2 Fundamentals

Ulrik Brandes and Thomas Erlebach

In this chapter we discuss basic terminology and notation for graphs, some fundamental algorithms, and a few other mathematical preliminaries.

We denote the set of integers by $\mathbb{Z}$, the set of real numbers by $\mathbb{R}$, the set of complex numbers by $\mathbb{C}$, and the set of rationals by $\mathbb{Q}$. For a set X of numbers, X^+ denotes the subset of positive numbers in X, and X_0^+ the subset of non-negative numbers. The set of positive integers is denoted by $\mathbb{N} = \mathbb{Z}^+$ and the set of non-negative integers by $\mathbb{N}_0 = \mathbb{Z}_0^+$.

We use $\mathbb{R}^{n \times m}$ to denote the set of all real-valued matrices with n rows and m columns. If the entries of the matrix can be complex numbers, we write $\mathbb{C}^{n \times m}$. The n-dimensional identity matrix is denoted by I_n. The n-dimensional vector with all entries equal to 1 (equal to 0) is denoted by $\mathbf{1}_n$ (by $\mathbf{0}_n$).

For two functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$, we say that f is in $\mathcal{O}(g)$ if there are positive constants $n_0 \in \mathbb{N}$ and $c \in \mathbb{R}^+$ such that $f(n) \leq c \cdot g(n)$ holds for all $n \geq n_0$. Furthermore, we say that f is in $\Omega(g)$ if g is in $\mathcal{O}(f)$. This notation is useful to estimate the asymptotic growth of functions. In particular, running-times of algorithms are usually specified using this notation.

2.1 Graph Theory

We take the term *network* to refer to the informal concept describing an object composed of elements and interactions or connections between these elements. For example, the Internet is a network composed of nodes (routers, hosts) and connections between these nodes (e.g. fiber cables). The natural means to model networks mathematically is provided by the notion of graphs.

A *graph* $G = (V, E)$ is an abstract object formed by a set V of *vertices* (nodes) and a set E of edges (links) that join (connect) pairs of vertices. The vertex set and edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively. The cardinality of V is usually denoted by n, the cardinality of E by m. The two vertices joined by an edge are called its *endvertices*. If two vertices are joined by an edge, they are *adjacent* and we call them *neighbors*. Graphs can be *undirected* or *directed*. In undirected graphs, the order of the endvertices of an edge is immaterial. An undirected edge joining vertices $u, v \in V$ is denoted by $\{u, v\}$. In directed graphs, each directed edge (arc) has an *origin* (*tail*) and a *destination* (*head*). An edge with origin $u \in V$ and destination $v \in V$ is represented by an ordered pair (u, v). As a shorthand notation, an edge $\{u, v\}$ or (u, v) can also be denoted by uv. In a directed graph, uv is short for (u, v), while in an undirected graph, uv and vu are the same and both stand for $\{u, v\}$. For a directed graph $G = (V, E)$, the *underlying undirected graph* is the undirected graph with vertex set V that has an undirected edge between two vertices $u, v \in V$ if (u, v) or (v, u) is in E. Graphs that can have directed edges as well as undirected edges are called *mixed graphs*, but such graphs are encountered rarely and we will not discuss them explicitly in the following.

Multigraphs. In both undirected and directed graphs, we may allow the edge set E to contain the same edge several times, i.e., E can be a multiset. If an edge occurs several times in E, the copies of that edge are called *parallel edges.* Graphs with parallel edges are also called *multigraphs.* A graph is called *simple*, if each of its edges is contained in E only once, i.e., if the graph does not have parallel edges. An edge joining a vertex to itself, i.e., an edge whose endvertices are identical, is called a *loop.* A graph is called *loop-free* if it has no loops. We will assume all graphs to be loop-free unless specified otherwise.

