A Labeling Algorithm for the Maximum-Flow Network Problem

Appendix C

Network-flow problems can be solved by several methods. In Chapter 8 we introduced this topic by exploring the special structure of network-flow problems and by applying the simplex method to this structure. This appendix continues the analysis of network problems by describing the application of the labeling algorithm to the maximum-flow network problem. Labeling methods provide an alternative approach for solving network problems. The basic idea behind the labeling procedure is to systematically attach labels to the nodes of a network until the optimum solution is reached.

Labeling techniques can be used to solve a wide variety of network problems, such as shortest-path problems, maximal-flow problems, general minimal-cost network-flow problems, and minimal spanning-tree problems. It is the purpose of this appendix to illustrate the general nature of the labeling algorithms by describing a labeling method for the maximum-flow problem.

C.1 THE MAXIMAL-FLOW PROBLEM

The maximal-flow problem was introduced in Section 8.2 of the text. If v denotes the amount of material sent from node s, called the source, to note t, called the sink, the problem can be formulated as follows:

Maximize v,

subject to:

$$\sum_j x_{ij} - \sum_k x_{ki} = \begin{cases} v & \text{if } i = s, \\ -v & \text{if } i = t, \\ 0 & \text{otherwise,} \end{cases}$$

$$0 \le x_{ij} \le u_{ij}, \qquad (i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, n).$$

We assume that there is no arc from t to s. Also, $u_{ij} = +\infty$ if arc i–j has unlimited capacity. The interpretation is that v units are supplied at s and consumed at t.

Letting x_{ts} denote the variable v and rearranging, we see that the problem assumes the following special form of the general network problem:

Maximize $x_{ts'}$

subject to:

$$\sum_j x_{ij} - \sum_k x_{ki} = 0 \qquad (i = 1, 2, \ldots, n)$$

$$0 \le x_{ij} \le u_{ij} \qquad (i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, n).$$

Here arc $t - s$ has been introduced into the network with u_{ts} defined to be $+\infty$, x_{ts} simply returns the v units from node t back to node s, so that there is no formal external supply of material. Let us recall the example (see Fig. C.1) that was posed in Chapter 8 in terms of a water-pipeline system. The numbers above the arcs indicate flow capacity and the bold-faced numbers below the arcs specify a tentative flow plan.

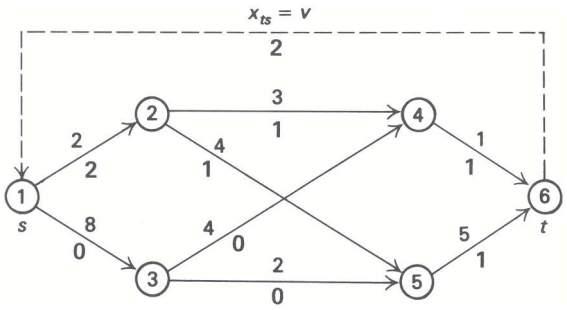
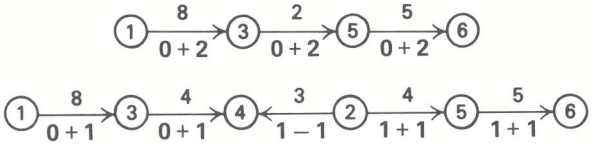


Figure C.1

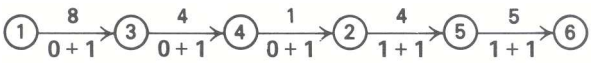
The algorithm for finding maximal flow rests on observing two ways to improve the flow in this example. The following two ‘‘paths’’ appear in Fig. C.1



In the first case, the directed path 1–3–6 has the capacity to carry 2 additional units from the source to the sink, as given by the capacity of its weakest link, arc 3–5. Note that adding this flow gives a feasible flow pattern, since 2 units are added as input as well as output to both of nodes 3 and 5.

The second case is not a directed path from the source to the sink since arc 2–4 appears with the wrong orientation. Note, however, that adding one unit of flow to the ‘‘forward arcs’’ from 1 to 6 and subtracting one unit from the ‘‘reverse arc’’ 2-4 provides a feasible solution, with increased source-to-sink flow. Mass balance is maintained at node 4, since the one more unit sent from node 3 cancels with the one less unit sent from node 2. Similarly, at node 2 the one additional unit sent to node 5 cancels with the one less unit sent to node 4.

