
2	$10 + 7 = 17 < 18$	Yes—17 (3)
4	$10 + 4 = 14 < \infty$	Yes—14 (3)
6	$10 + 18 = 28 < \infty$	Yes—28 (3)

This gives Figure CD5.15*b*.

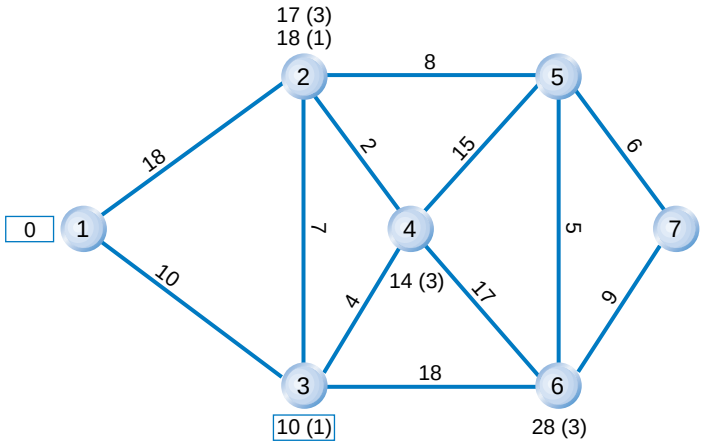

FIGURE CD5.15*b* Shortest Path Solution: Iteration 2

ITERATION 3

According to Figure CD5.15*b*:

Minimum Temporary Label (Made Permanent): 14 at N*ode* 4

Temporary Nodes Adjacent to Node 4: Nodes 2, 5 and 6

Adjacent Node	Distance from Assigned Node	New Temporary Label at Adjacent Node?
2	14 + 2 = 16 < 17	Yes—16 (4)
5	14 + 15 = 29 < ∞	Yes—29 (4)
6	14 + 17 = 31 ≮ 28	No

This gives Figure CD5.15*c*.

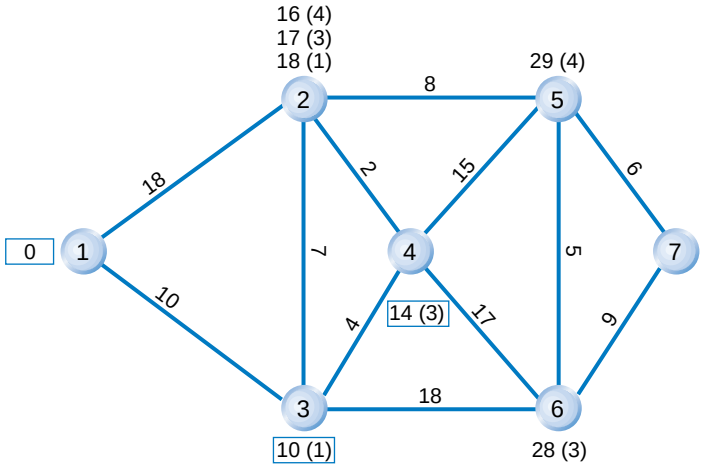

FIGURE CD5.15*c* Shortest Path Solution: Iteration 3

ITERATION 4

According to Figure CD5.15*c*:

Minimum Temporary Label (Made Permanent): 16 at Node 2
Temporary Nodes Adjacent to Node 2: Node 5

Adjacent Node	Distance from Assigned Node	New Temporary Label at Adjacent Node?
5	16 + 8 = 24 < 29	Yes—24 (2)

This gives Figure CD5.15*d*.

