Chapter One

An introduction to distributed algorithms

Graph theory, distributed algorithms, and linear distributed algorithms are a fascinating scientific subject. In this chapter we provide a broad introduction to distributed algorithms by reviewing some preliminary graphical concepts and by studying some simple algorithms. We begin the chapter with one section introducing some basic notation and another section stating a few useful facts from matrix theory, dynamical systems, and convergence theorems based on invariance principles. In the third section of the chapter, we provide a primer on graph theory with a particular emphasis on algebraic aspects, such as the properties of adjacency and Laplacian matrices associated to a weighted digraph. In the next section of the chapter, we introduce the notion of synchronous network and of distributed algorithm. We state various complexity notions and study them in simple example problems such as the broadcast problem, the tree computation problem, and the leader election problem. In the fifth section of the chapter, we discuss linear distributed algorithms. We focus on linear algorithms defined by sequences of stochastic matrices and review the results on their convergence properties. We end the chapter with three sections on, respectively, bibliographical notes, proofs of the results presented in the chapter, and exercises.

1.1 ELEMENTARY CONCEPTS AND NOTATION

1.1.1 Sets and maps

We assume that the reader is familiar with basic notions from topology, such as the notions of open, closed, bounded, and compact sets. In this section, we just introduce some basic notation. We let $x \in S$ denote a point x belonging to a set S. If S is finite, we let $|S|$ denote the number of its elements. For a set S, we let $\mathbb{P}(S)$ and $\mathbb{F}(S)$ denote the collection of subsets of S and the collection of finite subsets of S, respectively. The empty set is denoted by $\emptyset$. The interior and the boundary of a set S are denoted by $\text{int}(S)$ and ∂S, respectively. If R is a subset of or equal to S, then we write $R \subset S$. If R is a strict subset of S, then we write $R \subsetneq S$. We describe

subsets of S defined by specific conditions via the notation

$$\{x \in S \mid \text{condition(s) on } x\}.$$

Given two sets S_1 and S_2, we let $S_1 \cup S_2$, $S_1 \cap S_2$, and $S_1 \times S_2$ denote the union, intersection, and Cartesian product of S_1 and S_2, respectively. Given a collection of sets $\{S_a\}_{a\in A}$ indexed by a set A, we interchangeably denote their Cartesian product by $\prod_{a\in A} S_a$ or by $\prod\{S_a \mid a \in A\}$. We adopt analogous notations for union and intersection. We denote by S^n the Cartesian product of n copies of the same S. The *diagonal set* $\mathrm{diag}(S^n)$ of S^n is given by $\mathrm{diag}(S^n) = \{(s, \ldots, s) \in S^n \mid s \in S\}$. The set $S_1 \setminus S_2$ contains all points in S_1 that do not belong to S_2.

We let $\mathbb{N}$ and $\mathbb{Z}_{\geq 0}$ denote the set of natural numbers and of non-negative integers, respectively. We let $\mathbb{R}$, $\mathbb{R}_{>0}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{C}$ denote the set of real numbers, strictly positive real numbers, non-negative real numbers, and complex numbers, respectively. The sets $\mathbb{R}^d$, $\mathbb{C}^d$, and $\mathbb{S}^d \subset \mathbb{R}^{d+1}$ are the d-dimensional Euclidean space, the d-dimensional complex space, and the d-dimensional sphere, respectively. The tangent space of $\mathbb{R}^d$, denoted by $T\mathbb{R}^d$, is the set of all vectors tangent to $\mathbb{R}^d$. Note that $T\mathbb{R}^d$ can be identified with $\mathbb{R}^d \times \mathbb{R}^d$ by mapping a vector v tangent to $\mathbb{R}^d$ at $x \in \mathbb{R}^d$ to the pair (x, v). Likewise, $T\mathbb{S}^d$ is the set of all vectors tangent to $\mathbb{S}^d$, and can be identified with $\mathbb{S}^d \times \mathbb{R}^d$. The Euclidean space $\mathbb{R}^d$ contains the vectors $\mathbf{0}_d = (0, \ldots, 0)$, $\mathbf{1}_d = (1, \ldots, 1)$, and the standard basis $\boldsymbol{e}_1 = (1, 0, \ldots, 0), \ldots, \boldsymbol{e}_d = (0, \ldots, 0, 1)$. Given $a < b$, we let $[a, b]$ and $]a, b[$ denote the closed interval and the open interval between a and b, respectively.

Given two sets S and T, we let $f : S \to T$ denote a map from S to T, that is, a unique way of associating an element of T to an element of S. The *image* of the map $f : S \to T$ is the set $\mathrm{image}(f) = \{f(s) \in T \mid s \in S\}$. Given the map $f : S \to T$ and a set $S_1 \subset S$, we let $f(S_1) = \{f(s) \mid s \in S_1\}$ denote the image of the set S_1 under the map f. Given $f : S \to T$ and $g : U \to S$, we let $f \circ g : U \to T$, defined by $f \circ g\,(u) = f(g(u))$, denote the composition of f and g. The map $\mathrm{id}_S : S \to S$ is the identity map on S. Given $f : S \to \mathbb{R}$, the *support* of f is the set of elements s such that $f(s) \neq 0$. Given a subset $R \subsetneq S$, the indicator map $1_R : S \to \mathbb{R}$ associated with R is given by $1_R(q) = 1$ if $q \in R$, and $1_R(q) = 0$ if $q \notin R$. Given two sets S and T, a *set-valued map*, denoted by $h : S \rightrightarrows T$, associates to an element of S a subset of T. Given a map $f : S \to T$, the *inverse map* $f^{-1} : T \rightrightarrows S$ is defined by

$$f^{-1}(t) = \{s \in S \mid f(s) = t\}.$$

If f is a real-valued function, that is, a function of the form $f : S \to \mathbb{R}$, then $f^{-1}(x) \subset S$, for any $x \in \mathbb{R}$, is a *level set* of f. In what follows, we require the reader to be familiar with some basic smoothness notions

for functions. Specifically, we will use the notions of locally and globally Lipschitz functions, differentiable, piecewise differentiable and continuously differentiable functions, and functions that are multiple times differentiable.

Finally, we introduce the so-called *Bachmann–Landau symbols*. For $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$, we say that $f \in O(g)$ (resp., $f \in \Omega(g)$) if there exist $n_0 \in \mathbb{N}$ and $K \in \mathbb{R}_{>0}$ (resp., $k \in \mathbb{R}_{>0}$) such that $f(n) \leq Kg(n)$ for all $n \geq n_0$ (resp., $f(n) \geq kg(n)$ for all $n \geq n_0$). If $f \in O(g)$ and $f \in \Omega(g)$, then we use the notation $f \in \Theta(g)$.

1.1.2 Distance functions

A function $\mathrm{dist} : S \times S \to \mathbb{R}_{\geq 0}$ defines a *distance* on a set S if it satisfies: (i) $\mathrm{dist}(x, y) = 0$ if and only if $x = y$; (ii) $\mathrm{dist}(x, y) = \mathrm{dist}(y, x)$, for all $x, y \in S$; and (iii) $\mathrm{dist}(x, y) \leq \mathrm{dist}(x, z) + \mathrm{dist}(z, y)$, for all $x, y, z \in S$. The pair (S, dist) is usually called a *metric space*.

Some relevant examples of distance functions include the following:

L^p-distance on $\mathbb{R}^d$. For $p \in [1, +\infty[$, consider the L^p-norm on $\mathbb{R}^d$ defined by $\|x\|_p = (\sum_{i=1}^d |x_i|^p)^{1/p}$. For $p = +\infty$, consider the L^∞-norm on $\mathbb{R}^d$ defined by $\|x\|_\infty = \max_{i \in \{1,\ldots,d\}} |x_i|$. Any of these norms defines naturally a *L^p-distance* in $\mathbb{R}^d$ by $\mathrm{dist}_p(x, y) = \|y - x\|_p$. In particular, the most widely used is the Euclidean distance, corresponding to $p = 2$. Unless otherwise noted, it is always understood that $\mathbb{R}^d$ is endowed with this notion of distance. We will also use the L^1- and the L^∞-distances. Finally, it is convenient to define the norm $\|z\|_{\mathbb{C}}$ of a complex number $z \in \mathbb{C}$ to be the Euclidean norm of z regarded as a vector in $\mathbb{R}^2$.

Geodesic distance on $\mathbb{S}^d$. Another example is the notion of *geodesic distance* on $\mathbb{S}^d$. This is defined as follows. For $x, y \in \mathbb{S}^d$, $\mathrm{dist}_g(x, y)$ is the length of the shortest curve in $\mathbb{S}^d$ connecting x and y. We will use this notion of distance in dimensions $d = 1$ and $d = 2$. On the unit circle $\mathbb{S}^1$, by convention, let us define positions as angles measured counterclockwise from the positive horizontal axis. Then, the geodesic distance can be expressed as

$$\mathrm{dist}_g(x, y) = \min\{\mathrm{dist}_{\mathsf{c}}(x, y), \mathrm{dist}_{\mathsf{cc}}(x, y)\}, \quad x, y \in \mathbb{S}^1,$$

where $\mathrm{dist}_{\mathsf{c}}(x, y) = (x - y) \mod 2\pi$ is the clockwise distance and $\mathrm{dist}_{\mathsf{cc}}(x, y) = (y - x) \mod 2\pi$ is the counterclockwise distance. Here the clockwise distance between two angles is the path length from an angle to the other traveling clockwise, and $x \mod 2\pi$ is the remainder

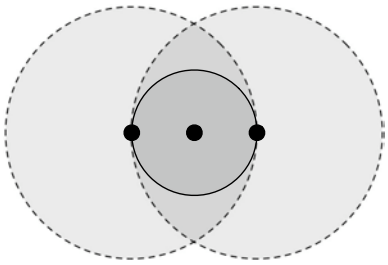


Figure 1.1 Open balls (dashed lines), a closed ball (solid line), and an open lune for the Euclidean distance on the plane.

of the division of x by 2π. On the sphere $\mathbb{S}^2$, the geodesic distance can be computed as follows. Given $x, y \in \mathbb{S}^2$, one considers the great circle determined by x and y. Then, the geodesic distance between x and y is exactly the length of the shortest arc in the great circle connecting x and y.

Cartesian product distance on $\mathbb{R}^{d_1} \times \mathbb{S}^{d_2}$. Consider $\mathbb{R}^{d_1}$ endowed with an L^p-distance, $p \in [1, +\infty]$, and $\mathbb{S}^{d_2}$ endowed with the geodesic distance. Given $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^{d_1} \times \mathbb{S}^{d_2}$, their *Cartesian product distance* is given by $\mathrm{dist}_p(x_1, x_2) + \mathrm{dist}_g(y_1, y_2)$. Unless otherwise noted, it is understood that $\mathbb{R}^{d_1} \times \mathbb{S}^{d_2}$ is endowed with the Cartesian product distance $(\mathrm{dist}_2, \mathrm{dist}_g)$.

Given a metric space (S, dist), the *open* and *closed balls* of center $x \in S$ and radius $\varepsilon \in \mathbb{R}_{>0}$ are defined by, respectively,

$$B(x, \varepsilon) = \{y \in S \mid \mathrm{dist}(x, y) < \varepsilon\},$$
$$\overline{B}(x, \varepsilon) = \{y \in S \mid \mathrm{dist}(x, y) \leq \varepsilon\}.$$

Consider a point $x \in X$ and a set $S \subset X$. A *neighborhood* of a point $x \in X$ is a subset of X that contains an open ball centered at x. A *neighborhood* of a set $Y \subset X$ is a subset of X that, for each point $y \in Y$, contains an open ball centered at y. The *open lune* associated to $x, y \in S$ is $B(x, \mathrm{dist}(x, y)) \cap B(y, \mathrm{dist}(x, y))$. These notions are illustrated in Figure 1.1 for the plane equipped with the Euclidean distance.

Given a metric space (S, dist), the distance between a point $x \in S$ and a set $W \subset S$ is the infimum of all distances between x and each of the points in W. Formally, we set

$$\mathrm{dist}(x, W) = \inf\{\mathrm{dist}(x, y) \mid y \in W\}.$$

The projection of a point $x \in S$ onto a set $W \subset S$ is the set-valued map

$\text{proj}_W : S \rightrightarrows W$ defined by

$$\text{proj}_W(x) = \{y \in W \mid \text{dist}(x, y) = \text{dist}(x, W)\}.$$

If W is a closed set, then $\text{proj}_W(x) = \emptyset$ for any $x \in S$. The *diameter* of a set is the maximum distance between any two points in the set; formally, we set $\text{diam}(S) = \sup\{\text{dist}(x, y) \mid x, y \in S\}$. With a slight abuse of notation, we often use $\text{diam}(P)$ to denote $\text{diam}(\{p_1, \ldots, p_n\})$ for $P = (p_1, \ldots, p_n)$.

1.1.3 Curves

A curve is the image of a continuous map $\gamma : [a, b] \to \mathbb{R}^d$. The map γ is called a *parameterization* of the curve. We usually identify a parameterization with the curve it defines. Without loss of generality, any curve can be given a parametrization with $a = 0$ and $b = 1$. A curve connects the two points p and q if $\gamma(0) = p$ and $\gamma(1) = q$. A curve $\gamma : [0, 1] \to \mathbb{R}^d$ is *not self-intersecting* if γ is injective on $(0, 1)$. A curve is *closed* if $\gamma(0) = \gamma(1)$.

A set $S \subset \mathbb{R}^d$ is *path connected* if any two points in S can be joined by a curve. A set $S \subset X$ is *simply connected* if it is path connected and any not self-intersecting closed curve can be continuously deformed to a point in the set; that is, for any injective continuous map $\gamma : [0, 1] \to S$ that satisfies $\gamma(0) = \gamma(1)$, there exist $p \in S$ and a continuous map $H : [0, 1] \times [0, 1] \to S$ such that $H(t, 0) = \gamma(t)$ and $H(t, 1) = p$ for all $t \in [0, 1]$. Informally, a simply connected set is a set that consists of a single piece and does not have any holes.

Next, consider a piecewise continuously differentiable curve $\gamma : [0, 1] \to \mathbb{R}^d$; the *length* of γ is

$$\text{length}(\gamma) = \int_0^1 \|\dot{\gamma}(s)\|_2 ds,$$

and its *arc-length parameter* is

$$s_{\text{arc}}(s) = \int_0^s \|\dot{\gamma}(t)\|_2 dt.$$

Note that as the parameter t varies in $[0, 1]$, the arc-length parameter $s_{\text{arc}}(t)$ varies in $[0, \text{length}(\gamma)]$. The *arc-length parameterization* of the curve is the map $\gamma_{\text{arc}} : [0, \text{length}(\gamma)] \to \mathbb{R}^d$ defined by the equation $\gamma_{\text{arc}}(s_{\text{arc}}(s)) = \gamma(s)$. With a slight abuse of notation, we will often drop the subindex arc and denote the arc-length parameterization by γ too.