The second case is conceptually equivalent to the first if we view decreasing the flow along arc 2-4 as sending flow from node 4 back to node 2 along the reverse arc 4-2. That is, the unit of flow from 2 to 4 increases the *effective capacity* of the ‘‘return’’ arc 4-2 from 0 in the original network to 1. At the same time, it decreases the usable or effective capacity along arc 2-4 from 3 to 2. With this view, the second case becomes:

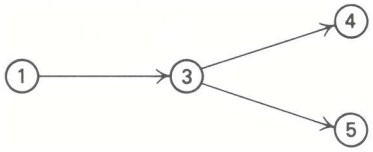


Now both instances have determined a directed *flow-carrying path* from source to sink, that is, a directed path with the capacity to carry additional flow.

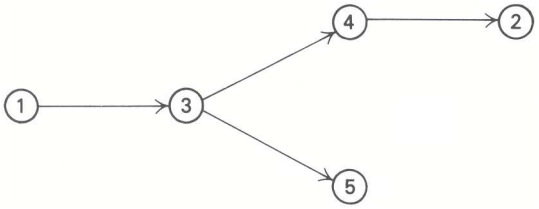
The maximal-flow algorithm inserts return, arcs, such as 4–2 here, and searches for flow-carrying paths. It utilizes a procedure common to network algorithms by ‘‘fanning our’’ from the source node, constructing flow-carrying paths to other nodes, until the sink node is reached.

The procedure starts by labeling all nodes with the capacity to receive flow from the source node by a single arc. For the pipeline example, arc 1–2 is saturated and has a zero effective capacity, whereas are 1–3 can carry 8 additional units. thus, only node 3 is labeled from the source

The procedure is repeated by labeling all nodes with the capacity to receive flow directly from node 3 by a single arc. In this case, nodes 4 and 5 are labeled and the labeled nodes become:

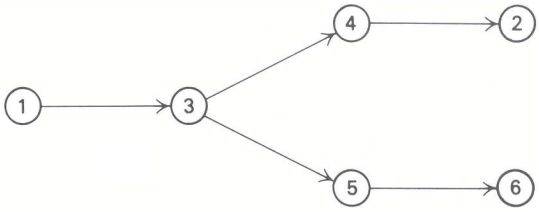


Additional nodes can now be labeled from either node 4 or 5. If node 4 is selected next, then node 2 is labeled, since the effective capacity (return capacity) of arc 4-2 is positive (node 6 cannot be labeled from node 4 since arc 4-6 is saturated). The labeled nodes are:



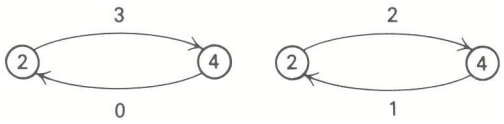
At any point in the algorithm, some of the labeled nodes, here nodes 1, 3 and 4, will already have been used to label additional nodes. We will say that these nodes have been *scanned.* Any unscanned labeled node can be selected and scanned next.

Scanning node 2 produces no additional labelings, since node 2 can only send flow directly to nodes 1, 4, and 5, and these have been labeled previously. Node 6 is labeled when node 5 is scanned, and the labeled nodes are:



A flow-carrying path has been identified, since the sink has been labeled. In this case, the path discovered is 1–3–5–6, which was the first path given above. The flow along this path is updated by increasing the flow along each arc in the path by the flow capacity of the path, here 2 units. Effective capacities are now recomputed and the labeling procedure is repeated, starting with only the source node labeled.

Following is a formal description of the algorithm and a solution of the water-pipeline example. Notice that the flow value on the arcs need not be recorded from step to step in the algorithm since this information is computed easily from the effective (or usable) capacities of the arcs and their return arcs. For the example above, the effective capacity on arc 2–4 is less than its initial capacity.



The difference $3 = 2 - 1$ must be the flow value on that arc that resulted in the reduced capacity. The effective capacity on arc 4–2 has been increased from 0 to 1 by adding return-flow capacity to that arc, not

by physically adding flow to that arc. In general, this is the case whenever effective capacity exceeds the original capacity. These arcs, consequently, carry no flow.

