Chapter 3

Paths, Trees, and Cycles

3.1 Introduction

Graphs and networks arise everywhere, so several disciplines have contributed important ideas to the evolution of network flows. The main drawback of this is that the literature on networks and graph theory lacks unity and uniform conventions, notation, and terminology.

In this chapter, we have three objectives. First, we bring together many basic definitions of network flows and graph theory, and we set the notation we will use. Second, we introduce several different data structures used to represent networks within a computer and discuss the relative advantages and disadvantages. Third, we discuss a number of different ways to transform a network flow problem and obtain an equivalent model.

3.2 Notation and Definitions

In this section, we give several basic definitions from graph theory and present some basic notation. We also state some elementary properties of graphs.

Directed Graph A *directed graph* (or directed network) $G = (N, A)$ consists of a set N of nodes (or vertexes) and a set A of arcs whose elements are ordered pairs of distinct nodes. Figure 3.1 gives an example of a directed graph, where $N = \{1, 2, 3, 4, 5, 6, 7\}$ and $A = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 6), (4, 5), (4, 7), (5, 2), (5, 3), (5, 7), (6, 7)\}$. An arc (i, j) behaves like a one-way street and permits flow from node i to node j only, but not vice versa. Some numerical values (i.e., costs, capacities, etc) are typically associated with the arcs. We let n denote the number of nodes (i.e., $n = |N|$) and m denote the number of arcs (i.e., $m = |A|$) in G

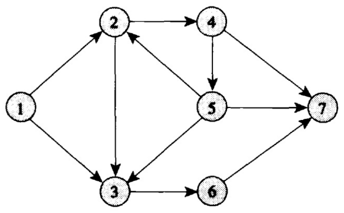


Figure 3.1: Directed graph

Undirected Graph An *undirected graph* $G = (N, E)$ consists of a set N of n nodes (or vertexes) and a set E of m edges whose elements are unordered pairs of distinct nodes. Figure 3.2 gives an example of an undirected graph. In an undirected graph, we can refer to an edge joining the node pair i and j as either $\{i, j\}$ or $\{j, i\}$. An edge $\{i, j\}$ can be regarded as a two-way street with flow permitted in both directions

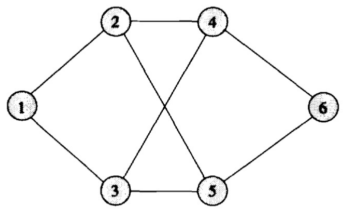


Figure 3.2: Undirected graph

Tail and Head An arc (i, j) has two *endpoints* i and j: i is the *tail*, and j is the *head*. We say that the arc (i, j) *emanates* from node i and *terminates* at node j. An arc (i, j) is *incident* to nodes i and j. The arc (i, j) is an *outgoing arc* of node i and an *incoming arc* of node j. Whenever an arc $(i, j) \in A$, we say that node j is *adjacent* to node i

Degree The *indegree* of a node is the number of its incoming arcs and its *outdegree* is the number of its outgoing arcs. The *degree* of a node is the sum of its indegree and outdegree. For example, in Figure 3.1, node 3 has an indegree of 3, an outdegree of 1, and a degree of 4. Notice that the sum of indegrees of all nodes equals the sum of outdegrees of all nodes and both are equal to m

Adjacency List The *arc adjacency list* $A(i)$ of a node i is the set of arcs emanating from i, i.e., $A(i) = \{(i, j) \in A \mid j \in N\}$. The *node adjacency list* $A(i)$ is the set of nodes adjacent to i; in this case, $A(i) = \{j \in N \mid (i, j) \in A\}$. We will often omit the terms "arc" and "node" and simply refer to the adjacency list; in all cases, it will be clear from context whether we mean arc adjacency list or node adjacency list. Notice that $|A(i)|$ equals the outdegree of i. Since the sum of all node outdegrees equals m, we obtain the following property:

Property 3.1. $\sum_{i \in N} |A(i)| = m$

Subgraph A graph $G' = (N', A')$ is a *subgraph* of $G = (N, A)$ if $N' \subseteq N$ and $A' \subseteq A$. We say that $G' = (N', A')$ is the subgraph of G *induced* by N' if A' contains each arc of A with both endpoints in N'