For closed, not self-intersecting curves in the plane, we introduce the notion of signed and absolute curvatures as follows. Let $\gamma : [0, \text{length}(\gamma)] \to$

$\mathbb{R}^2$ be the counterclockwise arc-length parameterization of a curve. Assume γ is closed, not self-intersecting and twice continuously differentiable. Define the *tangent vector* $\gamma' : [0, \text{length}(\gamma)] \to \mathbb{R}^2$ by $\gamma'(s) = \frac{d\gamma}{ds}$. Note that the tangent vector has unit length, that is, $\|\gamma'(s)\|_2 = 1$ for all s. Additionally, define the *outward normal vector* $\text{n}_{\text{out}} : [0, \text{length}(\gamma)] \to \mathbb{R}^2$ to be the unit-length vector that is point-wise orthogonal to the tangent vector and directed outside the set enclosed by the closed curve γ. With these notations, the *signed curvature* $\kappa_{\text{signed}} : [0, \text{length}(\gamma)] \to \mathbb{R}$ is defined by requiring that it satisfies

$$\gamma''(s) = -\kappa_{\text{signed}}(s)\, \text{n}_{\text{out}}(s), \quad \text{and} \quad \text{n}'_{\text{out}}(s) = \kappa_{\text{signed}}(s)\gamma'(s).$$

If the set enclosed by the closed curve γ is strictly convex, then the signed curvature of γ is strictly positive. In general, the *(absolute) curvature* $\kappa_{\text{abs}} : [0, \text{length}(\gamma)] \to \mathbb{R}_{\geq 0}$ and the *radius of curvature* $\rho : [0, \text{length}(\gamma)] \to \mathbb{R}_{\geq 0}$ of the curve γ are defined by, respectively,

$$\kappa_{\text{abs}}(s) = |\kappa_{\text{signed}}(s)|, \quad \text{and} \quad \rho(s) = |\kappa_{\text{signed}}(s)|^{-1}.$$

1.2 MATRIX THEORY

Here, we present basic notions and results about matrix theory, following the treatments in Horn and Johnson (1985) and Meyer (2001). We let $\mathbb{R}^{n\times m}$ and $\mathbb{C}^{n\times m}$ denote the set of $n \times m$ real and complex matrices. Given a real matrix A and a complex matrix U, we let A^T and U^* denote the transpose of A and the conjugate transpose matrix of U, respectively. We let I_n denote the $n \times n$ identity matrix. For a square matrix A, we write $A > 0$, resp. $A \geq 0$, if A is symmetric positive definite, resp. symmetric positive semidefinite. For a real matrix A, we let kernel(A) and rank(A) denote the kernel and rank of A, respectively. Given a vector v, we let diag(v) denote the square matrix whose diagonal elements are equal to the component v and whose off-diagonal elements are zero.

1.2.1 Matrix sets

A matrix $A \in \mathbb{R}^{n\times n}$ with entries a_{ij}, $i, j \in \{1, \ldots, n\}$, is

(i) *Orthogonal* if $AA^T = I_n$, and is *special orthogonal* if it is orthogonal with $\det(A) = +1$. The set of orthogonal matrices is a group.[1]

[1] A set G with a binary operation, denoted by $G \times G \ni (a, b) \mapsto a \star b \in G$, is a *group* if: (i) $a \star (b \star c) = (a \star b) \star c$ for all $a, b, c \in G$ (associativity property); (ii) there exists $e \in G$ such that $a \star e = e \star a = a$ for all $a \in G$ (existence of an identity element); and (iii) there exists $a^{-1} \in G$ such that $a \star a^{-1} = a^{-1} \star a = e$ for all $a \in G$ (existence of inverse elements).

(ii) *Nonnegative* (resp., *positive*) if all its entries are nonnegative (resp., positive).

(iii) *Row-stochastic* (or *stochastic* for brevity) if it is nonnegative and $\sum_{j=1}^{n} a_{ij} = 1$, for all $i \in \{1, \ldots, n\}$; in other words, A is row-stochastic if

$$A\mathbf{1}_n = \mathbf{1}_n.$$

(iv) *Column-stochastic* if it is nonnegative and $\sum_{i=1}^{n} a_{ij} = 1$, for all $j \in \{1, \ldots, n\}$.

(v) *Doubly stochastic* if A is row-stochastic and column-stochastic.

(vi) *Normal* if $A^T A = AA^T$.

(vii) A *permutation matrix* if A has precisely one entry equal to one in each row, one entry equal to one in each column, and all other entries equal to zero. The set of permutation matrices is a group.

The scalars $\mu_1, \ldots, \mu_k$ are *convex combination coefficients* if $\mu_i \geq 0$, for $i \in \{1, \ldots, k\}$, and $\sum_{i=1}^{k} \mu_i = 1$. (Each row of a row-stochastic matrix contains convex combination coefficients.) A *convex combination* of vectors is a linear combination of the vectors with convex combination coefficients. A subset U of a vector space V is *convex* if the convex combination of any two elements of U takes value in U. For example, the set of stochastic matrices and the set of doubly stochastic matrices are convex.

Theorem 1.1 (Birkhoff–von Neumann). *A square matrix is doubly stochastic if and only if it is a convex combination of permutation matrices.*

Next, we review two families of relevant matrices with useful properties. *Toeplitz matrices* are square matrices with equal entries along each diagonal parallel to the main diagonal. In other words, a Toeplitz matrix is a matrix of the form

$$\begin{bmatrix} t_0 & t_1 & \ddots & \ddots & \ddots & t_{n-2} & t_{n-1} \\ t_{-1} & t_0 & t_1 & \ddots & \ddots & \ddots & t_{n-2} \\ \ddots & t_{-1} & t_0 & t_1 & \ddots & \ddots & \ddots \\ \ddots & \ddots & t_{-1} & t_0 & t_1 & \ddots & \ddots \\ \ddots & \ddots & \ddots & t_{-1} & t_0 & t_1 & \ddots \\ t_{-n+2} & \ddots & \ddots & \ddots & t_{-1} & t_0 & t_1 \\ t_{-n+1} & t_{-n+2} & \ddots & \ddots & \ddots & t_{-1} & t_0 \end{bmatrix}.$$

An $n \times n$ Toeplitz matrix is determined by its first row and column, and hence by $2n-1$ scalars.

Circulant matrices are square Toeplitz matrices where each two subsequent row vectors v_i and v_{i+1} have the following two properties: the last entry of v_i is the first entry of v_{i+1} and the first $(n-1)$ entries of v_i are the second $(n-1)$ entries of v_{i+1}. In other words, a circulant matrix is a matrix of the form

$$\begin{bmatrix} c_0 & c_1 & \ddots & \ddots & \ddots & c_{n-2} & c_{n-1} \\ c_{n-1} & c_0 & c_1 & \ddots & \ddots & \ddots & c_{n-2} \\ \ddots & c_{n-1} & c_0 & c_1 & \ddots & \ddots & \ddots \\ \ddots & \ddots & c_{n-1} & c_0 & c_1 & \ddots & \ddots \\ \ddots & \ddots & \ddots & c_{n-1} & c_0 & c_1 & \ddots \\ c_2 & \ddots & \ddots & \ddots & c_{n-1} & c_0 & c_1 \\ c_1 & c_2 & \ddots & \ddots & \ddots & c_{n-1} & c_0 \end{bmatrix},$$

and, therefore, it is determined by its first row.

1.2.2 Eigenvalues, singular values, and induced norms

We require the reader to be familiar with the notion of eigenvalue and of simple eigenvalue, that is, an eigenvalue with algebraic and geometric multiplicity[2] equal to 1. The set of eigenvalues of a matrix $A \in \mathbb{R}^{n\times n}$ is called its *spectrum* and is denoted by $\mathrm{spec}(A) \subset \mathbb{C}$. The *singular values* of the matrix $A \in \mathbb{R}^{n\times n}$ are the positive square roots of the eigenvalues of $A^T A$.

We begin with a well-known property of the spectrum of a matrix.

Theorem 1.2 (Geršgorin disks). *Let A be an $n \times n$ matrix. Then*

$$\mathrm{spec}(A) \subset \bigcup_{i\in\{1,\ldots,n\}} \Big\{ z \in \mathbb{C} \;\Big|\; \|z - a_{ii}\|_{\mathbb{C}} \leq \sum_{j=1, j\neq i}^{n} |a_{ij}| \Big\}.$$

Next, we review a few facts about normal matrices, their eigenvectors and their singular values.

[2]The algebraic multiplicity of an eigenvalue is the multiplicity of the corresponding root of the characteristic equation. The geometric multiplicity of an eigenvalue is the number of linearly independent eigenvectors corresponding to the eigenvalue. The algebraic multiplicity is greater than or equal to the geometric multiplicity.

Lemma 1.3 (Normal matrices). *For a matrix $A \in \mathbb{R}^{n\times n}$, the following statements are equivalent:*

(i) A is normal;

(ii) A has a complete orthonormal set of eigenvectors; and

(iii) A is unitarily similar to a diagonal matrix, that is, there exists a unitary[3] matrix U such that U^*AU is diagonal.

Lemma 1.4 (Singular values of a normal matrix). *If a normal matrix has eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$, then its singular values are $\{|\lambda_1|, \ldots, |\lambda_n|\}$.*

It is well known that real symmetric matrices are normal, are diagonalizable by orthogonal matrices, and have real eigenvalues. Additionally, circulant matrices are normal.

We conclude by defining the notion of induced norm of a matrix. For $p \in \mathbb{N}\cup\{\infty\}$, the *p-induced norm* of $A \in \mathbb{R}^{n\times n}$ is

$$\|A\|_p = \max\{\|Ax\|_p \mid \|x\|_p = 1\}.$$

One can see that

$$\|A\|_1 = \max_{j\in\{1,\ldots,n\}} \sum_{i=1}^{n} |a_{ij}|, \quad \|A\|_\infty = \max_{i\in\{1,\ldots,n\}} \sum_{j=1}^{n} |a_{ij}|,$$

$$\|A\|_2 = \max\{\sigma \mid \sigma \text{ is a singular value of } A\}.$$

1.2.3 Spectral radius and convergent matrices

The *spectral radius* of a matrix $A \in \mathbb{R}^{n\times n}$ is

$$\rho(A) = \max\{\|\lambda\|_{\mathbb{C}} \mid \lambda \in \operatorname{spec}(A)\}.$$

In other words, $\rho(A)$ is the radius of the smallest disk centered at the origin that contains the spectrum of A.

Lemma 1.5 (Induced norms and spectral radius). *For any square matrix A and in any norm $p \in \mathbb{N}\cup\{\infty\}$, $\rho(A) \le \|A\|_p$.*

We will often deal with matrices with an eigenvalue equal to 1 and all other eigenvalues strictly inside the unit disk. Accordingly, we generalize the notion of spectral radius as follows. For a square matrix A with $\rho(A) = 1$, we define the *essential spectral radius*

$$\rho_{\text{ess}}(A) = \max\{\|\lambda\|_{\mathbb{C}} \mid \lambda \in \operatorname{spec}(A) \setminus \{1\}\}. \tag{1.2.1}$$

[3] A complex matrix $U \in \mathbb{C}^{n\times n}$ is *unitary* if $U^{-1} = U^*$.

Next, we will consider matrices with useful convergence properties.

Definition 1.6 (Convergent and semi-convergent matrices). A matrix $A \in \mathbb{R}^{n\times n}$ is

(i) *semi-convergent* if $\lim_{\ell\to+\infty} A^\ell$ exists; and

(ii) *convergent* if it is semi-convergent and $\lim_{\ell\to+\infty} A^\ell = 0$. •

These two notions are characterized as follows.

Lemma 1.7 (Convergent and semi-convergent matrices). *The square matrix A is convergent if and only if $\rho(A) < 1$. Furthermore, A is semi-convergent if and only if the following three properties hold:*

(i) $\rho(A) \le 1$*;*

(ii) $\rho_{\text{ess}}(A) < 1$*, that is,* 1 *is an eigenvalue and* 1 *is the only eigenvalue on the unit circle; and*

(iii) the eigenvalue 1 *is* semisimple*, that is, it has equal algebraic and geometric multiplicity (possibly larger than one).*

In other words, A is semi-convergent if and only if there exists a nonsingular matrix T such that

$$A = T \begin{bmatrix} I_k & 0 \\ 0 & B \end{bmatrix} T^{-1},$$

where $B \in \mathbb{R}^{(n-k)\times(n-k)}$ is convergent, that is, $\rho(B) < 1$. With this notation, we have $\rho_{\text{ess}}(A) = \rho(B)$ and the algebraic and geometric multiplicity of the eigenvalue 1 is k.

1.2.4 Perron–Frobenius theory

Positive and nonnegative matrices have useful spectral properties. In what follows, the first theorem amounts to the original Perron's Theorem for positive matrices and the following theorems are the extension due to Frobenius for certain nonnegative matrices. We refer to (Horn and Johnson, 1985, Chapter 8) for a detailed treatment.

Theorem 1.8 (Perron-Frobenius for positive matrices). *If the square matrix A is positive, then*

(i) $\rho(A) > 0$*;*

(ii) $\rho(A)$ *is an eigenvalue, it is simple, and* $\rho(A)$ *is strictly larger than the magnitude of any other eigenvalue; and*

(iii) $\rho(A)$ *has an eigenvector with positive components.*

Requiring the matrix to be strictly positive is a key assumption that limits the applicability of this theorem. It turns out that it is possible to obtain the same results of the theorem under weaker assumptions.

Definition 1.9 (Irreducible matrix). A nonnegative matrix $A \in \mathbb{R}^{n \times n}$ is *irreducible* if, for any nontrivial partition $J \cup K$ of the index set $\{1, \ldots, n\}$, there exist $j \in J$ and $k \in K$ such that $a_{jk} = 0$.

Remark 1.10 (Properties of irreducible matrices). An equivalent definition of irreducibility is given as follows. A matrix $A \in \mathbb{R}^{n \times n}$ is *irreducible* if it is not reducible, and is *reducible* if either:

(i) $n = 1$ and $A = 0$; or

(ii) there exists a permutation matrix $P \in \mathbb{R}^{n \times n}$ and a number $r \in \{1, \ldots, n-1\}$ such that $P^T AP$ is block upper triangular with diagonal blocks of dimensions $r \times r$ and $(n-r) \times (n-r)$.

It is an immediate consequence that the property of irreducibility depends upon only the patterns of zeros and nonzero elements of the matrix. •

We can now weaken the assumption in Theorem 1.8 and obtain a comparable, but weaker, result for irreducible matrices.