When a flow-carrying path has been found from source to terminal, that is able to carry θ additional units, the effective capacity in every arc i–j of that path is reduced by θ. At the same time, the effective capacity of each reverse arc j–i increases by θ, since they can now be used to divert (or back up) these units of flow.

C.2 MAXIMAL-FLOW ALGORITHM—FORMAL STATEMENT

Initialization

Assume a given feasible flow plan x_{ij} (if none is given, use the feasible plan with all $x_{ij} = 0$). The initial effective capacity u^*_{ij} on arc i–j is given by calculating $u^*_{ij} = u_{ij} - x_{ij} + x_{ji}$ (i.e., unused capacity $u_{ij} - x_{ij}$ plus return capacity x_{ji}).

Path Search

Start with the source node s and label (mark) every node k with $u^*_{sk} > 0$. Then, in turn, select any labeled node i not already scanned (i.e., used to label other nodes) and label every node j with $u^*_{ij} > 0$ until either t has been labeled or no further labeling can take place.

Capacity Update

If t has been labeled, then a flow-carrying path P has been found from source to sink ($u^*_{ij} > 0$ for every arc i–j on the path), and

$$\theta = \text{Min } \{u^*_{ij} \mid i\text{–}j \text{ in } P\}$$

is the flow capacity of the path. For every arc i–j of P, change u^*_{ij} to $u^*_{ij} - \theta$, and change u^*_{ji} to $u^*_{ji} + \theta$; i.e., increase the effective capacity of the return path. (Adding or subtracting finite θ to any $u^*_{ij} = +\infty$ keeps the u^*_{ij} at $+\infty$. If every u^*_{ij} in P is $+\infty$, then $\theta = +\infty$ and the optimal flow is infinite.)

Termination

If the path search ends without labeling t, then terminate. The optimal flow pattern is given by:

$$x_{ij} = \begin{cases} u_{ij} - u^*_{ij} & \text{if } u^*_{ij} > u^*_{ij}, \\ 0 & \text{if } u_{ij} \le u^*_{ij}. \end{cases}$$

C.3 SAMPLE SOLUTION

Figure C.2 solves the water-pipeline example in Fig. C.1 by this algorithm. Checks next to the rows indicate that the node corresponding to that row has already been scanned. The first three tableaus specify the first application of the path-search step in detail. In Tableau 1, node 1 has been used to label node 3. In Tableau 2, node 3 was used to label nodes 4 and 5 (since $u^*_{34} > 0$, $u^*_{35} > 0$). At this point either node 4 or 5 can be used next for labeling. The choice is arbitrary, and in Tableau 3, node 5 has been used to label node 6. Since 6 is the sink, flow is updated. The last column in the tableau keeps track of how nodes have been labeled.[†] By backtracking, we get a flow-carrying path P from source to terminal. For instance, from Tableau 3, we know that 6 has been labeled from 5, 5 form 3, and 3 from 1, so that the path is 1–3–5–6.

[†] For computer implementation, another column might be maintained in the tableau to keep track of the capacity of the flow-carrying paths as they are extended. In this way, θ need not be calculated at the end.

Initial capacity u_{ij}

		1	2	3	4	5	6
Source	1		2	8			
	2				3	4	
	3				4	2	
	4						1
	5						5
Destination	6						

Tableau 1

		1	2	3	4	5	6	*Labeled from*	
Source	1			8				*Start*	✓
	2	2			2	3			
	3				4	2		1	
	4		1						
	5		1				4		
Destination	6				1	1			

Tableau 2

		1	2	3	4	5	6	*Labeled from*	
Source	1			8				*Start*	✓
	2	2			2	3			
	3				4	2		1	✓
	4		1					3	
	5		1				4	3	
Destination	6				1	1			

Figure C.2 Tableaus for the maximal-flow algorithm.

Tableau 3

	1	2	3	4	5	6	*Labeled from*		*Path P*
1			8				*Start*	✓	①
2	2			2	3				↓
3				4	2	1	1	✓	③
4		1				3	3		↓
5		1				4	3	✓	⑤
6				1	1		5		↓ ⑥

$\theta = \underset{i-j \text{ in } P}{\text{Min}} \{u_{ij}^*\} = \text{Min}\{8, 2, 4\} = 2$

Subtract 2 from u_{13}^*, u_{35}^*, and u_{56}^*.