Walk A *walk* in a directed graph $G = (N, A)$ is a subgraph of G consisting of a sequence of nodes $i_1 - i_2 - \ldots - i_r$ satisfying the property that for $k = 1, \ldots, r-1$ either $(i_k, i_{k+1}) \in A$ or $(i_{k+1}, i_k) \in A$. A walk in the graph of Figure 3.1 is $1 - 2 - 5 - 7$. A *directed walk* is a walk such that $(i_k, i_{k+1}) \in A$ for $k = 1, \ldots, r-1$. A directed walk in the graph of Figure 3.1 is $1 - 2 - 4 - 5 - 2 - 3$

Path A *path* is a walk without any repetition of nodes, e.g., $1 - 2 - 5 - 7$ is a path of the graph of Figure 3.1. A *directed path* is a directed walk without any repetition of nodes

Cycle A *cycle* is a path $i_1 - i_2 - \ldots - i_r$ such that $i_1 = i_r$. A *directed cycle* is a directed path such that the first and the last nodes coincide

Acyclic Graph A graph is a *acyclic* if it does not contain any directed cycles

Connectivity Two nodes i and j are connected if the graph contains at least one path from i to j. A graph is *connected* if every pair of its nodes is connected; otherwise, the graph is *disconnected*

Cut A *cut* is a partition of the node set N into two parts, S and $\overline{S} = N \setminus S$. Each cut defines a *cutset* consisting of those arcs that have one endpoint in S and another endpoint in $\overline{S}$. We refer to the cutset defined by S and $\overline{S}$ as $A(S, \overline{S})$. Figure 3.3 shows the cutset $A(S, \overline{S}) = \{(2,4), (3,6), (5,2), (5,3)\}$ with $S = \{1,2,3\}$ and $\overline{S} = \{4,5,6,7\}$

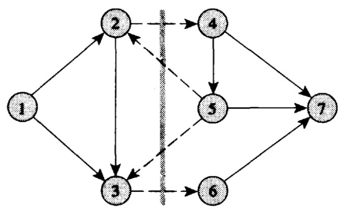


Figure 3.3: Cutset

s-t Cut An *s-t cut* is a cut defined with respect to two distinguished nodes s and t such that $s \in S$ and $t \in \overline{S}$. For instance, if $s = 1$ and $t = 6$, the cut depicted in Figure 3.3 is an *s-t cut*

Tree A *tree* is a connected graph that contains no cycle. In the graph of Figure 3.1, the arcs $\{(1,2), (1,3), (3,6)\}$ represent a tree.

Property 3.2. *Trees have the following properties*

1. *A tree on k nodes contains exactly $k - 1$ arcs*
2. *Every two nodes of a tree are connected by a unique path*

Spanning Tree A tree T is a *spanning tree* of G if T is a spanning subgraph of G. Figure 3.4 shows two spanning trees of the graph shown in Figure 3.1. Every spanning tree of a connected n-node graph G has $(n - 1)$ arcs. We refer to the arcs belonging to a spanning tree T as *tree arcs* and arcs not belonging to T as *non-tree arcs*

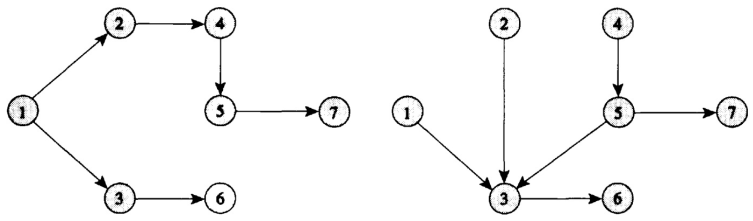


Figure 3.4: Spanning trees

Exercise 3.1.

The first paper on graph theory was written by Leonhard Euler in 1736. He started with the following mathematical puzzle:

The city of Konigsberg has seven bridges, arranged as shown in Figure 3.5. Is it possible to start at some place in the city and cross every bridge exactly once?

Either specify such a tour or prove that it is impossible to do so.