Theorem 1.11 (Perron–Frobenius for irreducible matrices). *If the nonnegative square matrix* A *is irreducible, then*

(i) $\rho(A) > 0$*;*

(ii) $\rho(A)$ *is an eigenvalue, and it is simple; and*

(iii) $\rho(A)$ *has an eigenvector with positive components.*

In general, the spectral radius of a nonnegative irreducible matrix does not need to be the only eigenvalue of maximum magnitude. For example, the matrix $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ has eigenvalues $\{1, -1\}$. In other words, irreducible matrices do indeed have weaker spectral properties than positive matrices. Therefore, it remains unclear which nonnegative matrices have the same properties as those stated for positive matrices in Theorem 1.8.

Definition 1.12 (Primitive matrix). A nonnegative square matrix A is *primitive* if there exists $k \in \mathbb{N}$ such that A^k is positive. •

It is easy to see that if a nonnegative square matrix is primitive, then it is irreducible. In later sections we will provide a graph-theoretical characterization of primitive matrices; for now, we are finally in a position to sharpen the results of Theorem 1.11.

Theorem 1.13 (Perron–Frobenius for primitive matrices). *If the nonnegative square matrix A is primitive, then*

(i) $\rho(A) > 0$*;*

(ii) $\rho(A)$ *is an eigenvalue, it is simple, and* $\rho(A)$ *is strictly larger than the magnitude of any other eigenvalue; and*

(iii) $\rho(A)$ *has an eigenvector with positive components.*

We conclude this section by noting the following convergence property that is an immediate corollary to Lemma 1.7 and to Theorem 1.13.

Corollary 1.14. *If the nonnegative square matrix A is primitive, then the matrix $\rho(A)^{-1}A$ is semi-convergent.*

1.3 DYNAMICAL SYSTEMS AND STABILITY THEORY

In this section, we introduce some basic concepts about dynamical and control systems; see, for example Sontag (1998) and Khalil (2002). We discuss stability and attractivity notions as well as the invariance principle. We conclude with a treatment of set-valued systems and time-dependent systems.

1.3.1 State machines and dynamical systems

Here, we introduce three classes of dynamical and control systems: (i) state machines or discrete-time discrete-space dynamical systems; (ii) discrete-time continuous-space control systems; and (iii) continuous-time continuous-space control systems.

We begin with our specific definition of state machine. A *(deterministic, finite) state machine* is a tuple (X, U, X_0, f), where X is a finite set called the *state space*, U is a finite set called the *input space*, $X_0 \subset X$ is the set of *allowable initial states*, and $f : X \times U \to X$ is the *evolution map*. Given an input sequence $u : \mathbb{Z}_{\geq 0} \to U$, the state machine evolution $x : \mathbb{Z}_{\geq 0} \to X$ starting from $x(0) \in X_0$ is given by

$$x(\ell + 1) = f(x(\ell), u(\ell)), \quad \ell \in \mathbb{Z}_{\geq 0}.$$

We will often refer to a state machine as a *processor*. Note that, in a state

machine, both the state and the input spaces are finite or *discrete*. Often times, we will find it useful to consider systems that evolve in continuous space and that are time dependent. Let us then provide two additional definitions in the following paragraphs.

A *(time-dependent) discrete-time continuous-space control system* is a tuple (X, U, X_0, f), where X is a d-dimensional space chosen among $\mathbb{R}^d$, $\mathbb{S}^d$, and the Cartesian products $\mathbb{R}^{d_1} \times \mathbb{S}^{d_2}$, for some $d_1 + d_2 = d$, U is a compact subset of $\mathbb{R}^m$ containing $\mathbf{0}_m$, $X_0 \subset X$, and $f : \mathbb{Z}_{\geq 0} \times X \times U \to X$ is a continuous map. As before, the individual objects X, U, X_0, and f are termed the *state space*, *input space*, *allowable initial states*, and *evolution map*, respectively. Given an input sequence $u : \mathbb{Z}_{\geq 0} \to U$, the evolution $x : \mathbb{Z}_{\geq 0} \to X$ of the dynamical system starting from $x(0) \in X_0$ is given by

$$x(\ell + 1) = f(\ell, x(\ell), u(\ell)), \quad \ell \in \mathbb{Z}_{\geq 0}.$$

A *(time-dependent) continuous-time continuous-space control system* is a tuple (X, U, X_0, f), where X is a d-dimensional space chosen among $\mathbb{R}^d$, $\mathbb{S}^d$, and the Cartesian products $\mathbb{R}^{d_1} \times \mathbb{S}^{d_2}$, for some $d_1 + d_2 = d$, U is a compact subset of $\mathbb{R}^m$ containing $\mathbf{0}_m$, $X_0 \subset X$, and $f : \mathbb{R}_{\geq 0} \times X \times U \to TX$ is a continuously differentiable map. The individual objects X, U, X_0, and f are termed the *state space*, *input space*, *allowable initial states*, and *control vector field*, respectively. Given an input function $u : \mathbb{R}_{\geq 0} \to U$, the evolution $x : \mathbb{R}_{\geq 0} \to X$ of the dynamical system starting from $x(0) \in X_0$ is given by

$$\dot{x}(t) = f(t, x(t), u(t)), \quad t \in \mathbb{R}_{\geq 0}.$$

We often consider the case when the control vector field can be written as $f(t, x, u) = f_0(t, x) + \sum_{a=1}^{m} f_a(t, x)\, u_a$, for some continuously differentiable maps $f_0, f_1, \ldots, f_m : \mathbb{R}_{\geq 0} \times X \to TX$. Each of these individual maps is called a *(time-dependent) vector field*, and f is said to be a *control-affine* vector field. The control vector field f is *driftless* if $f(t, x, \mathbf{0}_m) = 0$ for all $x \in X$ and $t \in \mathbb{R}_{\geq 0}$.

Finally, the term *dynamical system* denotes a control system that is not subject to any external control action; this terminology is applicable both in discrete and continuous time. Furthermore, we will sometimes neglect to define a specific set of allowable initial states; in this case we mean that any point in the state space is allowable as initial condition.

1.3.2 Stability and attractivity notions

In this section, we consider a continuous-space dynamical system (X, f). We first consider the discrete-time case and later we briefly present the analogous continuous-time case. We study dynamical systems that are *time-invariant*. In discrete time, a time-invariant system is simply described by an evolution map of the form $f : X \to X$.

Definition 1.15 (Equilibrium point). A point $x_* \in X$ is an *equilibrium point* for the time-invariant dynamical system (X, f) if the constant curve $x : \mathbb{Z}_{\geq 0} \to X$, defined by $x(\ell) = x_*$ for all $\ell \in \mathbb{Z}_{\geq 0}$, is an evolution of the system. •

It can immediately be seen that a point x_* is an equilibrium point if and only if $f(x_*) = x_*$. We denote the set of equilibrium points of the dynamical system by $\mathrm{Equil}(X, f)$.

Definition 1.16 (Trajectories and sets). Let (X, f) be a time-invariant dynamical system and let W be a subset of X. Then:

(i) The set W is *positively invariant* for (X, f) if each evolution with initial condition in W remains in W for all subsequent times.

(ii) A trajectory $x : \mathbb{Z}_{\geq 0} \to X$ *approaches* a set $W \subset X$ if, for every neighborhood Y of W, there exists a time $\ell_0 > 0$ such that $x(\ell)$ takes values in Y for all subsequent times $\ell \geq \ell_0$. In such a case, we write $x(\ell) \to W$ as $\ell \to +\infty$. •

In formal terms, W is positively invariant if $x(0) \in W$ implies $x(\ell) \in W$ for all $\ell \in \mathbb{Z}_{\geq 0}$, where $x : \mathbb{Z}_{\geq 0} \to X$ is the evolution of (X, f) starting from $x(0)$.

Definition 1.17 (Stability and attractivity). For a time-invariant dynamical system (X, f), a set S is:

(i) *stable* if, for any neighborhood Y of S, there exists a neighborhood W of S such that every evolution of (X, f) with initial condition in W remains in Y for all subsequent times;

(ii) *unstable* if it is not stable;

(iii) *locally attractive* if there exists a neighborhood Y of S such that every evolution with initial condition in Y approaches the set S; and

(iv) *locally asymptotically stable* if it is stable and locally attractive.

Additionally, the set S is *globally attractive* if every evolution of the dynamical system approaches it and it is *globally asymptotically stable* if it is stable and globally attractive. •

Remark 1.18 (Continuous-time dynamical systems). It is straightforward to extend the previous definitions to the setting of continuous-time continuous-space dynamical systems. These notions are illustrated in Figure 1.2. •

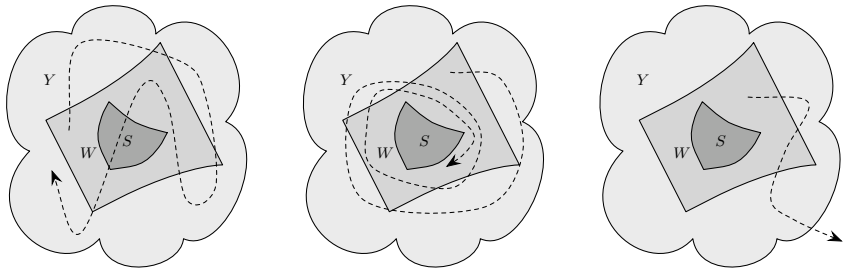


Figure 1.2 Illustrations of stability, asymptotic stability, and instability.

1.3.3 Invariance principles

Before discussing various versions of the invariance principle, we begin with a useful notion. Given a discrete-time time-invariant continuous-space dynamical system (X, f) and a set $W \subset X$, a function $V : X \to \mathbb{R}$ is *non-increasing along f in W* if $V(f(x)) \leq V(x)$ for all $x \in W$. (Such functions are often referred to as *Lyapunov functions*.) In other words, if a function V is non-increasing along f, then the composite function $\ell \mapsto V(y(\ell))$ is non-increasing for each evolution y of the dynamical system (X, f). The following theorem exploits this fact to establish useful properties of the evolutions of (X, f).

Theorem 1.19 (LaSalle Invariance Principle for discrete-time dynamical systems). *Let (X, f) be a discrete-time time-independent dynamical system. Assume that:*

(i) there exists a closed set $W \subset X$ that is positively invariant for (X, f);

(ii) there exists a function $V : X \to \mathbb{R}$ that is non-increasing along f on W;

(iii) all evolutions of (X, f) with initial conditions in W are bounded; and

(iv) f and V are continuous on W.

Then each evolution with initial condition in W approaches a set of the form $V^{-1}(c) \cap S$, where c is a real constant and S is the largest positively invariant set contained in $\{w \in W \mid V(f(w)) = V(w)\}$.

We refer to Section 1.8.1 for a discussion about the proof of this result. Next, we present the continuous-time version of the invariance principle. In other words, we now assume that (X, f) is a continuous-time time-invariant continuous-space dynamical system.

We begin by revisiting the notion of non-increasing function. Given a continuously differentiable function $V : X \to \mathbb{R}$, the *Lie derivative* of V along f, denoted by $\mathcal{L}_f V : X \to \mathbb{R}$, is defined by

$$\mathcal{L}_f V(x) = \frac{\mathrm{d}}{\mathrm{d}t} V(\gamma(t))_{\,t=0},$$

where the trajectory $\gamma :]-\varepsilon, \varepsilon[\to X$ satisfies $\dot{\gamma}(t) = f(\gamma(t))$ and $\gamma(0) = x$. If $X = \mathbb{R}^d$, then we can write x in components $(x_1, \ldots, x_d)$ and we can give the following explicit formula for the Lie derivative:

$$\mathcal{L}_f V(x) = \sum_{i=1}^{d} \frac{\partial V}{\partial x_i}(x) f_i(x).$$

Similar formulas can be obtained for more general state spaces. Note that, given a set $W \subset X$, a function $V : X \to \mathbb{R}$ is non-increasing along f in W if $\mathcal{L}_f V(x) \leq 0$ for all $x \in W$.

Finally, we state the invariance principle for continuous-time systems.

Theorem 1.20 (LaSalle Invariance Principle for continuous-time dynamical systems). *Let (X, f) be a continuous-time time-independent dynamical system. Assume that:*

(i) there exists a closed set $W \subset X$ that is positively invariant for (X, f);

(ii) there exists a function $V : X \to \mathbb{R}$ that is non-increasing along f on W;

(iii) all evolutions of (X, f) with initial conditions in W are bounded; and

(iv) f and V are continuously differentiable[4] on W.

Then, each evolution with initial condition in W approaches a set of the form $V^{-1}(c) \cap S$, where c is a real constant and S is the largest positively invariant set contained in $\{w \in W \mid \mathcal{L}_f V(w) = 0\}$.

[4]It suffices that f be locally Lipschitz and V be continuously differentiable; see Cortés (2008b).

1.3.4 Notions and results for set-valued systems

Next, we focus on a more sophisticated version of the LaSalle Invariance Principle for more general dynamical systems, that is, dynamical systems described by set-valued maps that allow for non-deterministic evolutions. To do so, we need to present numerous notions, including set-valued dynamical systems, closedness properties, and weak positive invariance.

Specifically, a *discrete-time continuous-space set-valued dynamical system* (in short, *set-valued dynamical system*) is determined by a tuple (X, X_0, T), where X is a d-dimensional space chosen among $\mathbb{R}^d$, $\mathbb{S}^d$, and the Cartesian products $\mathbb{R}^{d_1} \times \mathbb{S}^{d_2}$, for some $d_1 + d_2 = d$, $X_0 \subset X$, and $T : X \rightrightarrows X$ is a set-valued map. We assume that T assigns to each point $x \in X$ a nonempty set $T(x) \subset X$. The individual objects X, X_0, and T are termed the *state space*, *allowable initial states*, and *evolution map*, respectively. An *evolution* of the dynamical system (X, X_0, T) is any trajectory $x : \mathbb{Z}_{\geq 0} \to X$ satisfying

$$x(\ell + 1) \in T(x(\ell)), \quad \ell \in \mathbb{Z}_{\geq 0}.$$

Figure 1.3 illustrates this notion. In particular, a (time-invariant) discrete-time continuous-space dynamical system (X, X_0, f) can be seen as a discrete-time continuous-space set-valued dynamical system (X, X_0, T), where the evolution set-valued map is just the singleton-valued map $x \mapsto T(x) = \{f(x)\}$.

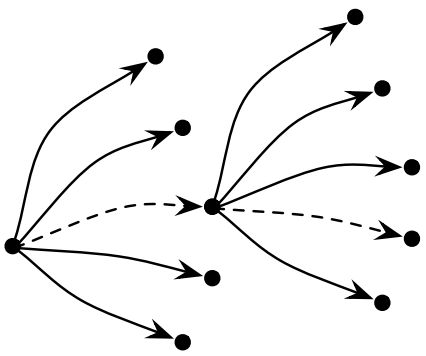


Figure 1.3 A discrete-time continuous-space set-valued dynamical system. A sample evolution is shown dashed.