Add 2 to u_{31}^*, u_{53}^*, and u_{65}^*.

Path 1–3–5–6

Tableau 4

	1	2	3	4	5	6	*Labeled from*		*Path P*
1			6				*Start*	✓	
2	2			2	3		4	✓	
3	2			4			1	✓	①
4		1					3	✓	② ↓
5		1	2			2	2	✓	③ ↖ ④
6				1	3		5		⑤ ↓ ⑥

$\theta = \underset{i-j \text{ in } P}{\text{Min}} \{u_{ij}^*\} = \text{Min}\{6,4,1,5,2\} = 1$

Subtract 1 from u_{13}^*, u_{34}^*, u_{42}^*, u_{25}^*, u_{56}^*.

Add 1 to u_{31}^*, u_{43}^*, u_{24}^*, u_{52}^*, u_{65}^*.

Path 1–3–4–2–5–6

Tableau 5

	1	2	3	4	5	6	*Labeled from*	
1			5				*Start*	✓
2	2			3	2			
3	3			3			1	✓
4			1				3	✓
5		2	2			2		
6				1	4			

Key: Entry in ith row and jth column of each tableau is u_{ij}^*. Blank entries denote zeros.

Figure C.2 (*Cont.*)

Tableau 4 contains the updated capacities and a summary of the next path search, which used nodes 1, 3, 4, 2, and 5 for labeling. The fifth tableau contains the final updated capacities and path search. Only nodes 1, 3, and 4 can be labeled in this tableau, so the algorithm is completed.

The flow at any point in the algorithm is obtained by subtracting effective capacities from initial capacities, $u_{ij} - u^*_{ij}$, and discarding negative numbers. For Tableaus 1, 2, or 3, the flow is given by Fig. C.1. After making the two indicated flow changes to obtain Tableau 4 and then Tableau 5, the solutions are given in two networks shown in Figs. C.3 and C.4. The optimal solution sends two units along each of the paths 1–2–5–6 and 1–3–5–6 and one unit along 1–3–4–6.

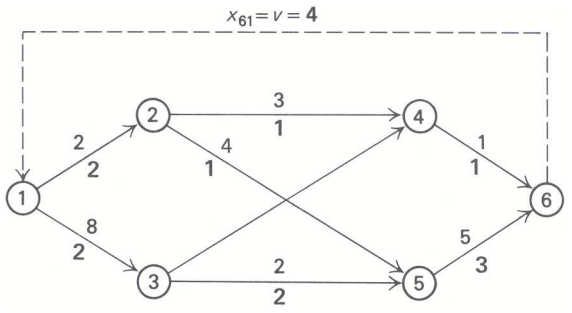


Figure C.3

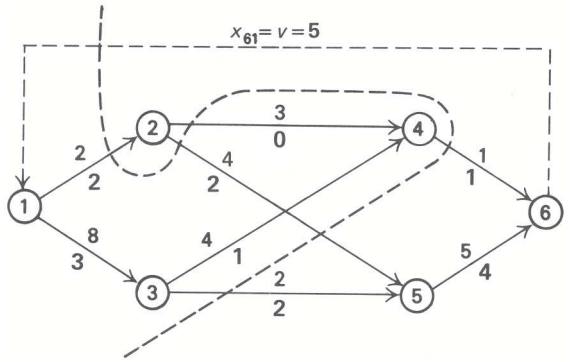


Figure C.4

C.4 VERIFYING THE ALGORITHM—MAX-FLOW/MIN-CUT

The final solution to the sample problem shown in Fig. C.4 illustrates an important conceptual feature of the maximum-flow problem. The heavy dashed line in that figure ‘‘cuts" the network in two, in the sense that it divides the nodes into two groups, one group containing the source node and one group containing the sink node. Note that the most that can ever be sent across this cut to the sink is two units from node 1 to node 2, one unit from 4 to 6, and two units from 3 to 5, for a total of 5 units (arc 2–5 connects nodes that are both to the right of the cut and is not counted). Since no flow pattern can ever send more than these 5 units and the final solution achieves this value, it must be optimal.