Weighted graphs. Often it is useful to associate numerical values (weights) with the edges or vertices of a graph $G = (V, E)$. Here we discuss only edge weights. Edge weights can be represented as a function $\omega : E \to \mathbb{R}$ that assigns each edge $e \in E$ a weight $\omega(e)$. Depending on the context, edge weights can describe various properties such as cost (e.g. travel time or distance), capacity, strength of interaction, or similarity. One usually tries to indicate the characteristics of the edge weights by the choice of the name for the function. In particular, a function assigning (upper) capacities to edges is often denoted by u, especially in the context of network flow problems (see below). In general, we will mostly use ω to denote edge weights that express costs and other letters to denote edge weights that express capacities or interaction strengths. For most purposes, an unweighted graph $G = (V, E)$ is equivalent to a weighted graph with *unit edge weights* $\omega(e) = 1$ for all $e \in E$.

Degrees. The *degree* of a vertex v in an undirected graph $G = (V, E)$, denoted by $d(v)$, is the number of edges in E that have v as an endvertex. If G is a multigraph, parallel edges are counted according to their multiplicity in E. The set of edges that have v as an endvertex is denoted by $\Gamma(v)$. The set of neighbors of v is denoted by $N(v)$. In a directed graph $G = (V, E)$, the *out-degree* of $v \in V$, denoted by $d^+(v)$, is the number of edges in E that have origin v. The *in-degree* of $v \in V$, denoted by $d^-(v)$, is the number of edges with destination v. For weighted graphs, all these notions are generalized by summing over edge weights rather than taking their number. The set of edges with origin v is denoted by $\Gamma^+(v)$, the set of edges with destination v by $\Gamma^-(v)$. The set of destinations of edges in $\Gamma^+(v)$ is denoted by $N^+(v)$, the set of origins of edges in $\Gamma^-(v)$ by $N^-(v)$. If the graph under consideration is not clear from the context, these notations can be augmented by specifying the graph as an index. For example, $d_G(v)$ denotes the degree of v in G. The maximum and minimum degree of an undirected graph $G = (V, E)$ are denoted by $\Delta(G)$ and $\delta(G)$, respectively.

The average degree is denoted by $\bar{d}(G) = \frac{1}{|V|}\sum_{v \in V} d(v)$. An undirected graph $G = (V, E)$ is called *regular* if all its vertices have the same degree, and *r-regular* if that degree is equal to r.

Subgraphs. A graph $G' = (V', E')$ is a *subgraph* of the graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. It is a *(vertex-)induced subgraph* if E' contains all edges $e \in E$ that join vertices in V'. The induced subgraph of $G = (V, E)$ with vertex set $V' \subseteq V$ is denoted by $G[V']$. The *(edge-)induced subgraph* with edge set $E' \subseteq E$, denoted by $G[E']$, is the subgraph $G' = (V', E')$ of G, where V' is the set of all vertices in V that are endvertices of at least one edge in E', If C is a proper subset of V, then $G - C$ denotes the graph obtained from G by deleting all vertices in C and their incident edges. If F is a subset of E, $G - F$ denotes the graph obtained from G by deleting all edges in F.

Walks, paths and cycles. A *walk* from x_0 to x_k in a graph $G = (V, E)$ is an alternating sequence $x_0, e_1, x_1, e_2, x_2, \ldots, x_{k-1}, e_k, x_k$ of vertices and edges, where $e_i = \{x_{i-1}, x_i\}$ in the undirected case and $e_i = (x_{i-1}, x_i)$ in the directed case. The length of the walk is defined as the number of edges on the walk. The walk is called a *path*, if $e_i \neq e_j$ for $i \neq j$, and a path is a *simple path* if $x_i \neq x_j$ for $i \neq j$. A path with $x_0 = x_k$ is a *cycle*. A cycle is a *simple cycle* if $x_i \neq x_j$ for $0 \leq i < j \leq k - 1$.