Next, we introduce a notion of continuity for set-valued maps. The evolution map T is said to be *closed at* $x \in X$ if, for any sequences $\{x_k \mid k \in \mathbb{Z}_{\geq 0}\}$

and $\{y_k \mid k \in \mathbb{Z}_{\geq 0}\}$ such that

$$\lim_{k\to+\infty} x_k = x, \quad \lim_{k\to+\infty} y_k = y, \quad \text{and} \quad y_k \in T(x_k),$$

it holds that $y \in T(x)$. The evolution set-valued map T is *closed at* $W \subset X$ if for any $x \in W$, T is closed at x. Note that a continuous map $f : X \to X$ is closed when viewed as a singleton-valued map.

(i) A set $C \subset X$ is *weakly positively invariant with respect to* T if, for any $x \in C$, there exists $y \in C$ such that $y \in T(x)$.

(ii) A set $C \subset X$ is *strongly positively invariant with respect to* T if $T(x) \subset C$ for any $x \in C$.

A point x_0 is said to be a *fixed point of* T if $x_0 \in T(x_0)$. A continuous function $V : X \to \mathbb{R}$ is *non-increasing along* T *in* $W \subset X$ if $V(y) \leq V(x)$ for all $x \in W$ and $y \in T(x)$.

We finally state and prove a general version of the invariance principle, whose proof is presented in Section 1.8.1.

Theorem 1.21 (LaSalle Invariance Principle for set-valued discrete-time dynamical systems). *Let (X, X_0, T) be a discrete-time set-valued dynamical system. Assume that:*

(i) there exists a closed set $W \subset X$ that is strongly positively invariant for (X, X_0, T);

(ii) there exists a function $V : X \to \mathbb{R}$ that is non-increasing along T on W;

(iii) all evolutions of (X, X_0, T) with initial conditions in W are bounded; and

(iv) T is nonempty and closed at W and V is continuous on W.

Then, each evolutions with initial condition in W approaches a set of the form $V^{-1}(c) \cap S$, where c is a real constant and S is the largest weakly positively invariant set contained in $\{w \in W \mid \exists w' \in T(w)$ such that $V(w') = V(w)\}$.

1.3.5 Notions and results for time-dependent systems

In this final subsection, we consider time-dependent discrete-time dynamical systems and discuss *uniform* stability and convergence notions. We begin with some uniform boundedness, stability, and attractivity definitions.

In what follows, given a time-dependent discrete-time dynamical system (X, X_0, f), an evolution with initial condition in W at time $\ell_0 \in \mathbb{Z}_{\geq 0}$ is a trajectory $x : [\ell_0, +\infty[\to X$ of the dynamical system (X, X_0, f) defined by the initial condition $x(\ell_0) = x_0$, for some $x_0 \in W$. In other words, for time-dependent systems we will often consider trajectories that begin at time ℓ_0 not necessarily equal to zero.

Definition 1.22 (Uniformly bounded evolutions). A time-dependent discrete-time dynamical system (X, X_0, f) has *uniformly bounded evolutions* if, given any bounded set Y, there exists a bounded set W such that every evolution with initial condition in Y at any time $\ell_0 \in \mathbb{Z}_{\geq 0}$ remains in W for all subsequent times $\ell \geq \ell_0$. •

Definition 1.23 (Uniform stability and attractivity notions). For a time-dependent discrete-time dynamical system (X, X_0, f), the set S is:

(i) *uniformly stable* if, for any neighborhood Y of S, there exists a neighborhood W of S such that every evolution with initial condition in W at any time $\ell_0 \in \mathbb{Z}_{\geq 0}$ remains in Y for all subsequent times $\ell \geq \ell_0$;

(ii) *uniformly locally attractive* if there exists a neighborhood Y of S such that every evolution with initial condition in Y at any time ℓ_0 approaches the set S in the following *time-uniform* manner:

> for all $\ell_0 \in \mathbb{Z}_{\geq 0}$, for all $x_0 \in Y$, and for all neighborhoods W of S, there exists a single $\tau_0 \in \mathbb{Z}_{\geq 0}$ such that the evolution $x : [\ell_0, +\infty[\to X$ defined by $x(\ell_0) = x_0$ takes value in W for all times $\ell \geq \ell_0 + \tau_0$; and

(iii) *uniformly locally asymptotically stable* if it is uniformly stable and uniformly locally attractive.

Additionally, the set S is *uniformly globally attractive* if every evolution of the dynamical system approaches the set in a time-uniform manner, and it is *uniformly globally asymptotically stable* if it is uniformly stable and uniformly globally attractive. •

With the same notation in the definition, the set S is *(non-uniformly) locally attractive* if for all $\ell_0 \in \mathbb{Z}_{\geq 0}$, $x_0 \in Y$, and neighborhoods W of S, the evolution $x : [\ell_0, +\infty[\to X$ defined by $x(\ell_0) = x_0$, takes value in W for all times $\ell \geq \ell_0 + \tau_0(\ell_0)$, for some $\tau_0(\ell_0) \in \mathbb{Z}_{\geq 0}$.

To establish uniform stability and attractivity results we will overapproximate the evolution of the time-dependent dynamical system by considering the larger set of evolutions of an appropriate set-valued dynamical system. Given a time-dependent evolution map $f : \mathbb{Z}_{\geq 0} \times X \to X$, define a set-valued

overapproximation map $T_f : X \rightrightarrows X$ by

$$T_f(x) = \{f(\ell, x) \mid \ell \in \mathbb{Z}_{\geq 0}\}.$$

With this notion we can state a useful result, whose proof is left to the reader as an exercise.

Lemma 1.24 (Overapproximation Lemma). *Consider a discrete-time time-dependent dynamical system* (X, X_0, f)*:*

(i) If $x : [\ell_0, +\infty[\to X$ *is an evolution of the dynamical system* (X, f)*, then* $y : \mathbb{Z}_{\geq 0} \to X$ *defined by* $y(\ell) = x(\ell + \ell_0)$ *is an evolution of the set-valued* overapproximation system (X, T_f)*.*

(ii) If the set S *is locally attractive for the set-valued overapproximation system* (X, T_f)*, then it is uniformly locally attractive for* (X, f)*.*

In other words, every evolution of the time-dependent dynamical system from any initial time is an evolution of the set-valued overapproximation system and, therefore, the set of trajectories of the set-valued overapproximation system contains the set of trajectories of the original time-dependent system. Uniform attractivity is a consequence of attractivity for the time-invariant set-valued overapproximation.

1.4 GRAPH THEORY

Here we present basic definitions about graph theory, following the treatments in the literature; see, for example Biggs (1994), Godsil and Royle (2001), and Diestel (2005).

A *directed graph*—in short, *digraph*—of order n is a pair $G = (V, E)$, where V is a set with n elements called *vertices* (or *nodes*) and E is a set of ordered pair of vertices called *edges*. In other words, $E \subseteq V \times V$. We call V and E the *vertex set* and *edge set*, respectively. When convenient, we let $V(G)$ and $E(G)$ denote the vertices and edges of G, respectively. For $u, v \in V$, the ordered pair (u, v) denotes an edge *from* u *to* v.

An *undirected graph*—in short, *graph*—consists of a vertex set V and of a set E of unordered pairs of vertices. For $u, v \in V$ and $u = v$, the set $\{u, v\}$ denotes an unordered edge. A digraph is *undirected* if $(v, u) \in E$ anytime $(u, v) \in E$. It is possible and convenient to identify an undirected digraph with the corresponding graph; vice versa, the *directed version* of a graph (V, E) is the digraph (V', E') with the property that $(u, v) \in E'$ if and only if $\{u, v\} \in E$. In what follows, our convention is to allow self-loops in both graphs and digraphs.

A digraph (V', E') is a *subgraph* of a digraph (V, E) if $V' \subset V$ and $E' \subset E$; additionally, a digraph (V', E') is a *spanning subgraph* if it is a subgraph and $V' = V$. The subgraph of (V, E) *induced by* $V' \subset V$ is the digraph (V', E'), where E' contains all edges in E between two vertices in V'. For two digraphs $G = (V, E)$ and $G' = (V', E')$, the *intersection* and *union* of G and G' are defined by

$$G \cap G' = (V \cap V', E \cap E'),$$
$$G \cup G' = (V \cup V', E \cup E').$$

Analogous definitions may be given for graphs.

In a digraph G with an edge $(u, v) \in E$, u is called an *in-neighbor* of v, and v is called an *out-neighbor* of u. We let $\mathcal{N}^{\text{in}}_G(v)$ (resp., $\mathcal{N}^{\text{out}}_G(v)$) denote the set of in-neighbors, (resp. the set of out-neighbors) of v in the digraph G. We will drop the subscript when the graph G is clear from the context. The *in-degree* and *out-degree* of v are the cardinality of $\mathcal{N}^{\text{in}}(v)$ and $\mathcal{N}^{\text{out}}(v)$, respectively. A digraph is *topologically balanced* if each vertex has the same in- and out-degrees (even if distinct vertices have distinct degrees). Likewise, in an undirected graph G, the vertices u and v are *neighbors* if $\{u, v\}$ is an undirected edge. We let $\mathcal{N}_G(v)$ denote the set of neighbors of v in the undirected graph G. As in the directed case, we will drop the subscript when the graph G is clear from the context. The *degree* of v is the cardinality of $\mathcal{N}(v)$.

Remark 1.25 (Additional notions). For a digraph $G = (V, E)$, the *reverse digraph* $\text{rev}(G)$ has vertex set V and edge set $\text{rev}(E)$ composed of all edges in E with reversed direction. A digraph $G = (V, E)$ is *complete* if $E = V \times V$. A *clique* (V', E') of a digraph (V, E) is a subgraph of (V, E) which is complete, that is, such that $E' = V' \times V'$. Note that a clique is fully determined by its set of vertices, and hence there is no loss of precision in denoting it by V'. A *maximal clique* V' *of an edge* of a digraph is a clique of the digraph with the following two properties: it contains the edge, and any other subgraph of the digraph that strictly contains $(V', V' \times V')$ is not a clique. •

1.4.1 Connectivity notions

Let us now review some basic connectivity notions for digraphs and graphs. We begin with the setting of undirected graphs because of its simplicity.

A *path* in a graph is an ordered sequence of vertices such that any pair of consecutive vertices in the sequence is an edge of the graph. A graph is *connected* if there exists a path between any two vertices. If a graph is not

connected, then it is composed of multiple *connected components*, that is, multiple connected subgraphs. A path is *simple* if no vertices appear more than once in it, except possibly for initial and final vertex. A *cycle* is a simple path that starts and ends at the same vertex. A graph is *acyclic* if it contains no cycles. A connected acyclic graph is a *tree*. A *forest* is a graph that can be written as the disjoint union of trees. Trees have interesting properties: for example, $G = (V, E)$ is a tree if and only if G is connected and $|E| = |V| - 1$. Alternatively, $G = (V, E)$ is a tree if and only if G is acyclic and $|E| = |V| - 1$. Figure 1.4 illustrates these notions.

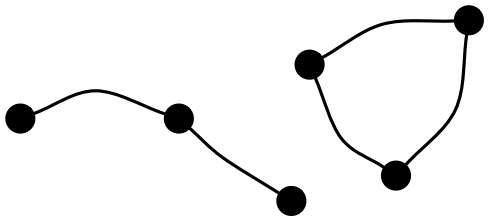
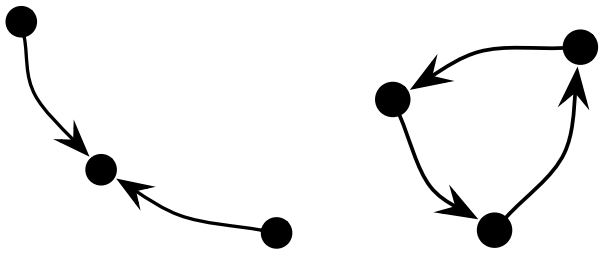


Figure 1.4 An illustration of connectivity notions on a graph. The graph has two connected components. The leftmost connected component is a tree, while the rightmost connected component is a cycle.

Next, we generalize these notions to the case of digraphs. A *directed path* in a digraph is an ordered sequence of vertices such that any ordered pair of vertices appearing consecutively in the sequence is an edge of the digraph. A *cycle* in a digraph is a directed path that starts and ends at the same vertex and that contains no repeated vertex except for the initial and the final vertex. A digraph is *acyclic* if it contains no cycles. In an acyclic graph, every vertex of in-degree 0 is named a *source*, and every vertex of out-degree 0 is named a *sink*. Every acyclic digraph has at least one source and at least one sink. Figure 1.5 illustrates these notions.

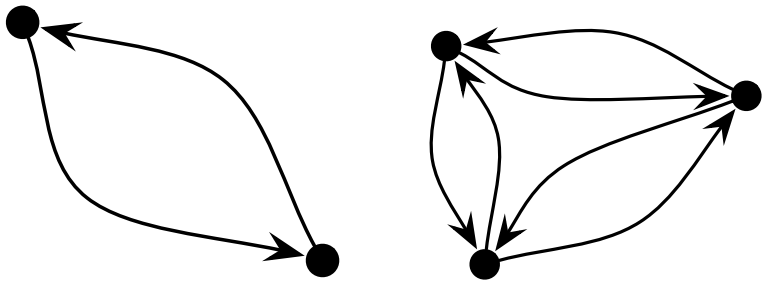


(a) (b)

Figure 1.5 Illustrations of connectivity notions on a digraph: (a) shows an acyclic digraph with one sink and two sources; (b) shows a directed path which is also a cycle.

The set of cycles of a directed graph is finite. A directed graph is *aperiodic* if there exists no $k > 1$ that divides the length of every cycle of the graph.

In other words, a digraph is aperiodic if the greatest common divisor of the lengths of its cycles is one. A digraph is *periodic* if it is not aperiodic. Figure 1.6 shows examples of a periodic and an aperiodic digraph.



(a) (b)

Figure 1.6 (a) A periodic digraph. (b) An aperiodic digraph with cycles of length 2 and 3.

A vertex of a digraph is *globally reachable* if it can be reached from any other vertex by traversing a directed path. A digraph is *strongly connected* if every vertex is globally reachable. The decomposition of a digraph into its strongly connected components and the notion of condensation digraph are discussed in Exercise E1.13.

A *directed tree* (sometimes called a rooted tree) is an acyclic digraph with the following property: there exists a vertex, called the *root*, such that any other vertex of the digraph can be reached by one and only one directed path starting at the root. In a directed tree, every in-neighbor of a vertex is called a *parent* and every out-neighbor is called a *child*. Two vertices with the same parent are called *siblings*. A *successor* of a vertex u is any other node that can be reached with a directed path starting at u. A *predecessor* of a vertex v is any other node such that a directed path exists starting at it and reaching v. A *directed spanning tree*, or simply a *spanning tree*, of a digraph is a spanning subgraph that is a directed tree. Clearly, a digraph contains a spanning tree if and only if the reverse digraph contains a globally reachable vertex. A *(directed) chain* is a directed tree with exactly one source and one sink. A *(directed) ring digraph* is the cycle obtained by adding to the edge set of a chain a new edge from its sink to its source. Figure 1.7 illustrates some of these notions.