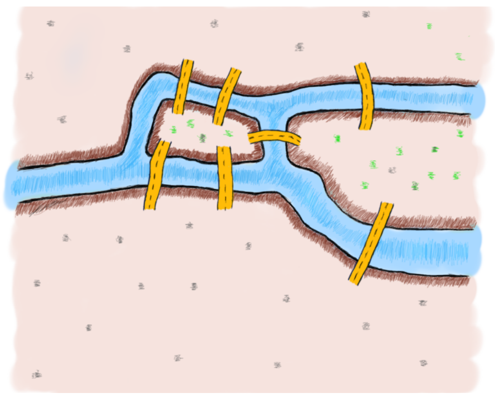


Figure 3.5: The Seven Bridges of Konigsberg

Euler proved that such a tour cannot exist. Indeed, he observed that an odd-degree node must be at the beginning and at the end of the trip. Since there can only be one beginning and one end, there can only be two odd-degree nodes. Since the bridge problem has four odd-degree nodes, it is impossible to do so!

Exercise 3.2.

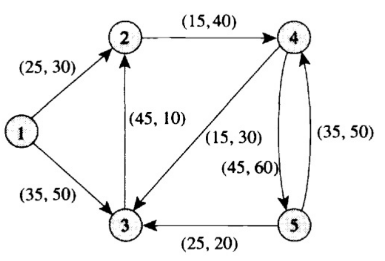
At the beginning of a dinner party, several participants shake hands with each other. Show that the participants that shook hands an odd number of times must be even in number

Construct an undirected graph with a node corresponding to each participant and an edge between two nodes if and only if the corresponding participants shook hands with one another. The number of people shaking hands an odd number of times equals the number of nodes having odd degree. Let δ_i denote the degree of node i. Notice that $\varphi = \sum_{i\in N} \delta_i$ is even because each edge contributes 2 to φ. Hence, the number of nodes having odd degrees is even; otherwise, φ would not be even.

3.3 Graph Representations

3.3.1 Node-Arc Incidence Matrix

The *node-arc incidence matrix* representation, or simply the *incidence matrix* representation, represents a graph $G = (N, A)$ as the constraint matrix of the MCFP problem that we discussed in Section 1.2. This representation stores the graph as an $n \times m$ matrix $\mathcal{N}$ that contains one row for each node and one column for each arc. The column corresponding to arc $(i, j) \in A$ has only two nonzero elements: a $+1$ in the row corresponding to node i and a -1 in the row corresponding to node j. The right part of Figure 3.6 gives this representation for the graph shown in the left part of the figure. The



$$\begin{array}{c|cccccccc} & (1,2) & (1,3) & (2,4) & (3,2) & (4,3) & (4,5) & (5,3) & (5,4) \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 3 & 0 & -1 & 0 & 1 & -1 & 0 & -1 & 0 \\ 4 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & -1 \\ 5 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 \end{array}$$

Figure 3.6: Incidence matrix representation

incidence matrix has a special structure: only $2m$ out of its nm entries are nonzero; all of its nonzero entries are $+1$ or -1; each column has exactly one $+1$ and one -1; the number of $+1$s in a row equals the outdegree of the corresponding node; the number of -1s in the row equals the indegree of the node. The incidence matrix contains few nonzero coefficients, so it is not space-efficient. However, it represents the constraint matrix of the MCFP and possesses several interesting theoretical properties (we will study them).

Additional information, such as arc costs and capacities, can be stored in m-dimensional arrays.

3.3.2 Node-Node Adjacency Matrix

The *node-node adjacency matrix* representation, or simply the *adjacency matrix* representation, stores the graph as an $n \times n$ matrix $\mathcal{H} = \{h_{ij}\}$. The matrix has a row and a column corresponding to every node, and its ijth entry h_{ij} equals 1 if $(i, j) \in A$ and equals 0 otherwise. Figure 3.7 specifies this representation for the graph of Figure 3.6. If we wish to store arc costs and capacities as well as the network topology, we can store this information in two additional $n \times n$ matrices. The adjacency matrix has n^2 elements, only m of which are nonzero. Consequently, this representation is space-efficient only if the graph is dense.

3.3.3 Adjacency Lists

The *adjacency list* representation stores the node adjacency list of each node as a linked list. A linked list is a collection of cells each containing one or more fields. The node adjacency list for node i is a linked list having $|A(i)|$ cells and each cell corresponds to an arc $(i, j) \in A$. The cell corresponding to the arc (i, j) has as many fields as the amount of information we wish to store. One data field

$$
\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 1 & 0 & 0 & 0 \\
4 & 0 & 0 & 1 & 0 & 1 \\
5 & 0 & 0 & 1 & 1 & 0
\end{array}
$$

Figure 3.7: Adjacency matrix representation

stores node j. We might use two other data fields to store the arc cost c_{ij} and the arc capacity u_{ij}. Each cell contains one additional field, called the *link*, which stores a pointer to the next cell in the adjacency list. If a cell happens to be the last cell in the adjacency list, by convention we set its link to value zero. Since we need to store and access n linked lists, one for each node, we also need an array of pointers that point to the first cell in each linked list. We accomplish this by defining an n-dimensional array, $first$, whose element $first(i)$ stores a pointer to the first cell in the adjacency list of node i. If the adjacency list of node i is empty, we set $first(i) = 0$. Figure 3.8 specifies the adjacency list representation of the graph shown in Figure 3.6.