2.2 Essential Problems and Algorithms

2.2.1 Connected Components

An undirected graph $G = (V, E)$ is *connected* if every vertex can be reached from every other vertex, i.e., if there is a path from every vertex to every other vertex. A graph consisting of a single vertex is also taken to be connected. Graphs that are not connected are called *disconnected*. For a given undirected graph $G = (V, E)$, a *connected component* of G is an induced subgraph $G' = (V', E')$ that is connected and maximal (i.e., there is no connected subgraph $G'' = (V'', E'')$ with $V'' \supset V'$). Checking whether a graph is connected and finding all its connected components can be done in time $\mathcal{O}(n + m)$ using depth-first search (DFS) or breadth-first search (BFS).

A directed graph $G = (V, E)$ is *strongly connected* if there is a directed path from every vertex to every other vertex. A *strongly connected component* of a directed graph G is an induced subgraph that is strongly connected and maximal. The strongly connected components of a directed graph can be computed in time $\mathcal{O}(n+m)$ using a modified DFS [542]. A directed graph is called *weakly connected* if its underlying undirected graph is connected.

2.2.2 Distances and Shortest Paths

For a path p in a graph $G = (V, E)$ with edge weights ω, the weight of the path, denoted by $\omega(p)$, is defined as the sum of the weights of the edges on p. A path from u to v in G is a *shortest path* (with respect to ω) if its weight is the smallest possible among all paths from u to v. The length of a shortest path from u to v, also called the shortest-path distance between u and v, is denoted by $d_{G,\omega}(u,v)$, where the subscripts G and/or ω are usually dropped if no confusion can arise.

The single-source shortest paths problem (SSSP) is defined as follows: Given a graph $G = (V, E)$ with edge weights $\omega : E \to \mathbb{R}$ and a vertex $s \in V$ (the source), compute shortest paths from s to all other vertices in the graph. The problem is only well-defined if the graph does not contain a cycle of negative weight. If the edge weights are non-negative, SSSP can be solved in time $\mathcal{O}(m + n \log n)$ using an efficient implementation of Dijkstra's algorithm [133]. If the edge weights are arbitrary, the Bellman-Ford algorithm uses time $\mathcal{O}(mn)$ to detect a cycle of negative length or, if no such cycle exists, solve the problem. For the special case of unit edge weights, BFS solves the problem in linear time $\mathcal{O}(n + m)$.

In the all-pairs shortest paths problem (APSP), one is given a graph $G = (V, E)$ with edge weights $\omega : E \to \mathbb{R}$ and wants to compute the shortest-path distances for all pairs of nodes. Provided that G does not contain a cycle of negative length, this problem can be solved by the Floyd-Warshall algorithm in time $\mathcal{O}(n^3)$, or by n SSSP computations in time $\mathcal{O}(nm + n^2 \log n)$.

These algorithms work for both directed and undirected graphs.

2.2.3 Network Flow

A *flow network* is given by a directed graph $G = (V, E)$, a function $u : E \to \mathbb{R}$ assigning non-negative capacities to the edges, and two distinct vertices $s, t \in V$ designated as the *source* and the *sink*, respectively. A flow f from s to t, or an s-t-flow for short, is a function $f : E \to \mathbb{R}$ satisfying the following constraints:

– Capacity constraints: $\forall e \in E : \ 0 \leq f(e) \leq u(e)$
– Balance conditions: $\forall v \in V \setminus \{s,t\} : \ \sum_{e \in \Gamma^-(v)} f(e) = \sum_{e \in \Gamma^+(v)} f(e)$

The *value* of the flow f is defined as

$$\sum_{e \in \Gamma^+(s)} f(e) - \sum_{e \in \Gamma^-(s)} f(e)\,.$$

The problem of computing a flow of maximum value is called the *max-flow problem.* The max-flow problem can be solved in time $\mathcal{O}(nm \log(n^2/m))$ using the algorithm of Goldberg and Tarjan [252], for example.

For a given graph $G = (V, E)$, a *cut* is a partition $(S, \bar{S})$ of V into two non-empty subsets S and $\bar{S}$. A cut $(S, \bar{S})$ is an *s-t-cut*, for $s, t \in V$, if $s \in S$ and $t \in \bar{S}$. The capacity of a cut $(S, \bar{S})$ is defined as the sum of the capacities of the edges with origin in S and destination in $\bar{S}$. A *minimum s-t-cut* is an s-t-cut whose capacity is minimum among all s-t-cuts. It is easy to see that the value of an s-t-flow can never be larger than the capacity of a s-t-cut. A classical result in the theory of network flows states that the maximum value and the minimum capacity are in fact the same.