The proof of the following result is given in Section 1.8.2.

Lemma 1.26 (Connectivity in topologically balanced digraphs). *Let G be a digraph. The following statements hold:*

(i) if G is strongly connected, then it contains a globally reachable vertex

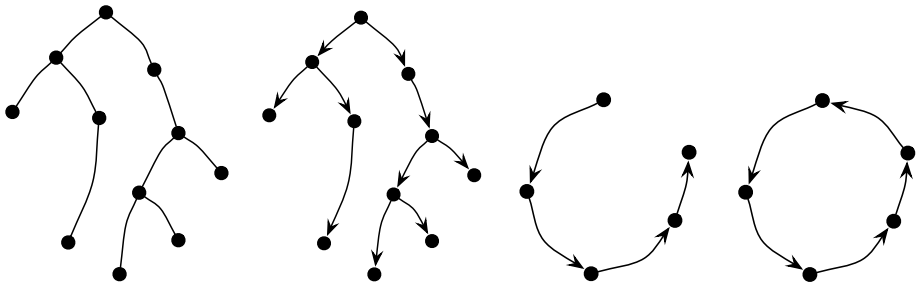


Figure 1.7 From left to right, tree, directed tree, chain, and ring digraphs.

and a spanning tree; and

(ii) if G is topologically balanced and contains either a globally reachable vertex or a spanning tree, then G is strongly connected and is Eulerian.[5]

Given a digraph $G = (V, E)$, an *in-neighbor* of a nonempty set of nodes U is a node $v \in V \setminus U$ for which there exists an edge $(v, u) \in E$ for some $u \in U$.

Lemma 1.27 (Disjoint subsets and spanning trees). *Given a digraph G with at least two nodes, the following two properties are equivalent:*

(i) G has a spanning tree; and

(ii) for any pair of nonempty disjoint subsets $U_1, U_2 \subset V$, either U_1 has an in-neighbor or U_2 has an in-neighbor.

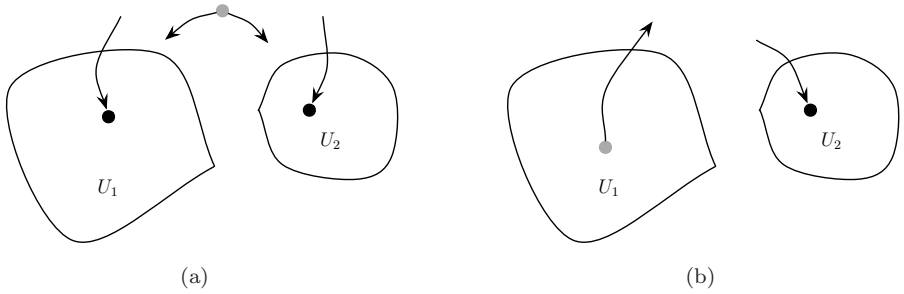


Figure 1.8 An illustration of Lemma 1.27. The root of the spanning tree is plotted in gray. In (a), the root is outside the sets U_1 and U_2. Because these sets are non-empty, there exists a directed path from the root to a vertex in each one of these sets. Therefore, both U_1 and U_2 have in-neighbors. In (b), the root is contained in U_1. Because U_2 is non-empty, there exists a directed path from the root to a vertex in U_2, and, therefore, U_2 has in-neighbors. The case when the root belongs to U_2 is treated analogously.

[5] A graph is Eulerian if it has a cycle that visits all the graph edges exactly once.

We will postpone the proof to Section 1.8.2. The result is illustrated in Figure 1.8. We can also state the result in terms of global reachability: G has a globally reachable node if and only if, for any pair of nonempty disjoint subsets U_1, $U_2 \subset V$, either U_1 has an out-neighbor or U_2 has an out-neighbor. We let the reader give a proper definition of the out-neighbor of a set.

1.4.2 Weighted digraphs

A *weighted digraph* is a triplet $G = (V, E, A)$, where the pair (V, E) is a digraph with nodes $V = \{v_1, \ldots, v_n\}$, and where the nonnegative matrix $A \in \mathbb{R}^{n \times n}_{\geq 0}$ is a *weighted adjacency matrix* with the following property: for $i, j \in \{1, \ldots, n\}$, the entry $a_{ij} > 0$ if (v_i, v_j) is an edge of G, and $a_{ij} = 0$ otherwise. In other words, the scalars a_{ij}, for all $(v_i, v_j) \in E$, are a set of weights for the edges of G. Note that the edge set is uniquely determined by the weighted adjacency matrix and it can therefore be omitted. When convenient, we denote the adjacency matrix of a weighted digraph G by $A(G)$. Figure 1.9 shows an example of a weighted digraph.

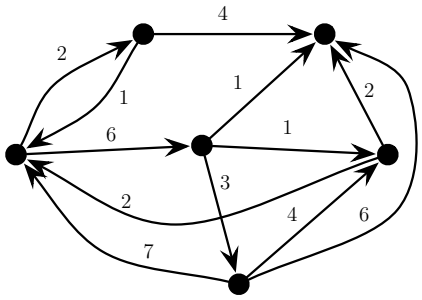


Figure 1.9 A weighted digraph with natural weights.

A digraph $G = (V, E)$ can be naturally thought of as a weighted digraph by defining the weighted adjacency matrix $A \in \{0, 1\}^{n \times n}$ as

$$a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \\ 0, & \text{otherwise,} \end{cases} \tag{1.4.1}$$

where $V = \{v_1, \ldots, v_n\}$. The adjacency matrix of a graph is the adjacency matrix of the directed version of the graph. Reciprocally, given a weighted digraph $G = (V, E, A)$, we refer to the digraph (V, E) as the *unweighted version* of G and to its associated adjacency matrix as the *unweighted adjacency matrix*. A weighted digraph is *undirected* if $a_{ij} = a_{ji}$ for all $i, j \in \{1, \ldots, n\}$. Clearly, G is undirected if and only if $A(G)$ is symmetric.

Numerous concepts introduced for digraphs remain equally valid for the case of weighted digraphs, including the connectivity notions and the definitions of in- and out-neighbors.

Finally, we generalize the notions of in- and out-degree to weighted digraphs. In a weighted digraph $G = (V, E, A)$ with $V = \{v_1, \ldots, v_n\}$, the *weighted out-degree* and the *weighted in-degree* of vertex v_i are defined by, respectively,

$$d_{\text{out}}(v_i) = \sum_{j=1}^{n} a_{ij}, \quad \text{and} \quad d_{\text{in}}(v_i) = \sum_{j=1}^{n} a_{ji}.$$

The weighted digraph G is *weight-balanced* if $d_{\text{out}}(v_i) = d_{\text{in}}(v_i)$ for all $v_i \in V$. The *weighted out-degree matrix* $D_{\text{out}}(G)$ and the *weighted in-degree matrix* $D_{\text{in}}(G)$ are the diagonal matrices defined by

$$D_{\text{out}}(G) = \text{diag}(A\mathbf{1}_n), \quad \text{and} \quad D_{\text{in}}(G) = \text{diag}(A^T\mathbf{1}_n).$$

That is, $(D_{\text{out}}(G))_{ii} = d_{\text{out}}(v_i)$ and $(D_{\text{in}}(G))_{ii} = d_{\text{in}}(v_i)$, respectively.

1.4.3 Distances on digraphs and weighted digraphs

We first present a few definitions for unweighted digraphs. Given a digraph G, the *(topological) length* of a directed path is the number of the edges composing it. Given two vertices u and v in the digraph G, the *distance* from u to v, denoted $\text{dist}_G(u, v)$, is the smallest length of any directed path from u to v, or $+\infty$ if there is no directed path from u to v. That is,

$$\text{dist}_G(u, v) = \min\big(\{\text{length}(p) \mid p \text{ is a directed path from } u \text{ to } v\} \cup \{+\infty\}\big).$$

Given a vertex v of a digraph G, the *radius* of v in G is the maximum of all the distances from v to any other vertex in G. That is,

$$\text{radius}(v, G) = \max\{\text{dist}_G(v, u) \mid u \in V(G)\}.$$

If T is a directed tree and v is its root, then the *depth* of T is $\text{radius}(v, T)$. Finally, the *diameter* of the digraph G is

$$\text{diam}(G) = \max\{\text{dist}_G(u, v) \mid u, v \in V(G)\}.$$

These definitions lead to the following simple results:

(i) $\text{radius}(v, G) \leq \text{diam}(G)$ for all vertices v of G;

(ii) G contains a spanning tree rooted at v if and only if $\text{radius}(v, G) < +\infty$; and

(iii) G is strongly connected if and only if $\text{diam}(G) < +\infty$.

The definitions of path length, distance between vertices, radius of a vertex, and diameter of a digraph can be easily applied to undirected graphs.

Next, we consider weighted digraphs. Given two vertices u and v in the weighted digraph G, the *weighted distance* from u to v, denoted $\mathrm{wdist}_G(u, v)$, is the smallest weight of any directed path from u to v, or $+\infty$ if there is no directed path from u to v. That is,

$$\mathrm{wdist}_G(u, v) = \min\big(\{\mathrm{weight}(p) \mid p \text{ is a directed path from } u \text{ to } v\} \cup \{+\infty\}\big).$$

Here, the weight of a subgraph of a weighted digraph is the sum of the weights of all the edges of the subgraph. Note that when a digraph is thought of as a weighted digraph (with the unweighted adjacency matrix (1.4.1)), the notions of weight and weighted distance correspond to the usual notions of length and distance, respectively. We leave it the reader to provide the definitions of weighted radius, weighted depth, and weighted diameter.

1.4.4 Graph algorithms

In this section, we present a few algorithms defined on graphs. We present only high-level descriptions and we refer to Cormen et al. (2001) for a comprehensive discussion including a detailed treatment of computationally efficient data structures and algorithmic implementations.

1.4.4.1 Breadth-first spanning tree

Let v be a vertex of a digraph G with $\mathrm{radius}(v, G) < +\infty$. A *breadth-first spanning (BFS) tree* of G with respect to v, denoted T_{BFS}, is a spanning directed tree rooted at v that contains a shortest path from v to every other vertex of G. (Here, a shortest path is one with the shortest topological length.) Let us provide the BFS ALGORITHM that, given a digraph G of order n and a vertex v with $\mathrm{radius}(v, G) < +\infty$, computes a BFS tree T_{BFS} rooted at v:

> *[Informal description]* Initialize a subgraph to contain only the root v. Repeat $\mathrm{radius}(v, G)$ times the following instructions: attach to the subgraph all out-neighbors of the subgraph as well as a single connecting edge for each out-neighbor. The final subgraph is the desired directed tree.

The algorithm is formally stated as follows:

`function` BFS(G, v)

1: $(V_1, E_1) := (\{v\}, \emptyset)$
2: **for** $k = 2$ to radius(v, G) **do**
3: find all vertices $w_1, \ldots, w_m$ not in V_{k-1} that are out-neighbors of some vertex in V_{k-1} and, for $j \in \{1, \ldots, m\}$, let e_j be an edge connecting a vertex in V_{k-1} to w_j
4: $V_k := V_{k-1} \cup \{w_1, \ldots, w_m\}$
5: $E_k := E_{k-1} \cup \{e_1, \ldots, e_m\}$
6: **return** (V_n, E_n)

Note that the output of this algorithm is not necessarily unique, since the choice of edges at step `3:` in the algorithm is not unique. Figure 1.10 shows an execution of the BFS ALGORITHM.

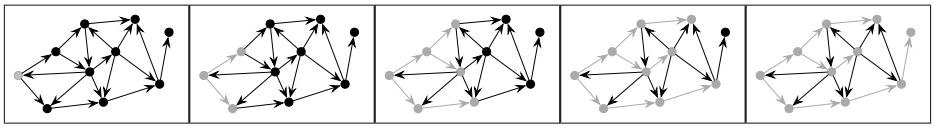


Figure 1.10 Execution of the BFS ALGORITHM. In the leftmost frame, vertex v is colored in gray. The other frames correspond to incremental additions of vertices and edges as specified by the function BFS. The output of the algorithm is a BFS tree of the digraph. The BFS tree is represented in the last frame with vertices and edges colored in gray.

Some properties of the BFS ALGORITHM are characterized as follows.

Lemma 1.28 (BFS tree). *For a digraph G with a vertex v, any digraph T computed by the* BFS ALGORITHM, *$T \in \text{BFS}(G, v)$, has the following properties:*

(i) T is a directed tree with root v;

(ii) T contains a shortest path from v to any other vertex reachable from v inside G, that is, if there is a path in G from v to w, then $w \in T$ and $\text{dist}_G(v, w) = \text{dist}_T(v, w)$; and

(iii) if G contains a spanning tree rooted at v, then T is spanning too and therefore, T is a BFS tree of G.

We leave the proof to the reader. The key property of the algorithm is that (V_k, E_k), $k \in \{1, \ldots, n\}$, is a sequence of directed trees with the property that $(V_k, E_k) \subset (V_{k+1}, E_{k+1})$, for $k \in \{1, \ldots, n-1\}$.

1.4.4.2 The depth-first spanning tree

Next, we define the DFS ALGORITHM that, given a digraph G and a vertex v with $\text{radius}(v, G) < +\infty$, computes what we term a *depth-first spanning (DFS) tree* T_{DFS} rooted at v:

> *[Informal description]* Visit all nodes of the graph recording the traveled edges to form the desired tree. Visit the nodes in the following recursive way: (1) as long as a node has an unvisited child, visit it; (2) when the node has no more unvisited children, then return to its parent (and recursively attempt to visit its unvisited children).

The algorithm is formally stated as a recursive procedure, as follows:

`function` DFS(G, v)
1: $(V_{\text{visited}}, E_{\text{visited}}) := (\{v\}, \emptyset)$
2: DFS-VISIT(G, v)
3: **return** $(V_{\text{visited}}, E_{\text{visited}})$

`function` DFS-VISIT(G, w)
1: **for** u out-neighbor of w **do**
2: **if** u does not belong to V_{visited} **then**
3: $V_{\text{visited}} := V_{\text{visited}} \cup \{u\}$
4: $E_{\text{visited}} := E_{\text{visited}} \cup \{(w, u)\}$
5: DFS-VISIT(G, u)

Note that the output of this algorithm is not necessarily unique, since the order in which the vertices are chosen in step `1:` of DFS-VISIT is not unique. Any digraph T computed by the DFS ALGORITHM, $T \in \text{DFS}(G, v)$, is a directed spanning tree with root v. Figure 1.11 shows an execution of the algorithm.

Some properties of the DFS ALGORITHM are characterized as follows.

Lemma 1.29 (DFS tree). *For a digraph G with a vertex v, any digraph T computed by the* DFS ALGORITHM, *$T \in \text{DFS}(G, v)$, has the following properties:*

(i) T is a directed tree with root v; and

(ii) if G contains a spanning tree rooted at v, then T is spanning too.