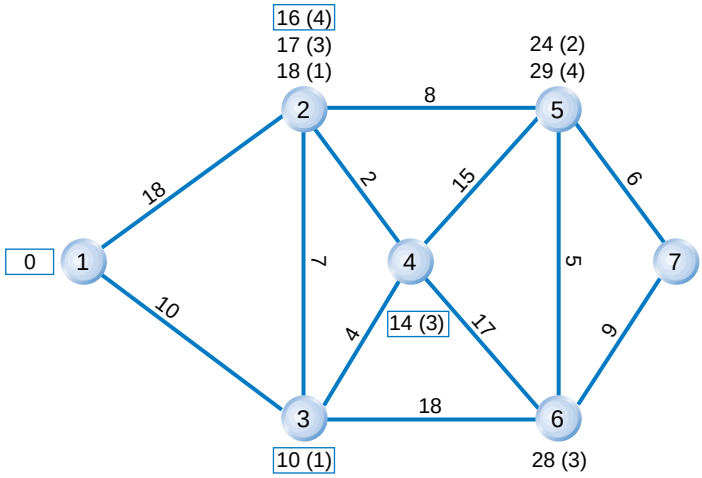

FIGURE CD5.15*d* Shortest Path Solution: Iteration 4

ITERATION 5

According to Figure CD5.15*d*:

Minimum Temporary Label (Made Permanent): 24 at N*ode* 5
Temporary Nodes Adjacent to Node 5: Nodes 6 and 7

Adjacent Node	*Distance from Assigned Node*	*New Temporary Label at Adjacent Node?*
6	24 + 5 = 29 < 28	No
7	24 + 6 = 30 < ∞	Yes—30 (5)

This gives Figure CD5.15*e*.

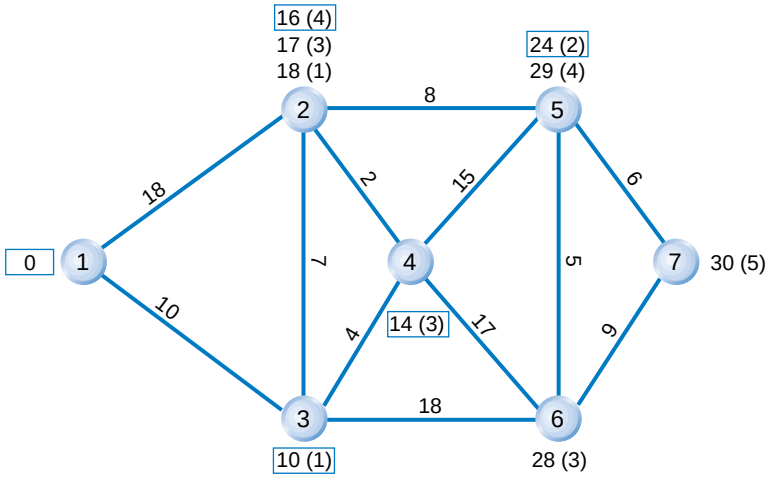

FIGURE CD5.15*e* Shortest Path Solution: Iteration 5

ITERATION 6

According to Figure CD5.15*e*:

Minimum Temporary Label (Made Permanent): 28 at N*ode* 6
Temporary Nodes Adjacent to Node 6: Node 7

Adjacent Node	*Distance from Assigned Node*	*New Temporary Label at Adjacent Node?*
7	28 + 9 = 37 < 30	No

This gives Figure CD5.15*f*.

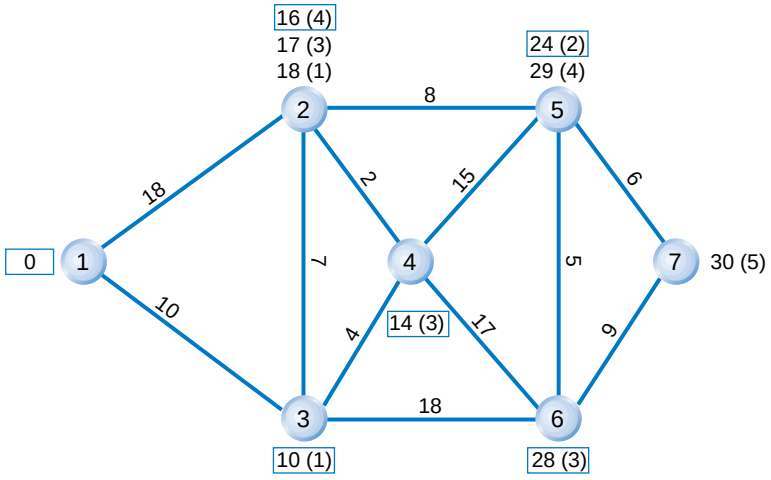

FIGURE CD5.15*f* Shortest Path Solution: Iteration 6

ITERATION 7

According to Figure CD5.15*f*:

Minimum Temporary Label (Made Permanent): 30 at N*ode* 7
Temporary Nodes Adjacent to Node 7: None

This gives Figure CD5.15*g*.