Theorem 2.2.1 (Ford and Fulkerson [218]). *The value of a maximum s-t-flow is equal to the capacity of a minimum s-t-cut.*

Algorithms for the max-flow problem can also be used to compute a minimum s-t-cut efficiently. A *minimum cut* in an undirected graph $G = (V, E)$ with edge capacities $u : E \to \mathbb{R}$ is a cut that is an s-t-cut for some vertices $s, t \in V$ and has minimum capacity.

In the *min-cost flow problem*, one is given a directed graph $G = (V, E)$, a non-negative capacity function $u : E \to \mathbb{R}$, a cost function $c : E \to \mathbb{R}$, and a function $b : V \to \mathbb{R}$ assigning each vertex a demand/supply value. Here, a flow is a function $f : E \to \mathbb{R}$ that satisfies the capacity constraints and, in addition, the following version of the balance conditions:

$$\forall v \in V : \sum_{e \in \Gamma^+(v)} f(e) - \sum_{e \in \Gamma^-(v)} f(e) = b(v)$$

The cost of a flow f is defined as $c(f) = \sum_{e \in E} f(e)c(e)$. The problem of computing a flow of minimum cost can be solved in polynomial time.

2.2.4 Graph *k*-Connectivity

An undirected graph $G = (V, E)$ is called *k-vertex-connected* if $|V| > k$ and $G - X$ is connected for every $X \subset V$ with $|X| < k$. Note that every (non-empty) graph is 0-vertex-connected, and the 1-vertex-connected graphs are precisely the connected graphs on at least two vertices. Furthermore, a graph consisting of a single vertex is connected and 0-vertex-connected, but not 1-vertex-connected. The *vertex-connectivity* of G is the largest integer k such that G is k-vertex-connected. Similarly, G is called *k-edge-connected* if $|V| \geq 2$ and $G - Y$ is connected for every $Y \subseteq E$ with $|Y| < k$. The *edge-connectivity* of G is the largest integer k such that G is k-edge-connected. The edge-connectivity of a disconnected graph and of a graph consisting of a single vertex is 0.

The notions of vertex-connectivity and edge-connectivity can be adapted to directed graphs by requiring in the definitions above that $G - X$ and $G - Y$, respectively, be strongly connected.

Consider an undirected graph $G = (V, E)$. A subset $C \subset V$ is called a *vertex-separator* (or *vertex cutset*) if the number of connected components of $G - C$ is larger than that of G. If two vertices s and t are in the same connected component of G, but in different connected components of $G - C$, then C is called an s-t-vertex-separator. *Edge-separators* (*edge cutsets*) and *s-t-edge-separators* are defined analogously. The notion of s-t-separators can be adapted to directed graphs in the natural way: a set of vertices or edges is an s-t-separator if there is no more path from s to t after deleting the set from the graph.

Let $G = (V, E)$ be an undirected or directed graph. Two (directed or undirected) paths p_1 and p_2 from $s \in V$ to $t \in V$ are called *vertex-disjoint* if they do not share any vertices except s and t. They are called *edge-disjoint* if they do not share any edges. By Menger's Theorem (see Chapter 7 for further details), a graph G with at least $k+1$ vertices is k-vertex-connected if and only if there are k vertex-disjoint paths between any pair of distinct vertices, and a graph G with at least 2 vertices is k-edge-connected if and only if there are at least k edge-disjoint paths between any pair of distinct vertices.

The number of vertex- or edge-disjoint paths between two given vertices in a graph can be computed in polynomial time using network flow algorithms. Therefore, the vertex- and edge-connectivity of a graph can be determined in polynomial time as well. Special algorithms for these problems will be discussed in Chapter 7.