Note that both BFS and DFS trees are uniquely defined once a lexicographic order is introduced for the children of a node.

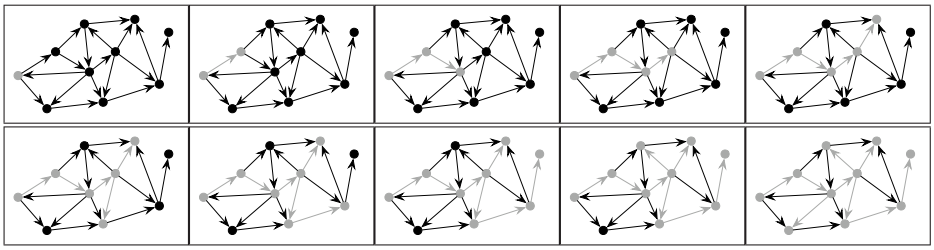


Figure 1.11 Execution of the DFS ALGORITHM. In the top leftmost frame, vertex v is colored in gray. The other frames correspond to incremental additions of vertices and edges as specified by the function DFS. The output of the algorithm is a DFS tree of the digraph. The DFS tree is represented in the last frame with vertices and edges in gray.

1.4.4.3 The shortest-paths tree in weighted digraphs via the Dijkstra algorithm

Finally, we focus on weighted digraphs and on the notion of weighted path length. Given a weighted digraph G of order n with weighted adjacency matrix A and a vertex v with $\text{radius}(v, G) < +\infty$, a *shortest-paths tree* of G with respect to v, denoted $T_{\text{shortest-paths}}$, is a spanning directed tree rooted at v that contains a (weighted) shortest path from v to every other vertex of G. This tree differs from the BFS tree defined above because here the path length is measured using the digraph weights.

We now provide the DIJKSTRA ALGORITHM that, given a digraph G of order n and a vertex v with $\text{radius}(v, G) < +\infty$, computes a shortest-paths tree $T_{\text{shortest-paths}}$ rooted at v:

> *[Informal description]* Incrementally construct a tree that contains only shortest paths. In each round, add to the tree (1) the node that is closest to the source and is not yet in the tree, and (2) the edge corresponding to the shortest path. The weighted distance to the source (required to perform step (1)) is computed via an array of distance estimates that is updated as follows: when a node is added to the tree, the distance estimates of all its out-neighbors are updated.

The algorithm is formally stated as follows:

`function` DIJKSTRA$\big((V, E, A), v\big)$

1: $T_{\text{shortest-paths}} := \emptyset$
   % Initialize estimated distances and estimated parent nodes
2: **for** $u \in V$ **do**

3: $\texttt{dist}(u) := \begin{cases} 0, & u = v, \\ +\infty, & \text{otherwise.} \end{cases}$
4: $\texttt{parent}(u) := u$
% Main loop to grow the tree and update estimates
5: **while** ($T_{\text{shortest-paths}}$ does not contain all vertices) **do**
6: find vertex u outside $T_{\text{shortest-paths}}$ with smallest $\texttt{dist}(u)$
7: add to $T_{\text{shortest-paths}}$ the vertex u
8: if $u = v$, add to $T_{\text{shortest-paths}}$ the edge $(\texttt{parent}(u), u)$
9: **for** each node w that is an out-neighbor of u in (V, E, A) **do**
10: **if** $\texttt{dist}(w) > \texttt{dist}(u) + a_{uw}$ **then**
11: $\texttt{dist}(w) := \texttt{dist}(u) + a_{uw}$
12: $\texttt{parent}(w) := u$
13: **return** $T_{\text{shortest-paths}}$

Note that the output of this algorithm is not necessarily unique, since the choice of vertex at step 6: in the algorithm is not unique. Figure 1.12 shows an execution of the the DIJKSTRA ALGORITHM.

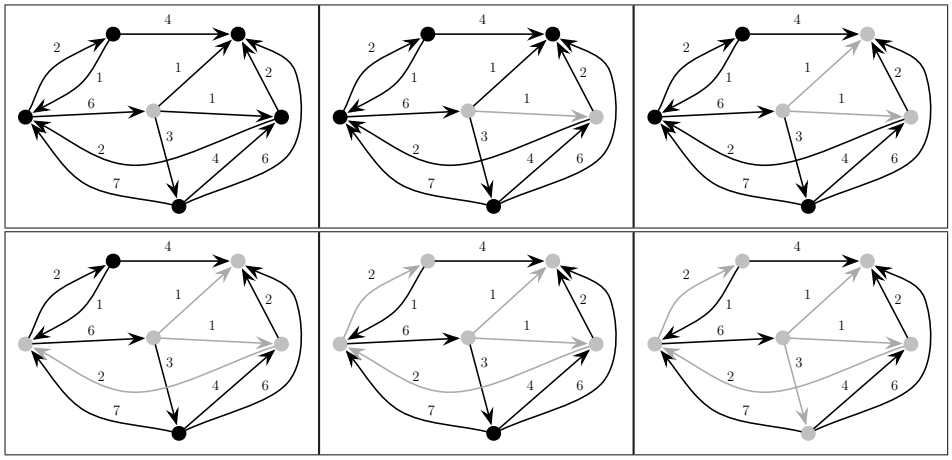


Figure 1.12 Execution of the DIJKSTRA ALGORITHM on the weighted digraph plotted in Figure 1.9. In the top leftmost frame, vertex v is colored in gray. The other frames correspond to incremental additions of vertices and edges as specified by the function DIJKSTRA. The output of the algorithm is a shortest-paths tree of the digraph rooted at v. This tree is represented in the last frame with vertices and edges colored in gray.

The following properties of the DIJKSTRA ALGORITHM mirror those of the BFS ALGORITHM in Lemma 1.28.

Lemma 1.30 (Dijkstra algorithm). *For a weighted digraph G with a vertex v, any digraph T computed by the* DIJKSTRA ALGORITHM*, $T \in$ DIJKSTRA(G, v), has the following properties:*

(i) *T is a directed tree with root v;*

(ii) *T contains a shortest path from v to any other vertex reachable from v inside G, that is, if there is a path in G from v to w, then $w \in T$ and* $\mathrm{wdist}_G(v, w) = \mathrm{wdist}_T(v, w)$*; and*

(iii) *if G contains a spanning tree rooted at v, then T is spanning too, and therefore, T is a shortest-paths tree of G.*

1.4.4.4 On combinatorial optimization problems

We conclude this section on graph algorithms with a brief mention of classic optimization problems defined on graphs. Standard references on combinatorial optimization include Vazirani (2001) and Korte and Vygen (2005). Given a weighted directed graph G, classical combinatorial optimization problems include the following:

Minimum-weight spanning tree. A *minimum-weight spanning tree* of G, denoted MST, is a spanning tree with the minimum possible weight. In order for the MST to exist, G must contain a spanning tree. If all the weights of the individual edges are different, then the MST is unique.

Traveling salesperson problem. A *traveling salesperson tour* of G, denoted TSP, is a cycle that passes through all the nodes of the digraph and has the minimum possible weight. In order for the TSP to exist, G must contain a cycle through all nodes.

Multicenter optimization problems. Given a weighted digraph G with vertices $V = \{v_1, \ldots, v_n\}$ and a set $U = \{u_1, \ldots, u_k\} \subset V$, the weighted distance from $v \in V$ to the set U is the smallest weighted distance from v to any vertex in $\{u_1, \ldots, u_k\}$. We now consider the cost functions $\mathcal{H}_{\max}, \mathcal{H}_{\Sigma} : V^k \to \mathbb{R}$ defined by

$$\mathcal{H}_{\max}(u_1, \ldots, u_k) = \max_{i \in \{1,\ldots,n\}} \min_{h \in \{1,\ldots,k\}} \mathrm{wdist}_G(v_i, u_h),$$

$$\mathcal{H}_{\Sigma}(u_1, \ldots, u_k) = \sum_{i=1}^{n} \min_{h \in \{1,\ldots,k\}} \mathrm{wdist}_G(v_i, u_h).$$

The *k-center problem* and the *k-median problem* consist of finding a set of vertices $\{u_1, \ldots, u_k\}$ that minimizes the *k-center function* $\mathcal{H}_{\max}$ and the *k-median function* $\mathcal{H}_{\Sigma}$, respectively. We refer to Vazirani (2001) for a discussion of the k-center and k-median problems (as well as the more general *uncapacited facility location* problem) over complete undirected graphs with edge costs satisfying the triangle inequality.

The Euclidean versions of these combinatorial optimization problems refer to the situation where one considers a weighted complete digraph whose vertex set is a point set in $\mathbb{R}^d$, $d \in \mathbb{N}$, and whose weight map assigns to each edge the Euclidean distance between the two nodes connected by the edge.

1.4.5 Algebraic graph theory

Algebraic graph theory (Biggs, 1994; Godsil and Royle, 2001) is the study of matrices defined by digraphs: in this section, we expose two topics. First, we study the equivalence between properties of graphs and of their associated adjacency matrices. We also specify how to associate a digraph to a nonnegative matrix. Second, we introduce and characterize the Laplacian matrix of a weighted digraph.

We begin by studying adjacency matrices. Note that the adjacency matrix of a weighted digraph is nonnegative and, in general, not stochastic. The following lemma expands on this point.

Lemma 1.31 (Weight-balanced digraphs and doubly stochastic adjacency matrices). *Let G be a weighted digraph of order n with weighted adjacency matrix A and weighted out-degree matrix D_{out}. Define the matrix*

$$F = \begin{cases} D_{\mathrm{out}}^{-1}A, & \text{if each out-degree is strictly positive,} \\ (I_n + D_{\mathrm{out}})^{-1}(I_n + A), & \text{otherwise.} \end{cases}$$

Then

(i) F is row-stochastic; and

(ii) F is doubly stochastic if G is weight-balanced and the weighted degree is constant for all vertices.

Proof. Consider first the case when each vertex has an outgoing edge so that D_{out} is invertible. We first note that $\mathrm{diag}(v)^{-1}v = \mathbf{1}_n$, for each $v \in (\mathbb{R}\backslash\{0\})^n$. Therefore

$$\big(D_{\mathrm{out}}^{-1}A\big)\mathbf{1}_n = \mathrm{diag}(A\mathbf{1}_n)^{-1}\big(A\mathbf{1}_n\big) = \mathbf{1}_n,$$

which proves (i). Furthermore, if $D_{\mathrm{out}} = D_{\mathrm{in}} = dI_n$ for some $d \in \mathbb{R}_{>0}$, then

$$\big(D_{\mathrm{out}}^{-1}A\big)^T\mathbf{1}_n = \frac{1}{d}\big(A^T\mathbf{1}_n\big) = D_{\mathrm{in}}^{-1}\big(A^T\mathbf{1}_n\big) = \mathrm{diag}(A^T\mathbf{1}_n)^{-1}\big(A^T\mathbf{1}_n\big) = \mathbf{1}_n,$$

which proves (ii). Finally, if (V, E, A) does not have outgoing edges at each vertex, then apply the statement to the weighted digraph $(V, E \cup \{(i,i) \mid i \in \{1,\dots,n\}\}, A + I_n)$. ∎

The next result characterizes the relationship between the adjacency matrix and directed paths in the digraph.

Lemma 1.32 (Directed paths and powers of the adjacency matrix). *Let G be a weighted digraph of order n with weighted adjacency matrix A, with unweighted adjacency matrix $A_{0,1} \in \{0,1\}^{n\times n}$, and possibly with self-loops. For all $i, j, k \in \{1, \ldots, n\}$*

(i) the (i, j) entry of $A_{0,1}^k$ equals the number of directed paths of length k (including paths with self-loops) from node i to node j; and

(ii) the (i, j) entry of A^k is positive if and only if there exists a directed path of length k (including paths with self-loops) from node i to node j.

Proof. The second statement is a direct consequence of the first. The first statement is proved by induction. The statement is clearly true for $k = 1$. Next, we assume the statement is true for $k \geq 1$ and we prove it for $k + 1$. By assumption, the entry $(A^k)_{ij}$ equals the number of directed paths from i to j of length k. Note that each path from i to j of length $k + 1$ identifies (1) a unique node ℓ such that (i, ℓ) is an edge of G and (2) a unique path from ℓ to j of length k. We write $A^{k+1} = AA^k$ in components as

$$(A^{k+1})_{ij} = \sum_{\ell=1}^{n} A_{i\ell}(A^k)_{\ell j}.$$

Therefore, it is true that the entry $(A^{k+1})_{ij}$ equals the number of directed paths from i to j of length $k+1$. This concludes the induction argument. ■

The following proposition characterizes in detail the relationship between various connectivity properties of the digraph and algebraic properties of the adjacency matrix. The result is illustrated in Figure 1.13 and its proof is postponed until Section 1.8.3.

Proposition 1.33 (Connectivity properties of the digraph and positive powers of the adjacency matrix). *Let G be a weighted digraph of order n with weighted adjacency matrix A. The following statements are equivalent:*

(i) G is strongly connected;

(ii) A is irreducible; and

(iii) $\sum_{k=0}^{n-1} A^k$ is positive.

For any $j \in \{1, \ldots, n\}$, the following two statements are equivalent:

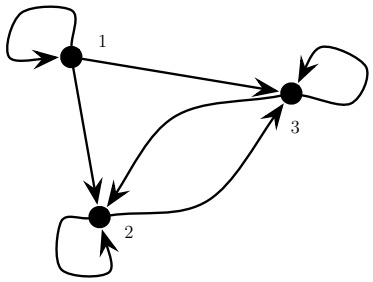


Figure 1.13 An illustration of Proposition 1.33. Even though vertices 2 and 3 are globally reachable, the digraph is not strongly connected because vertex 1 has no in-neighbor other than itself. Therefore, the associated adjacency matrix $A = (a_{ij})$ with $(a_{1j}) = \mathbf{1}_3$, $(a_{2j}) = (a_{3j}) = (0, 1, 1)$, is reducible.

(iv) the jth node of G is globally reachable; and

(v) the jth column of $\sum_{k=0}^{n-1} A^k$ has positive entries.

Stronger statements can be given for digraphs with self-loops.

Proposition 1.34 (Connectivity properties of the digraph and positive powers of the adjacency matrix: cont'd). *Let G be a weighted digraph of order n with weighted adjacency matrix A and with self-loops at each node. The following statements are equivalent:*

(iv) G is strongly connected; and

(v) A^{n-1} is positive.

For any $j \in \{1, \dots, n\}$, the following two statements are equivalent:

(iv) the jth node of G is globally reachable; and

(v) the jth column of A^{n-1} has positive entries.

Next, we characterize the relationship between irreducible aperiodic digraphs and primitive matrices (recall Definition 1.12). We will postpone the proof to Section 1.8.3.