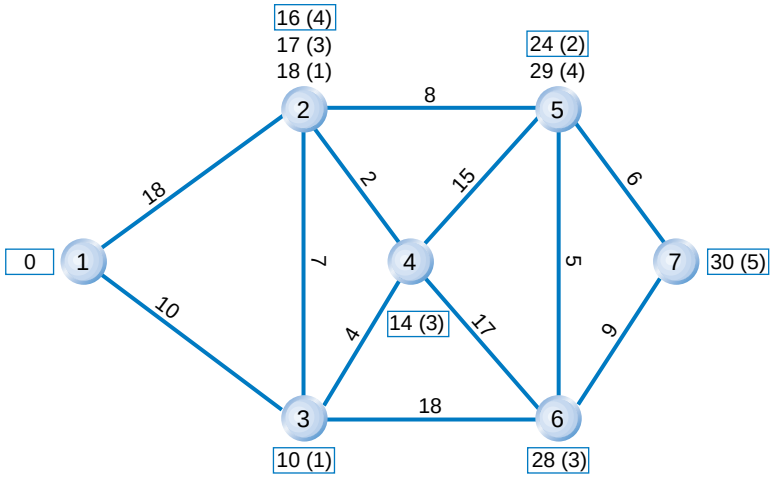

FIGURE CD5.15*g* Shortest Path Solution: Iteration 7

All nodes have now been assigned a permanent label, so we terminate the algorithm; the shortest distance is 30. Retracing the path from node 7, we see that we got to node 7 from node 5; to node 5 from node 2; to node 2 from node 4; to node 4 from node 3; and to node 3 from node 1. Thus, the shortest path is 1–3–4–2–5–7. The shortest distances and paths from node 1 to all other nodes in the network are as follows:

To	*Distance*	*Path*
Node 2	16	1–3–4–2
Node 3	10	1–3
Node 4	14	1–3–4
Node 5	24	1–3–4–2–5
Node 6	28	1–3–6
Node 7	30	1–3–4–2–5–7

VI THE MINIMAL SPANNING TREE PROBLEM—THE GREEDY ALGORITHM

In a minimal spanning tree problem, we seek the tree that interconnects all the nodes in a network at minimum total distance. The *Greedy algorithm* is an easy, efficient way to find this minimal spanning tree. The idea behind the Greedy algorithm is that we are, in fact, "greedy." We start by selecting the minimum arc out of a particular node (this can be node 1 or any node in the network) and then continue to build a tree by adding the lowest cost arcs that do not form cycles. The approach is formally stated as follows:

The Greedy Algorithm

Initialization Step
Select the minimum arc out of node 1 to start the tree.

Iterative Step
Add to the current tree the connecting arc of minimum distance that does not form a cycle. If all nodes are connected, STOP; we have the minimal spanning tree.

We repeat the above iterative step until all nodes have been connected.
If there are n nodes, the tree consists of n − 1 arcs.

In Figure CD5.16, we show in thick blue the minimal spanning tree for the same network as the one we used to illustrate the shortest path problem. It was generated by the iterations below.