2.2.5 Linear Programming

Let A be a real $m \times n$ matrix, b a real m-dimensional vector, and c a real n-dimensional vector. Furthermore, let $x = (x_1, \dots, x_n)$ be a vector of n real variables. The optimization problem

$$\begin{aligned} \max\ & c^T x \\ \text{s.t.}\ & Ax \le b \\ & x \ge 0 \end{aligned}$$

is called a *linear program*. It asks to find a real vector x that satisfies the constraints $Ax \le b$ and $x \ge 0$ (where $\le$ is to be understood component-wise) and maximizes the objective function $c^T x = \sum_{i=1}^{n} c_i x_i$. Linear programs with rational coefficients can be solved in time polynomial in the size of the input.

If the variables of a linear program are constrained to be integers, the program is called an *integer linear program*. Computing optimal solutions to integer linear programs is an $\mathcal{NP}$-hard problem (see the next section), and no polynomial-time algorithm is known for this problem.

2.2.6 $\mathcal{NP}$-Completeness

It is important to consider the running-time of an algorithm for a given problem. Usually, one wants to give an upper bound on the running time of the algorithm for inputs of a certain size. If the running-time of an algorithm is $n^{\mathcal{O}(1)}$ for inputs of size n, we say that the algorithm runs in polynomial time. (For graph problems, the running-time is usually specified as a function of n and m, the number of edges and vertices of the graph, respectively.) For many problems, however, no polynomial-time algorithm has been discovered. Although one cannot rule out the possible existence of polynomial-time algorithms for such problems, the theory of $\mathcal{NP}$-completeness provides means to give evidence for the computational intractability of a problem. A decision problem is in the complexity class $\mathcal{NP}$ if there is a non-deterministic Turing machine that solves the problem in polynomial time. Equivalently, for every yes-instance of the problem there is a proof of polynomial size that can be verified in polynomial time. A decision problem is $\mathcal{NP}$-hard if every decision problem in $\mathcal{NP}$ can be reduced to it by a polynomial many-one reduction. Problems that are in $\mathcal{NP}$ and $\mathcal{NP}$-hard are called

$\mathcal{NP}$-complete. An example of an $\mathcal{NP}$-complete problem is Satisfiability, i.e., checking whether a given Boolean formula in conjunctive normal form has a satisfying truth assignment. A polynomial-time algorithm for an $\mathcal{NP}$-hard problem would imply a polynomial-time algorithm for all problems in $\mathcal{NP}$—something that is considered very unlikely. Therefore, $\mathcal{NP}$-hardness of a problem is considered substantial evidence for the computational difficulty of the problem. For optimization problems (where the goal is to compute a feasible solution that maximizes or minimizes some objective function), we say that the problem is $\mathcal{NP}$-hard if the corresponding decision problem (checking whether a solution with objective value better than a given value k exists) is $\mathcal{NP}$-hard. In order to solve $\mathcal{NP}$-hard optimization problems, the only known approaches either settle with approximate solutions or incur a potentially exponential running-time.

2.3 Algebraic Graph Theory

Two directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* (written as $G_1 \simeq G_2$) if there is a bijection $\phi : V_1 \to V_2$ with

$$\forall u, v \in V : \ (u, v) \in E_1 \Leftrightarrow (\phi(u), \phi(v)) \in E_2 \,.$$

Such a bijection is called an *isomorphism.* An isomorphism that maps a graph onto itself is called an *automorphism.* Usually we consider two graphs to be the same if they are isomorphic. Isomorphism and automorphism for undirected graphs are defined analogously.

The *incidence matrix* (or *node-arc incidence matrix*) of a directed graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$ is a matrix B with n rows and m columns that has entries $b_{i,j}$ satisfying

$$b_{i,j} = \begin{cases} -1, & \text{if } v_i \text{ is the origin of } e_j \\ 1, & \text{if } v_i \text{ is the destination of } e_j \\ 0, & \text{otherwise} \end{cases}$$

The *adjacency matrix* of a simple directed graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ is an $n \times n$ matrix $A(G) = (a_{i,j})_{1 \le i,j \le n}$ with

$$a_{i,j} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

If G is an undirected graph, its adjacency matrix is symmetric and has $a_{i,j} = 1$ if v_i and v_j are adjacent. For weighted graphs, the non-zero entries are $\omega(v_i, v_j)$ rather than 1.