Proposition 1.35 (Strongly connected and aperiodic digraph and primitive adjacency matrix). *Let G be a weighted digraph of order n with weighted adjacency matrix A. The following two statements are equivalent:*

(i) G is strongly connected and aperiodic; and

(ii) A is primitive, that is, there exists $k \in \mathbb{N}$ such that A^k is positive.

This concludes our study of adjacency matrices associated to weighted digraphs. Next, we emphasize how all results obtained so far have analogs that hold when the original object is a nonnegative matrix, instead of a weighted digraph.

Remark 1.36 (From a nonnegative matrix to its associated digraphs). Given a nonnegative $n \times n$ matrix A, its *associated weighted digraph* is the weighted digraph with nodes $\{1, \ldots, n\}$, and weighted adjacency matrix A. The unweighted version of this weighted digraph is called the *associated digraph*. The following statements are analogs of the previous lemmas:

(i) if A is stochastic, then its associated digraph has weighted out-degree matrix equal to I_n;

(ii) if A is doubly stochastic, then its associated weighted digraph is weight-balanced and, additionally, both in-degree and out-degree matrices are equal to I_n; and

(iii) A is irreducible if and only if its associated weighted digraph is strongly connected. •

So far, we have analyzed in detail the properties of adjacency matrices. We conclude this section by studying a second relevant matrix associated to a digraph, called the Laplacian matrix. The *Laplacian matrix* of the weighted digraph G is

$$L(G) = D_{\text{out}}(G) - A(G).$$

Some immediate consequences of this definition are the following:

(i) $L(G)\mathbf{1}_n = \mathbf{0}_n$, that is, 0 is an eigenvalue of $L(G)$ with eigenvector $\mathbf{1}_n$;

(ii) G is undirected if and only if $L(G)$ is symmetric; and

(iii) $L(G)$ equals the Laplacian matrix of the digraph obtained by adding to or removing from G any self-loop with arbitrary weight.

Further properties are established as follows.

Theorem 1.37 (Properties of the Laplacian matrix). *Let G be a weighted digraph of order n. The following statements hold:*

(i) all eigenvalues of $L(G)$ have nonnegative real part (thus, if G is undirected, then $L(G)$ is symmetric positive semidefinite);

(ii) if G is strongly connected, then $\operatorname{rank}(L(G)) = n - 1$*, that is, 0 is a simple eigenvalue of $L(G)$;*

(iii) *G contains a globally reachable vertex if and only if* $\operatorname{rank}(L(G)) = n - 1$*;*

(iv) *the following three statements are equivalent:*

(a) *G is weight-balanced;*

(b) $\mathbf{1}_n^T L(G) = \mathbf{0}_n^T$*; and*

(c) *$L(G) + L(G)^T$ is positive semidefinite.*

1.5 DISTRIBUTED ALGORITHMS ON SYNCHRONOUS NETWORKS

Here, we introduce a synchronous network as a group of processors with the ability to exchange messages and perform local computations. What we present is a basic classic model studied extensively in the distributed algorithms literature. Our treatment is directly adopted with minor variations, from the texts by Lynch (1997) and Peleg (2000).

1.5.1 Physical components and computational models

Loosely speaking, a synchronous network is a group of processors, or nodes, that possess a local state, exchange messages along the edges of a digraph, and compute an update to their local state based on the received messages. Each processor alternates the two tasks of exchanging messages with its neighboring processors and of performing a computation step. Let us begin by describing what constitutes a network.

Definition 1.38 (Network). The physical component of a *synchronous network* $\mathcal{S}$ is a digraph (I, E_{cmm}), where:

(i) $I = \{1, \ldots, n\}$ is called the *set of unique identifiers (UIDs)*; and

(ii) E_{cmm} is a set of directed edges over the vertices $\{1, \ldots, n\}$, called the communication links. •

In general, the set of unique identifiers does not need to be n consecutive natural numbers, but we adopt this convention for simplicity. The set E_{cmm} models the topology of the communication service among the nodes: for $i, j \in \{1, \ldots, n\}$, processor i can send a message to processor j if the directed edge (i, j) is present in E_{cmm}. Note that, unlike the standard treatments in Lynch (1997) and Peleg (2000), we do not assume the digraph to be strongly connected; the required connectivity assumption will be specified on a case-by-case basis.

Next, we discuss the state and the algorithms that each processor possesses and executes, respectively. By convention, we let the superscript $[i]$ denote any quantity associated with the node i.

Definition 1.39 (Distributed algorithm). A *distributed algorithm* $\mathcal{DA}$ for a network $\mathcal{S}$ consists of the sets

(i) $\mathbb{A}$, a set containing the `null` element, called the *communication alphabet*—elements of $\mathbb{A}$ are called *messages*;

(ii) $W^{[i]}$, $i \in I$, called the *processor state sets*; and

(iii) $W_0^{[i]} \subseteq W^{[i]}$, $i \in I$, sets of *allowable initial values*;

and of the maps

(i) $\mathrm{msg}^{[i]} : W^{[i]} \times I \to \mathbb{A}$, $i \in I$, called *message-generation functions*; and

(ii) $\mathrm{stf}^{[i]} : W^{[i]} \times \mathbb{A}^n \to W^{[i]}$, $i \in I$, called *state-transition functions*.

If $W^{[i]} = W$, $\mathrm{msg}^{[i]} = \mathrm{msg}$, and $\mathrm{stf}^{[i]} = \mathrm{stf}$ for all $i \in I$, then $\mathcal{DA}$ is said to be *uniform* and is described by a tuple $(\mathbb{A}, W, \{W_0^{[i]}\}_{i \in I}, \mathrm{msg}, \mathrm{stf})$. •

Now, with all elements in place, we can explain in more detail how a synchronous network executes a distributed algorithm (see Figure 1.14). The *state* of processor i is a variable $w^{[i]} \in W^{[i]}$, initially set equal to an

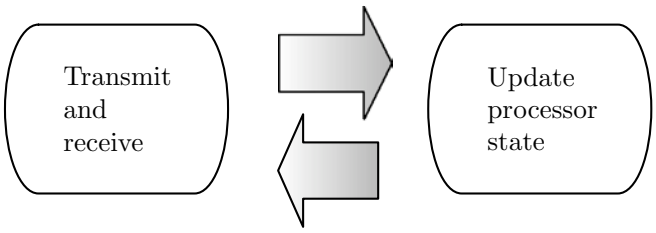


Figure 1.14 The execution of a distributed algorithm by a synchronous network.

allowable value in $W_0^{[i]}$. At each time instant $\ell \in \mathbb{Z}_{\geq 0}$, processor i sends to each of its out-neighbors j in the communication digraph (I, E_{cmm}) a message (possibly the `null` message) computed by applying the message-generation function $\mathrm{msg}^{[i]}$ to the current values of its state $w^{[i]}$ and to the identity j. Subsequently, but still at time instant $\ell \in \mathbb{Z}_{\geq 0}$, processor i updates the value of its state $w^{[i]}$ by applying the state-transition function $\mathrm{stf}^{[i]}$ to the current value of $w^{[i]}$ and to the messages it receives from its in-neighbors. At each round, the first step is transmission and the second one is computation. These notions are formalized in the following definition.

Definition 1.40 (Network evolution). Let $\mathcal{DA}$ be a distributed algorithm for the network $\mathcal{S}$. The *evolution* of $(\mathcal{S}, \mathcal{DA})$ from initial conditions $w_0^{[i]} \in W_0^{[i]}$, $i \in I$, is the collection of trajectories $w^{[i]} : \mathbb{Z}_{\geq 0} \to W^{[i]}$, $i \in I$, satisfying

$$w^{[i]}(\ell) = \mathrm{stf}^{[i]}(w^{[i]}(\ell - 1), y^{[i]}(\ell)),$$

where $w^{[i]}(-1) = w_0^{[i]}$, $i \in I$, and where the trajectory $y^{[i]} : \mathbb{Z}_{\geq 0} \to \mathbb{A}^n$ (describing the messages received by processor i) has components $y_j^{[i]}(\ell)$, for $j \in I$, given by

$$y_j^{[i]}(\ell) = \begin{cases} \mathrm{msg}^{[j]}(w^{[j]}(\ell - 1), i), & \text{if } (j, i) \in E_{\mathrm{cmm}}, \\ \texttt{null}, & \text{otherwise.} \end{cases}$$

Let $\ell \mapsto w(\ell) = (w^{[1]}(\ell), \ldots, w^{[n]}(\ell))$ denote the collection of trajectories. •

We conclude this section with two sets of remarks. We first discuss some aspects of our communication model that have a large impact on the subsequent development. We then collect a few general comments about control structures and failure modes relevant in the study of distributed algorithms on networks.

Remarks 1.41 (Aspects of the communication model).

(i) The network $\mathcal{S}$ and the algorithm $\mathcal{DA}$ are referred to as *synchronous* because the communications between all processors takes place at the same time for all processors.

(ii) Communication is modeled as a so-called "point-to-point" service: a processor can specify different messages for different out-neighbors and knows the processor identity corresponding to any incoming message.

(iii) Information is exchanged between processors as messages, that is, elements of the alphabet $\mathbb{A}$; the message `null` indicates no communication. Messages might encode logical expressions such as `true` and `false`, or finite-resolution quantized representations of integer and real numbers.

(iv) In some uniform algorithms, the messages between processors are the processors' states. In such cases, the corresponding communication alphabet is $\mathbb{A} = W \cup \{\texttt{null}\}$ and the message-generation function $\mathrm{msg}_{\mathrm{std}}(w, j) = w$ is referred to as the *standard message-generation function.* •

Remarks 1.42 (Advanced topics: Control structures and failures).

(i) Processors in a network have only partial information about the network topology. In general, each processor only knows its own UID, and the UID of its in- and out-neighbors. Sometimes we will assume that the processor knows the network diameter. In some cases (Peleg, 2000), actively running networks might depend upon "control structures," that is, structures that are computed at initial time and are exploited in subsequent algorithms. For example, routing tables might be computed for routing problems, "leader" processors might be elected, and tree structures might be computed and represented in a distributed manner for various tasks; for example, coloring or maximal independent set problems. We present some sample algorithms to compute these structures below.

(ii) A key issue in the study of distributed algorithms is the possible occurrence of failures. A network might experience intermittent or permanent communication failures: along given edges, a `null` message or an arbitrary message might be delivered instead of the intended value. Alternatively, a network might experience various types of processor failures: a processor might transmit only `null` messages (i.e., the msg function always returns `null`), a processor might quit updating its state (i.e., the stf function neglects incoming messages and returns the current state value), or a processor might implement arbitrarily modified msg and stf functions. The latter situation, in which completely arbitrary and possibly malicious behavior is adopted by faulty nodes, is referred to as a Byzantine failure in the distributed algorithms literature. •

1.5.2 Complexity notions

Here, we begin our analysis of the performance of distributed algorithms. We introduce a notion of algorithm completion and, in turn, we introduce the classic notions of time, space, and communication complexity.

Definition 1.43 (Algorithm completion). We say that an algorithm *terminates* when only `null` messages are transmitted and all processors' states become constants. •

Remarks 1.44 (Alternative termination notions).

(i) In the interest of simplicity, we have defined evolutions to be unbounded in time and we do not explicitly require algorithms to

actually have termination conditions, that is, to be able to detect when termination takes place.

(ii) It is also possible to define the termination time as the first instant when a given problem or task is achieved, independently of the fact that the algorithm might continue to transmit data subsequently. •

Definition 1.45 (Time complexity). The *(worst-case) time complexity* of a distributed algorithm $\mathcal{DA}$ on a network $\mathcal{S}$, denoted $\mathrm{TC}(\mathcal{DA})$, is the maximum number of rounds required by the execution of $\mathcal{DA}$ on $\mathcal{S}$ among all allowable initial states until termination. •

Next, it is of interest to quantify the memory and communication requirements of distributed algorithms. From an information theory viewpoint (Gallager, 1968), the information content of a memory variable or of a message is properly measured in bits. On the other hand, it is convenient to use the alternative notions of "basic memory unit" and "basic message." It is customary (Peleg, 2000) to assume that a "basic memory unit" or a "basic message" contains $\log(n)$ bits; so that, for example, the information content of a robot identifier $i \in \{1, \ldots, n\}$ is $\log(n)$ bits or, equivalently, one "basic memory unit." Note that elements of the processor state set W or of the alphabet set $\mathbb{A}$ might amount to multiple basic memory units or basic messages; the `null` message has zero cost. Unless specified otherwise, the following definitions and examples are stated in terms of basic memory units and basic messages.

Definition 1.46 (Space complexity). The *(worst-case) space complexity* of a distributed algorithm $\mathcal{DA}$ on a network $\mathcal{S}$, denoted by $\mathrm{SC}(\mathcal{DA})$, is the maximum number of basic memory units required by a processor executing $\mathcal{DA}$ on $\mathcal{S}$ among all processors and among all allowable initial states until termination. •

Remark 1.47 (Space complexity conventions). By convention, each processor knows its identity, that is, it requires $\log(n)$ bits to represent its unique identifier in a set with n distinct elements. We do not count this cost in the space complexity of an algorithm. •

Next, we introduce a notion of communication complexity.

Definition 1.48 (Communication complexity). The *(worst-case) communication complexity* of a distributed algorithm $\mathcal{DA}$ on a network $\mathcal{S}$, denoted by $\mathrm{CC}(\mathcal{DA})$, is the maximum number of basic messages transmitted over the entire network during the execution of $\mathcal{DA}$ among all allowable initial states until termination. •

We conclude this section by discussing ways of quantifying time, space and communication complexity. The idea, borrowed from combinatorial optimization, is to adopt asymptotic "order of magnitude" measures. Formally, complexity bounds will be expressed with respect to the Bachmann–Landau symbols O, Ω and Θ defined in Section 1.1. Let us be more specific:

(i) we will say that an algorithm has time complexity *of order* $\Omega(f(n))$ *over some network* if, for all n, there exists a network of order n and initial processor values such that the time complexity of the algorithm is greater than a constant factor times $f(n)$;

(ii) we will say that an algorithm has time complexity *of order* $O(f(n))$ *over arbitrary networks* if, for all n, for all networks of order n and for all initial processor values, the time complexity of the algorithm is lower than a constant factor times $f(n)$; and

(iii) we will say that an algorithm has time complexity *of order* $\Theta(f(n))$ if its time complexity is of order $\Omega(f(n))$ over some network and $O(f(n))$ over arbitrary networks at the same time.

Similar conventions will be used for space and communication complexity.

In many cases, the complexity of an algorithm will typically depend upon the number of vertices of the network. It is therefore useful to present a few simple facts about these functions now. Over arbitrary digraphs $\mathcal{S} = (I, E_{\mathrm{cmm}})$ of order n, we have

$$\mathrm{diam}(\mathcal{S}) \in \Theta(n), \quad |E_{\mathrm{cmm}}(\mathcal{S})| \in \Theta(n^2) \quad \text{and} \quad \mathrm{radius}(v, \mathcal{S}) \in \Theta(\mathrm{diam}(\mathcal{S})),$$

where v is any vertex of $\mathcal{S}$.