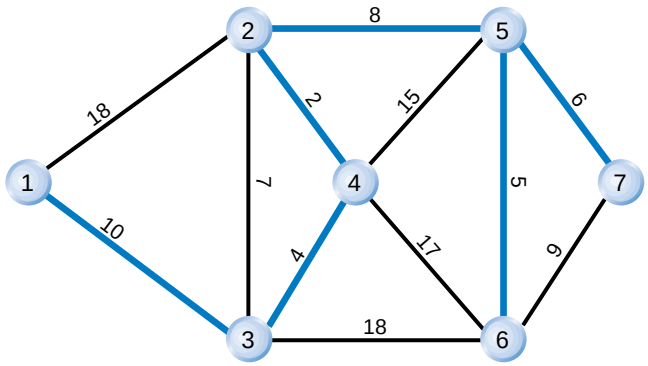

FIGURE CD5.16 Minimum Spanning Tree Solution

INITIALIZATION

Choose the minimum arc out of node 1—arc (1, 3)—distance 10.

Iteration	Minimum Distance Connecting Arc	Distance	Add Arc to Tree?	Cumulative Tree Distance
0	(1, 3)	10	Yes	10
1	(3, 4)	4	Yes	14
2	(2, 4)	2	Yes	16
3	(2, 3)	7	No—Cycle	
	(2, 5)	8	Yes	24
4	(5, 6)	5	Yes	29
5	(5, 7)	6	Yes	35

Thus, the minimum spanning tree has a total distance of 35 and consists of arcs (1, 3), (3, 4), (2, 4), (2, 5), (5, 6), and (5, 7).

VII The Maximal Flow Problem—The Maximal Flow Algorithm

In a maximal flow problem, we seek to find the maximum volume of flow from a source node to a terminal sink node in a capacitated network. Although it is a special case of

the capacitated transshipment problem and can be solved by the simplex method or the out-of-kilter algorithm, the following maximal flow algorithm is straightforward and easy to implement and avoids the necessity of keeping track of basic variables. In it, we determine if there is any path from the source to the sink that can carry flow. If there is, the flow is augmented as much as possible along this path, and the residual capacities of the arcs used on the path are reduced accordingly.

The only complication is that we might find that "if only we hadn't already committed to some flow along this arc, we could find an alternative path that could give a larger flow." Thus, when we *reduce* a residual arc capacity in one direction along an arc, we *increase* the residual capacity in the other direction on the arc (known as its *backwards capacity*) to account for a possible future "change of mind." As a result, the algorithm might determine an optimal solution in which five units are shipped from node i to node j and 3 units are shipped back from node j to node i. Of course, this means that only $5 - 3 = 2$ are shipped from node i to node j.

A formal statement of the maximal flow algorithm is as follows:

The Maximal Flow Algorithm

1. Find a path from the source to the sink that has positive residual capacities left on all arcs of this path. If no paths have positive flow, STOP; the maximal flow has been found.
2. Find the minimum residual capacity of the arcs on the path (call it k) and augment the flow on each arc by k.
3. Adjust the residual capacities of arcs on the path by *decreasing* the residual capacities of the arcs in the direction of the flow by k and *increasing* the residual capacities in the direction opposite the flow by k.

GO TO STEP 1.

Note that choosing which path to use in the network in step 1 is purely arbitrary. Often, however, it is not easy to determine if any path of positive flow exists from the source node to the sink node. One way to find such a path is to assign a "distance" of "1" to all arcs with positive residual capacities and $+\infty$ to the other arcs. We then use the shortest path algorithm to find the minimum distance from the source node to the terminal node; if this distance is less than $+\infty$, we have found a path with positive flow.

In the following example, however, we simply select the paths arbitrarily. In the network shown in Figure CD5.17 the numbers next to each node represent the residual capacity on the arc *from* the node *to* the adjacent node.