The *Laplacian* of an undirected graph $G = (V, E)$ is an $n \times n$ matrix defined by $L(G) = D(G) - A(G)$, where $D(G)$ is the diagonal matrix that has its i-th diagonal entry equal to $d_G(v_i)$. Note that $L(G) = BB^T$ for any fixed orientation of the edges of G. The *normalized Laplacian* of G is the $n \times n$ matrix defined by $\mathcal{L}(G) = D(G)^{-1/2} L(G) D(G)^{-1/2}$, where $D(G)^{-1/2}$ is the diagonal matrix where the i-th diagonal entry is 0 if $d_G(v_i) = 0$ and $1/\sqrt{d_G(v_i)}$ otherwise.

Let $A \in \mathbb{C}^{n \times n}$ be a matrix. A value $\lambda \in \mathbb{C}$ is called an *eigenvalue* of A if there is a non-zero n-dimensional vector x such that $Ax = \lambda x$. Such a vector x is then called an *eigenvector* of A (with eigenvalue λ). The (multi-)set of all eigenvalues of a matrix is called its *spectrum*. It is equal to the set of the roots of the *characteristic polynomial* of A, where the characteristic polynomial of A is defined as the determinant of $A - \lambda \cdot I_n$.

If A is a real symmetric matrix, all eigenvalues are real. Therefore, the spectra of the adjacency matrix, the Laplacian, and the normalized Laplacian of an undirected graph $G = (V, E)$ are multisets containing n real values. The spectrum of the adjacency matrix $A(G)$ of a graph G is also called the spectrum of G. The spectra of the Laplacian and the normalized Laplacian of G are called the Laplacian spectrum and the normalized Laplacian spectrum of G.

2.4 Probability and Random Walks

A *discrete probability space* is a pair $(\Omega, \Pr)$, where Ω is a non-empty, finite or countably infinite set and Pr is a mapping from the power set $\mathcal{P}(\Omega)$ of Ω to the real numbers satisfying the following:

- $\Pr[A] \geq 0$, for all $A \subseteq \Omega$
- $\Pr[\Omega] = 1$
- $\Pr\left[\bigcup_{i \in \mathbb{N}} A_i\right] = \sum_{i \in \mathbb{N}} \Pr[A_i]$, for every sequence $(A_i)_{i \in \mathbb{N}}$ of pairwise disjoint sets from $\mathcal{P}(\Omega)$.

We call Ω a *sample space*. Subsets of Ω are called *events*. Note that we write the probability of an event A as $\Pr[A]$ (and not as $\Pr(A)$). The conditional probability of event A given the occurrence of event B is written as $\Pr[A \mid B]$ and is well-defined by $\Pr[A \cap B]/\Pr[B]$ whenever $\Pr[B] \neq 0$.

A random variable X is a mapping from the sample space to the real numbers. The image of X is denoted by $I_X = X(\Omega)$. The expected value of a random variable X is defined as $\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega) \Pr[\omega]$. Note that this definition implies $\mathbb{E}[X] = \sum_{x \in X(\Omega)} x \Pr[X = x]$.

A *Markov chain* on state set S, where S can be finite or countably infinite, is given by a sequence $(X_t)_{t \in \mathbb{N}_0}$ of random variables X_t with $I_{X_t} \subseteq S$ and an initial distribution q_0 that maps S to $\mathbb{R}_0^+$ and satisfies $\sum_{s \in S} q_0(s) = 1$. It must satisfy the *Markov condition*, i.e. for all $t > 0$ and all $I \subseteq \{0, 1, \ldots, t-1\}$ and all $i, j, s_k \in S$ we must have:

$$\Pr[X_{t+1} = j \mid X_t = i, \forall k \in I : \; X_k = s_k] = \Pr[X_{t+1} = j \mid X_t = i]$$

In words, the probability distribution of the successor state X_{t+1} depends only on the current state X_t, not on the history of how the chain has arrived in the current state. We interpret X_t as the state of the Markov chain at *time t*. By q_t we denote the probability distribution on the state set S at time t, i.e., q_t is a vector whose i-th entry, for $i \in S$, is defined by $q_t(i) = \Pr[X_t = i]$.