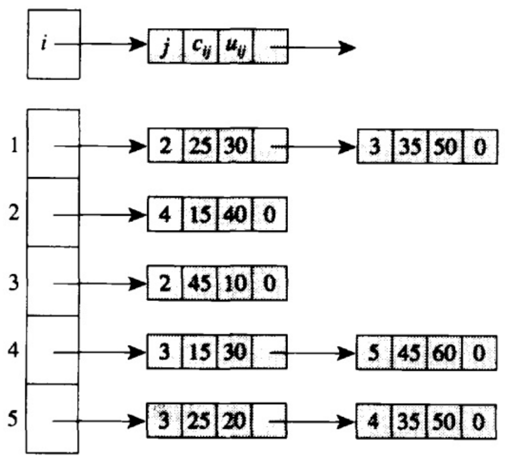
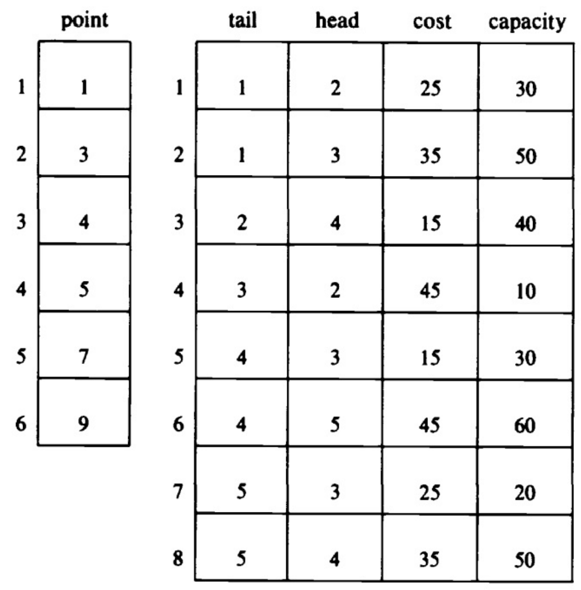


Figure 3.8: Adjacency list representation

3.3.4 Forward-Star Representation

The *forward-star* representation of a graph is similar to the adjacency list representation because it stores the node adjacency list of each node. However, instead of maintaining these lists as linked lists, it stores them in a single array. To develop this representation, we first associate a unique sequence number with each arc, thus defining an ordering of the arc list. We number the arcs in a specific order: first those emanating from node 1, then those emanating from node 2, and so on. We number the arcs emanating from the same node in an arbitrary fashion. We then sequentially store information about each arc (i.e., tail, head, cost, capacity, etc) in the arc list in different arrays. We also maintain a pointer with each node i, denoted by $point(i)$, that indicates the smallest-numbered arc in the arc list that emanates from node i – if node i has no outgoing arcs, we set $point(i) = point(i+1)$. Therefore, the forward-star representation stores the outgoing arcs of node i at positions $point(i)$ to $(point(i+1)-1)$ in the arc list. If $point(i) = point(i+1)$, node i has no outgoing arc. We also set $point(1) = 1$ and $point(n+1) = m+1$. Figure 3.9 specifies the forward -star representation of the graph given in Figure 3.6 - notice that the array $tail(i)$ may also be removed.



	point
1	1
2	3
3	4
4	5
5	7
6	9

	tail	head	cost	capacity
1	1	2	25	30
2	1	3	35	50
3	2	4	15	40
4	3	2	45	10
5	4	3	15	30
6	4	5	45	60
7	5	3	25	20
8	5	4	35	50

Figure 3.9: Forward-star representation

Exercise 3.3.

Consider the graph of Figure 3.10. Specify (1) the incidence matrix, (2) the adjacency matrix, and (3) the forward-star representation.