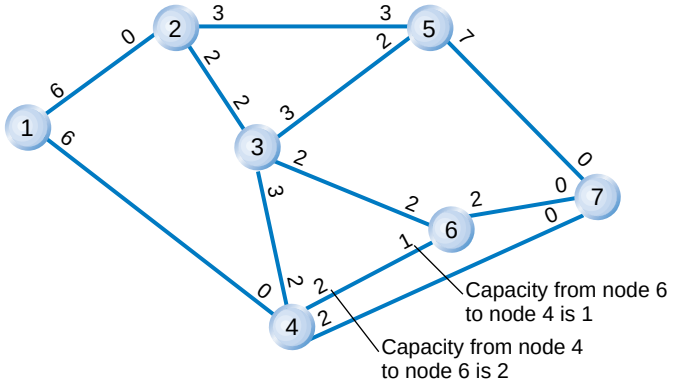

FIGURE CD5.17 Maximal Flow Problem

ITERATION 1

We have selected the path 1–2–5–7. The capacity on arc (1, 2) is 6; on arc (2, 5), 3; and on arc (5, 7), 7. Since the minimum of these is 3, we increase the flow along this path by 3, as shown in Figure CD5.18*a*. For iteration 2, we decrease the residual capacities along the arcs (1, 2), (2, 5), and (5, 7) by 3 to 3, 0, and 4, respectively, and increase the residual backwards capacities on arcs (2, 1), (5, 2), and (7, 5) by 3 to 3, 6, and 3, respectively.

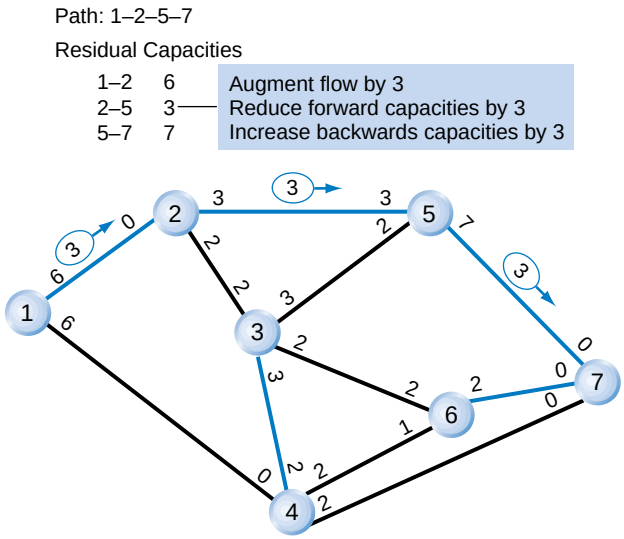

FIGURE CD5.18*a* Maximal Flow Solution: Iteration 1

ITERATION 2

From Figure CD5.18*b*, we see that no additional flow is possible along arc (2, 5). Thus, we must find a new path with positive residual capacities from node 1 to node 7. Here we select the path 1–2–3–6–7. The residual capacity on arc (1, 2) is 3; on arc (2, 3), 2; on arc (3, 6), 2; and on arc (6, 7), 2. Since the minimum of these is 2, we increase the flow along this path by 2. For iteration 3, we decrease the residual capacities along the arcs (1, 2), (2, 3), (3, 6), and (6, 7) by 2 to 1, 0, 0, and 0, respectively, and increase the residual backwards capacities on arcs (2, 1), (3, 2), (6, 3), and (7, 6) by 2 to 5, 4, 4, and 2, respectively.