If $\Pr[X_{t+1} = j \mid X_t = i]$ is independent of t for all states $i, j \in S$, the Markov chain is called *homogeneous*. We consider only homogeneous Markov chains with finite state set S in the following. For such Markov chains, the *transition matrix* is defined as the $|S| \times |S|$ matrix $T = (t_{i,j})$ with $t_{i,j} = \Pr[X_{t+1} = j \mid X_t = i]$. The transition matrix is a *stochastic matrix*, i.e., a non-negative matrix in which the entries in each row sum up to 1. Note that the probability distribution q_{t+1} on the state set S at time $t+1$, viewed as a row vector, can be computed from the probability distribution q_t at time t by $q_{t+1} = q_t \cdot T$, for all $t \geq 0$. This implies that $q_t = q_0 \cdot T^t$ holds for all $t \geq 0$.

A Markov chain is called *irreducible* if for every pair (i,j) of states there exists a $k > 0$ such that $\Pr[X_k = j \mid X_0 = i] > 0$. In other words, a Markov chain is irreducible if every state can be reached from any given state with positive probability. The graph of a Markov chain is defined as the directed graph with vertex set S and edges (i,j) for all i,j with $\Pr[X_{t+1} = j \mid X_t = i] > 0$. A Markov chain is irreducible if and only if its graph is strongly connected.

The *period* of a state $s \in S$ of an irreducible Markov chain is the greatest common divisor of all $k > 0$ such that $\Pr[X_k = s \mid X_0 = s] > 0$. A Markov chain is *aperiodic* if all its states have period 1.

For a given Markov chain with state set S and transition matrix T, a non-negative row vector $\pi = (\pi_s)_{s \in S}$ is called a *stationary distribution* if $\sum_{s \in S} \pi_s = 1$ and $\pi \cdot T = \pi$. Every irreducible Markov chain with finite state set S has a unique stationary distribution. If, in addition, the Markov chain is aperiodic, the probability distribution on the states converges to the stationary distibution independently of the initial distribution, i.e., $\lim_{t \to \infty} q_t = \pi$.

The *hitting time* of state j starting at state i is the expected number of steps the Markov chain makes if it starts in state i at time 0 until it first arrives in state j at some time $t \geq 1$.

A *random walk* in a simple directed graph $G = (V, E)$ is a Markov chain with $S = V$ and:

$$\Pr[X_{t+1} = v \mid X_t = u] = \begin{cases} \frac{1}{d^+(u)}, & \text{if } (u,v) \in E \\ 0, \text{otherwise} \end{cases}$$

In every step, the random walk picks a random edge leaving the current vertex and follows it to the destination of that edge. The random walk is well-defined only if $d^+(v) \geq 1$ for all $v \in V$. In this case, the transition matrix of the random walk is the stochastic $|V| \times |V|$ matrix $T = (t_{i,j})$ with $t_{i,j} = 1/d^+(i)$ if $(i,j) \in E$ and $t_{i,j} = 0$ otherwise. Note that the Markov chain given by a random walk in a directed graph G is irreducible if and only if G is strongly connected.

Random walks in undirected graphs can be defined analogously.

2.5 Chapter Notes

There are many good textbooks for the topics discussed in this chapter. Graph theory is treated in [145, 67]. An introduction to algorithms can be found in [133]. Network flows are treated in [6]. Linear programming is covered extensively in [505]. The standard reference for the theory of $\mathcal{NP}$-completeness is [240]. A textbook about algebraic graph theory is [247]. An introduction to probability theory is provided by [498].