Similarly, the cut separating labeled from unlabeled nodes when the algorithm terminates (see Fig. C.5) shows that the final solution will be optimal.

By conservation of mass, v equals the net flow from left to right across this cut, so that v can be no greater than the total capacity of all arcs $i-j$ pictured. In fact, this observation applies to any cut separating

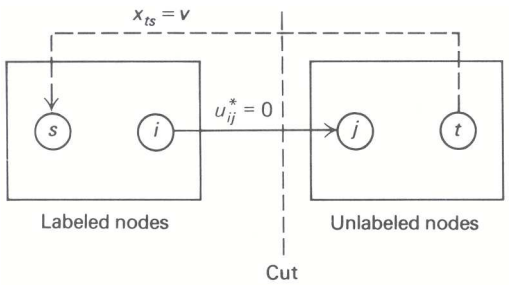


Figure C.5

the source node s from the sink node t.

$$\begin{bmatrix} \text{Any} \\ \text{feasible flow} \\ v \text{ from} \\ \text{node } s \text{ to} \\ \text{node } t \end{bmatrix} \leq \begin{pmatrix} \text{Capacity of} \\ \text{any cut separating} \\ \text{node } s \text{ and node } t \end{pmatrix}.$$

As in the example discussed above, the capacity of any cut is defined as the total capacity of all arcs directed ‘‘from left to right" across the cut.

By virtue of the labeling scheme, however, every arc i–j directed from left to right across the cut separating the labeled and unlabeled nodes must have $u_{ij}^* = 0$; otherwise j would have been labeled from i. This observation implies that

$$\boxed{x_{ij} = u_{ij} \quad \text{and} \quad x_{ij} = 0,}$$

since if either $x_{ij} < u_{ij}$ or $x_{ji} > 0$, then $u_{ij}^* = (u_{ij} - x_{ij}) + x_{ji}$ would be positive. Consequently, the net flow across the cut, and thus v, equals the cut capacity. Since no flow can do any better, this flow pattern is optimal.

We can recast this observation in a slightly different way. The inequality stated above shows that any feasible flow v form source to sink is bounded from above by the capacity of any cut. Consequently, the maximum flow from source to sink is bounded by the capacity of *any* cut separating the source and sink. In particular, the maximum flow from source to sink is bounded from above by the minimum capacity of any cut. We have just shown, though, that the maximum flow v equals the capacity of the cut separating the labeled and unlabeled nodes when the algorithm terminates; therefore, we have verified the famous *max-flow/min-cut theorem*.

$$\begin{pmatrix} \text{The maximum flow} \\ v \text{ from node } s \text{ to} \\ \text{node } t \end{pmatrix} = \begin{pmatrix} \text{The minimum capacity} \\ \text{of any cut separating} \\ \text{node } s \text{ and node } t \end{pmatrix}.$$

There are many important ramifications of this theorem. It can be used, for example, to establish elegant results in combinatorial theory. Although such results lead to many useful applications of combinatorial analysis, their development is beyond the scope of our coverage here. We have merely shown how the max-flow/min-cut theorem arises, and used the theorem to show that when the maximum flow-labeling algorithm terminates, it has found the maximum possible flow from the source node to the sink node.

Finally, we should note that the maximum-flow procedure eventually terminates, as in the above example, and does not continue to find flow-carrying paths from s to t forever. For certain cases this is easy to show. If the capacity data u_{ij} is all integral and every x_{ij} in the original flow plan is an integer, then every u_{ij}^* and θ encountered will be integral. But then, if the maximal flow v is not $+\infty$, we approach it in integral steps and must eventually reach it. Similarly, if the original data and flow is fractional, v increases at each step by at least a given fractional amount and the procedure terminates in a finite number of steps. If the data is

irrational, then it is known that the algorithm finds the maximum flow in a finite number of steps as long as the nodes are considered during path search on a first-labeled–first-scanned basis.

Note that starting with an integral flow plan x_{ij} when all u_{ij} are integral leads to integral u^*_{ij}. Consequently, the final (and optimal) flow plan will be integral. This is a special instance of the network integrality property introduced in Chapter 8.