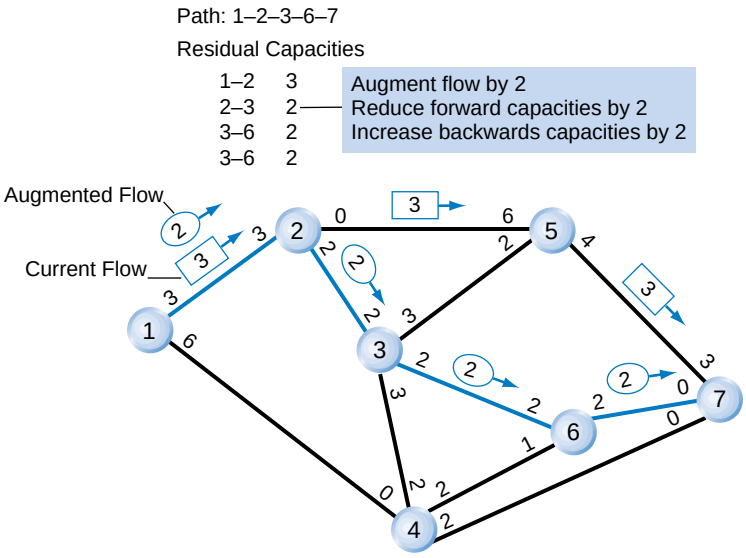

FIGURE CD5.18*b* Maximal Flow Solution: Iteration 2

ITERATION 3

In Figure CD5.18*c*, we see that because no additional flow is possible along arcs (2, 3) or (2, 5), no further flow is possible along arc (1, 2). Thus, we must find a path from node 1 to node 7 with positive residual capacities using arc (1, 4). Here we select 1–4–7. The residual capacity on arc (1, 4) is 6, and on arc (4, 7), 2. Since the minimum of these is 2, we increase the flow along this path by 2. For iteration 4, we decrease the residual capacities along the arcs (1, 4) and (4, 7) by 2 to 4 and 0, respectively, and increase the residual backwards capacities on arcs (4, 1) and (7, 4) by 2 to 2 and 2, respectively.

Path: 1–4–7

Residual Capacities

1–4	6
4–7	2

Augment flow by 2
Reduce forward capacities by 2
Increase backwards capacities by 2

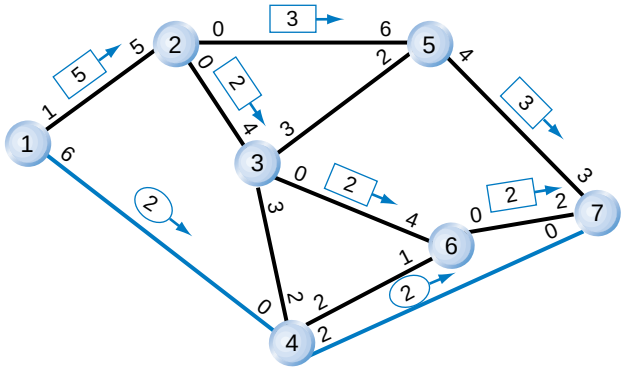

FIGURE CD5.18*c* Maximal Flow Solution: Iteration 3

ITERATION 4

In Figure CD5.18*d*, we see that no additional flow is possible along arc (4, 7); thus, we must find a new path with positive residual capacities from node 1 to node 7 using arc (1, 4). Here we select the path 1–4–3–5–7. The residual capacity on arc (1,4) is 4; on arc (4, 3), 2; on arc (3, 5), 3; and on arc (5, 7), 4. Since the minimum of these is 2, we increase the flow along this path by 2. For iteration 5, we decrease the residual capacities along the arcs (1, 4), (4, 3), (3, 5), and (5,7) by 2 to 2, 0, 1, and 2, respectively, and

Path: 1–4–3–5–7

Residual Capacities

1–4	4
4–3	2
3–5	3
5–7	4

Augment flow by 2
Reduce forward capacities by 2
Increase backwards capacities by 2

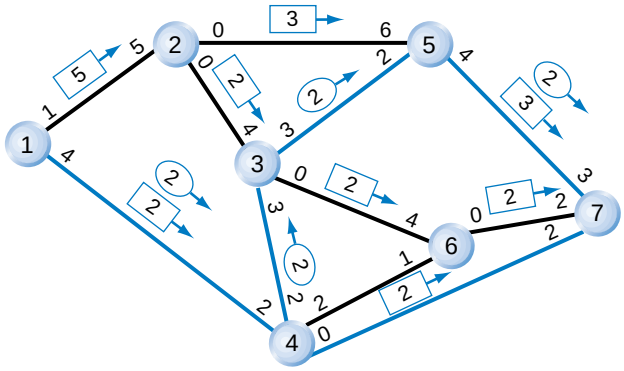

FIGURE CD5.18*d* Maximal Flow Solution: Iteration 4

increase the residual backwards capacities on arcs (4, 1), (3, 4), (5, 3), and (7,5) by 2 to 4, 5, 4, and 5, respectively.