1. Incidence matrix

	Arc					
Node	*(1,2)*	*(1,3)*	*(2,4)*	*(3,2)*	*(3,5)*	*(5,4)*
1	1	1	0	0	0	0
2	-1	0	1	-1	0	0
3	0	-1	0	1	1	0
4	0	0	-1	0	0	-1
5	0	0	0	0	-1	1

2. Adjacency matrix

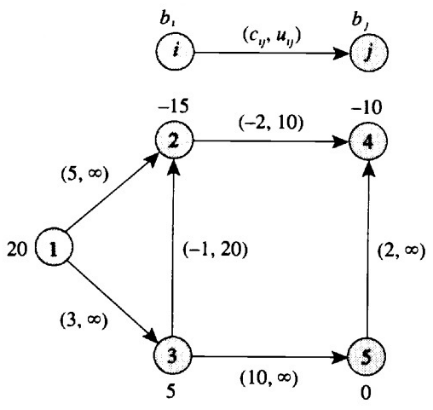


Figure 3.10: Directed graph

Node	*1*	*2*	*3*	*4*	*5*
1	0	1	1	0	0
2	0	0	0	1	0
3	0	1	0	0	1
4	0	0	0	0	0
5	0	0	0	1	0

3. Forward star representation

point					
1	*2*	*3*	*4*	*5*	*6*
1	3	4	6	6	7

tail	*head*	*cost*	*capacity*	*index*
1	2	5	∞	1
1	3	3	∞	2
2	4	-2	10	3
3	2	-1	20	4
3	5	10	∞	5
5	4	2	∞	6

3.4 Graph Transformations

Frequently, we require graph transformations to simplify a graph, show equivalences between different problems, or state a network problem in a standard form required by some software. We describe some of these important transformations by assuming that the network problem is a MCFP formulated with model (1.1).

3.4.1 Edges to Arcs

If the MCFP contains an edge $\{i, j\}$ with cost $c_{ij} \geq 0$ and capacity u_{ij}, model (1.1) needs two decision variables x_{ij} and x_{ji} to represent the flow along edge $\{i, j\}$ in the two directions. The objective function (1.1a) features the term $c_{ij}x_{ij} + c_{ij}x_{ji}$. Since $c_{ij} \geq 0$, it does not exist any optimal solutions where both $x_{ij} > 0$ and $x_{ji} > 0$. Notice that this transformation is correct only if the minimum flow required through edge $\{i, j\}$ is 0 (i.e., $l_{ij} = 0$) and cost c_{ij} is non-negative.

3.4.2 Removing Nonzero Lower Bounds

If an arc (i, j) has a nonzero lower bound l_{ij} on the arc flow x_{ij}, we replace x_{ij} by $x'_{ij} + l_{ij}$ in the problem formulation, where x'_{ij} represents the incremental flow beyond l_{ij}. The flow bound constraint becomes $l_{ij} \leq x'_{ij} + l_{ij} \leq u_{ij}$, or $0 \leq x'_{ij} \leq (u_{ij} - l_{ij})$. Making this substitution in the mass balance constraints (1.1b) decreases b_i by l_{ij} units and increases b_j by l_{ij} units. This substitution changes the objective function value by a constant $c_{ij}l_{ij}$. Figure 3.11 illustrates this transformation graphically.

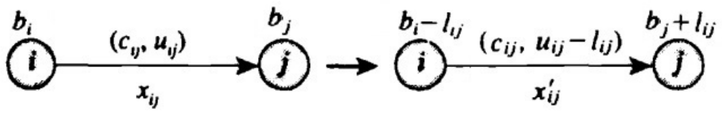


Figure 3.11: Removing nonzero lower bounds

Exercise 3.4.

Consider the MCFP shown in Figure 3.10. Suppose that arcs (1, 2) and (3, 5) have lower bounds equal to $l_{12} = l_{35} = 5$. Transform this problem to one where all arcs have zero lower bounds.

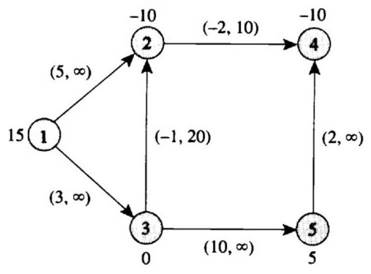


Figure 3.12: Directed graph after removing lower bounds $l_{12} = l_{35} = 5$