ITERATION 5

In Figure CD5.18*e*, we see that because no additional flow is possible along arcs (4, 3) and (4, 7), any additional flow from arc (1, 4) must flow along arc (4, 6). A path with positive residual capacities from node 1 to node 7 using this arc is 1–4–6–3–5–7. The residual capacity on arc (1, 4) is 2; on arc (4, 6), 2; on arc (6, 3), 4; on arc (3, 5), 1; and on arc (5, 7), 2. Since the minimum of these is 1, we increase the flow along this path by 1. We decrease the residual capacities along the arcs (1, 4), (4, 6), (6, 3), (3, 5), and (5, 7) by 1 to 1, 1, 3, 0, and 1, respectively, and increase the residual backwards capacities on arcs (4, 1), (6, 4), (3, 6) (5, 3), and (7, 5) by 2 to 5, 2, 1, 5, and 6, respectively.

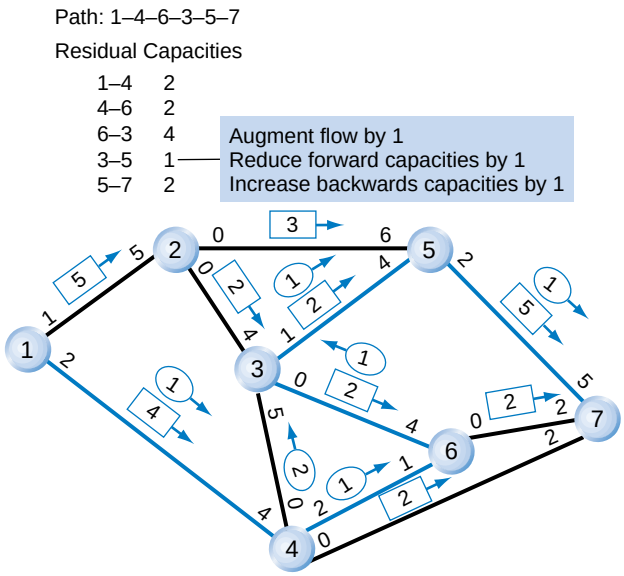

FIGURE CD5.18*e* Maximal Flow Solution: Iteration 5

ITERATION 6

As we see in Figure CD5.18*f*, no more flow is possible because there is no residual capacity left on the cut consisting of arcs (2, 5), (3, 5), (6, 7), and (4, 7). This is a minimal cut for the network. We also note that, at iteration 2, there is a flow of 2 from node 3 to node 6, while at iteration 5, there is a flow of 1 from node 6 to node 3. This means that the maximum flow has a net flow of 2 − 1 = 1 from node 3 to node 6.

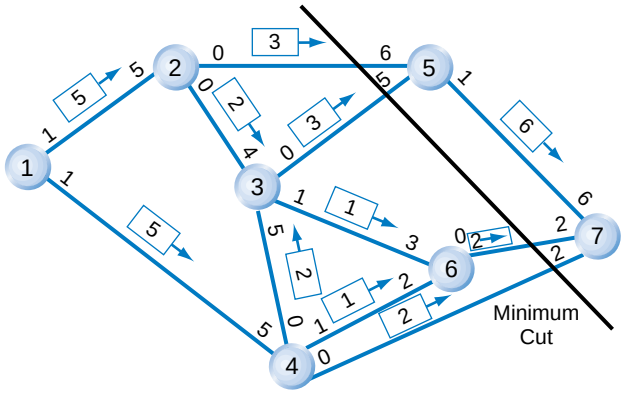

FIGURE CD5.18*f* Maximal Flow Solution: Final Network, No Additional Flow Possible

Thus, for this problem, we attain a maximum flow of 10 from node 1 to node 7 as follows:

From	*To*	*Flow*
1	2	5
1	4	5
2	3	2
2	5	3
3	5	3
3	6	1
4	3	2
4	6	1
4	7	2
5	7	6
6	7	2