Remark 1.49 (Additional complexity notions). Numerous variations of the proposed complexity notions are possible and may be of interest.

Global lower bounds. In the definition of lower bound, consider the logic quantifier describing the role of the network. The lower bound statement is "existential" rather than "global," in the sense that the bound does not hold for all graphs. As discussed in Peleg (2000), it is possible to define also "global" lower bounds, that is, lower bounds over all graphs, or lower bounds over specified classes of graphs.

Average complexity notions. The proposed complexity notions focus on the worst-case situation. It is possible to define *expected* or *average* complexity notions, where one is interested in characterizing, for example, the average number of rounds required or the average number of basic messages transmitted over the entire network during the algorithm execution among all allowable initial states until termination.

Problem complexity. It is possible to define complexity notions for problems, rather than algorithms, by considering, for example, the worst-case optimal performance among all algorithms that solve the given problem, or over classes of algorithms or classes of graphs. •

1.5.3 Broadcast and BFS tree computation

In the following, we consider some basic algorithmic problems such as the simple one-to-all communication task—that is, broadcasting—and the establishment of some "control structures" (see Remarks 1.42), such as the construction of a BFS spanning tree and the election of a leader.

Problem 1.50 (Broadcast). Assume that a processor, called the *source*, has a message, called the *token*. Transmit the token to all others processors in the network. •

Note that existence of a spanning tree rooted at the source is a necessary requirement for the broadcast problem to be solvable. We begin by establishing some analysis results for the broadcast problem.

Lemma 1.51 (Complexity lower bounds for the broadcast problem). *Let $\mathcal{S}$ be a network containing a spanning tree rooted at v. The broadcast problem for $\mathcal{S}$ from the source v has communication complexity lower bounded by $n-1$ and time complexity lower bounded by* $\mathrm{radius}(v, \mathcal{S})$.

In what follows, we shall solve the broadcast problem while simultaneously also considering the following problem.

Problem 1.52 (BFS tree computation). Let $\mathcal{S}$ be a network containing a spanning tree rooted at v. Compute a distributed representation for a BFS tree rooted at v. •

We add two remarks on the BFS tree computation problem:

(i) By a distributed representation of a directed tree with bounded memory at each node, we mean the following: each child vertex knows the identity of its parent and the root vertex knows that it has no parents. A more informative structure would require each parent to know the identity of its children; this is easy to achieve on undirected digraphs.

(ii) The BFS tree computation has the same lower bounds as the broadcast problem.

An elegant and classic solution to the broadcast and BFS tree computation problems is given by the FLOODING ALGORITHM. This algorithm implements the same "breadth-first search" mechanism of the (centralized) BFS ALGORITHM characterized in Lemma 1.28:

> *[Informal description]* The source broadcasts the token to its out-neighbors. In each communication round, each node determines whether it has received a non-`null` message from one of its in-neighbors. When a non-`null` message is received—that is, the token is received—the node performs two actions. First, the node stores the token in the variable `data` (this solves the Broadcast problem). Second, the node stores the identity of one of the transmitting in-neighbors in the variable `parent` (this solves the BFS tree computation problem). Specifically, if the message is received simultaneously from multiple in-neighbors, then the node stores the smallest among the identities of the transmitting in-neighbors. In the subsequent communication round, the node broadcasts the token to its out-neighbors.

To formally describe the algorithm, we assume that the node with the message to be broadcast is $v = 1$. Also, we assume that the token is a letter of the Greek alphabet $\{\alpha, \ldots, \omega\}$:

`Synchronous Network:` $\mathcal{S} = (\{1, \ldots, n\}, E_{\text{cmm}})$

`Distributed Algorithm:` FLOODING

`Alphabet:` $\mathbb{A} = \{\alpha, \ldots, \omega\} \cup \texttt{null}$

`Processor State:` $w = (\texttt{parent}, \texttt{data}, \texttt{snd-flag})$, where

`parent`	$\in \{0, \ldots, n\}$,	initially:	$\texttt{parent}^{[1]} = 1$,
			$\texttt{parent}^{[j]} = 0$ for all $j = 1$
`data`	$\in \mathbb{A}$,	initially:	$\texttt{data}^{[1]} = \mu$,
			$\texttt{data}^{[j]} = \texttt{null}$ for all $j = 1$
`snd-flag`	$\in \{\texttt{false}, \texttt{true}\}$,	initially:	$\texttt{snd-flag}^{[1]} = \texttt{true}$,
			$\texttt{snd-flag}^{[j]} = \texttt{false}$ for $j = 1$

`function` msg(w, i)

```
1: if (parent = i) AND (snd-flag = true) then
2:     return data
3: else
4:     return null
```

`function` stf(w, y)

```
1: case
2:   (data = null) AND (y contains only null messages):
       % The node has not yet received the token
3:     new-parent := null
4:     new-data := null
5:     new-snd-flag := false
6:   (data = null) AND (y contains a non-null message):
       % The node has just received the token
7:     new-parent := smallest UID among transmitting in-neighbors
8:     new-data := a non-null message
9:     new-snd-flag := true
10:  (data = null):
       % If the node already has the token, then do not re-broadcast it
11:    new-parent := parent
12:    new-data := data
13:    new-snd-flag := false
14: return (new-parent, new-data, new-snd-flag)
```

An execution of the FLOODING ALGORITHM is shown in Figure 1.15.

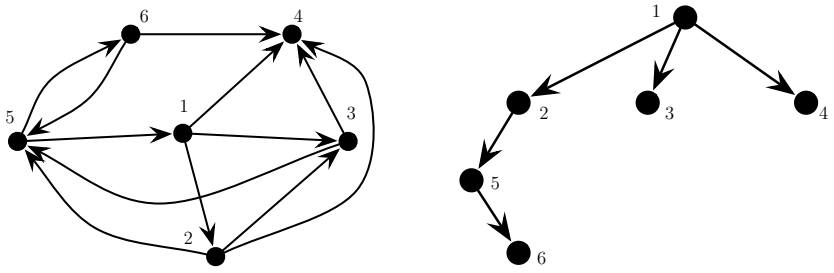


Figure 1.15 An example execution of the FLOODING ALGORITHM. The source is vertex 1: (a) shows the network and (b) shows the BFS tree that results from the execution.

This algorithm can analyzed by induction: one can show that, for $d \in \{1, \ldots, \text{radius}(v, \mathcal{S})\}$, every node at a distance d from the root receives a non-`null` message at round d. A summary of the results is given as follows.

Lemma 1.53 (Complexity upper bounds for the flooding algorithm). *For a network $\mathcal{S}$ containing a spanning tree rooted at v, the* FLOODING ALGORITHM *has communication complexity in $\Theta(|E_{\text{cmm}}|)$, time complexity in $\Theta(\text{radius}(v, \mathcal{S}))$, and space complexity in $\Theta(1)$.*

We conclude the section with a final remark.

Remark 1.54 (Termination condition for the flooding algorithm). As presented, the FLOODING ALGORITHM does not include a termination condition, that is, the processors do not have a mechanism to detect when the broadcast and tree computation are complete. If an upper bound on the graph diameter is known, then it is easy to design a termination condition based on this information; we do this in the next subsection. If no *a priori* knowledge is available, then one can design more sophisticated algorithms for networks with stronger connectivity properties. We refer to Lynch (1997) and Peleg (2000) for a complete discussion about this. •

1.5.4 Leader election

Next, we formulate another interesting problem for a network.

Problem 1.55 (Leader election). Assume that all processors of a network have a state variable, say `leader`, initially set to `unknwn`. We say that a leader is elected when one and only one processor has the state variable set to `true` and all others have it set to `false`. Elect a leader. •

This task that is a bit more global in nature. We display here a solution that requires individual processors to know the diameter of the network, denoted by $\mathrm{diam}(\mathcal{S})$, or an upper bound on it:

> *[Informal description]* In each communication round, each agent sends to its out-neighbors the maximum UID it has received up to that time. This is repeated for $\mathrm{diam}(\mathcal{S})$ rounds. At the last round, each agent compares the maximum received UID with its own, and declares itself a leader if they coincide, or a non-leader otherwise.

The algorithm is called the FLOODMAX ALGORITHM: the maximum UID in the network is transmitted to other agents in an incremental fashion. At the first communication round, agents that are neighbors of the agent with the maximum UID receive the message from it. At the next communication round, the neighbors of these agents receive the message with the maximum UID. This process goes on for $\mathrm{diam}(\mathcal{S})$ rounds, to ensure that every agent receives the maximum UID. Note that there are networks for which all agents receive the message with the maximum UID in fewer communication rounds than $\mathrm{diam}(\mathcal{S})$. The algorithm is formally stated as follows:

`Synchronous Network:` $\mathcal{S} = (\{1, \ldots, n\}, E_{\mathrm{cmm}})$

`Distributed Algorithm:` FLOODMAX

`Alphabet`: $\mathbb{A} = \{1, \ldots, n\} \cup \{\texttt{null}\}$

`Processor State`: $w = (\texttt{my-id}, \texttt{max-id}, \texttt{leader}, \texttt{round})$, where

`my-id`	$\in \{1, \ldots, n\}$,	initially: $\texttt{my-id}^{[i]} = i$ for all i
`max-id`	$\in \{1, \ldots, n\}$,	initially: $\texttt{max-id}^{[i]} = i$ for all i
`leader`	$\in \{\texttt{false}, \texttt{true}, \texttt{unknwn}\}$,	initially: $\texttt{leader}^{[i]} = \texttt{unknwn}$ for all i
`round`	$\in \{0, 1, \ldots, \mathrm{diam}(\mathcal{S})\}$,	initially: $\texttt{round}^{[i]} = 0$ for all i

`function` $\mathrm{msg}(w, i)$

1: **if** `round` $< \mathrm{diam}(\mathcal{S})$ **then**
2: **return** `max-id`
3: **else**
4: **return** `null`

`function` $\mathrm{stf}(w, y)$

1: `new-id`:= $\max\{$`max-id`, largest identifier in $y\}$
2: **case**
3: `round` $< \mathrm{diam}(\mathcal{S})$: `new-lead` := `unknwn`
4: `round` $= \mathrm{diam}(\mathcal{S})$ AND `max-id` $=$ `my-id`: `new-lead` := `true`
5: `round` $= \mathrm{diam}(\mathcal{S})$ AND `max-id` $>$ `my-id`: `new-lead` := `false`
6: **return** (`my-id`, `new-id`, `new-lead`, `round` $+1$)

Figure 1.16 shows an execution of the FLOODMAX ALGORITHM. Some properties of this algorithm are characterized in the following lemma. A complete analysis of this algorithm, including modifications to improve the communication complexity, is discussed in Lynch (1997, Section 4.1).

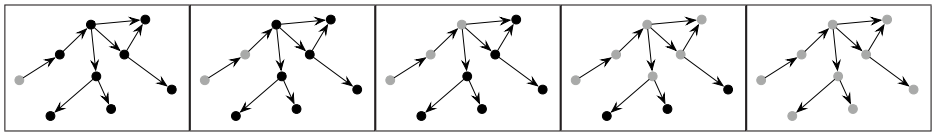


Figure 1.16 Execution of the FLOODMAX ALGORITHM. The diameter of the network is 4. In the leftmost frame, the agent with the maximum UID is colored in gray. After 4 communication rounds, its message has been received by all agents.

Lemma 1.56 (Complexity upper bounds for the floodmax algorithm). *For a network $\mathcal{S}$ containing a spanning tree, the* FLOODMAX ALGORITHM *has communication complexity in $O(\mathrm{diam}(\mathcal{S})|E_{\mathrm{cmm}}|)$, time complexity equal to $\mathrm{diam}(\mathcal{S})$, and space complexity in $\Theta(1)$.*

A simplification of the FLOODMAX ALGORITHM leads to the Le Lann–Chang–Roberts algorithm (or LCR ALGORITHM in short) for leader election in rings, see (Lynch, 1997, Chapter 3.3), which we describe next. The

LCR ALGORITHM runs on a ring digraph and does not require the agents to know the diameter of the network. We provide an informal and a formal description of the algorithm.

> *[Informal description]* In each communication round, each agent sends to its neighbors the maximum UID it has received up to that time. (Agents do not record the number of communication rounds.) When the agent with the maximum UID receives its own UID from an in-neighbor, it declares itself the leader.

`Synchronous Network:` ring digraph

`Distributed Algorithm:` LCR

`Alphabet:` $\mathbb{A} = \{1, \ldots, n\} \cup \{\texttt{null}\}$

`Processor State:` $w = (\texttt{my-id}, \texttt{max-id}, \texttt{leader}, \texttt{snd-flag})$, where

`my-id`	$\in \{1, \ldots, n\}$,	initially: $\texttt{my-id}^{[i]} = i$ for all i
`max-id`	$\in \{1, \ldots, n\}$,	initially: $\texttt{max-id}^{[i]} = i$ for all i
`leader`	$\in \{\texttt{true}, \texttt{false}, \texttt{unknwn}\}$,	initially: $\texttt{leader}^{[i]} = \texttt{unknwn}$ for all i
`snd-flag`	$\in \{\texttt{true}, \texttt{false}\}$,	initially: $\texttt{snd-flag}^{[i]} = \texttt{true}$ for all i

`function` msg(w, i)

```
1: if snd-flag = true then
2:    return max-id
3: else
4:    return null
```

`function` stf(w, y)

```
1: case
2:    (y contains only null msgs) OR (largest identifier in y < my-id):
3:        new-id := max-id
4:        new-lead := leader
5:        new-snd-flag := false
6:    (largest identifier in y = my-id):
7:        new-id := max-id
8:        new-lead := true
9:        new-snd-flag := false
10:   (largest identifier in y > my-id):
11:       new-id := largest identifier in y
12:       new-lead := false
13:       new-snd-flag := true
14: return (my-id, new-id, new-lead, new-snd-flag)
```

Figure 1.17 shows an execution of the LCR ALGORITHM. The properties of the LCR ALGORITHM can be characterized as follows.

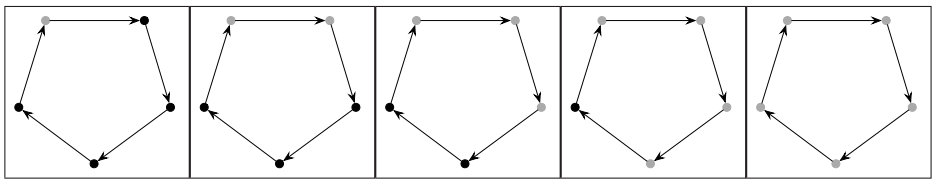


Figure 1.17 Execution of the LCR ALGORITHM. In the leftmost frame, the agent with the maximum UID is colored in gray. After 5 communication rounds, this agent receives its own UID from its in-neighbor and declares itself the leader.

Lemma 1.57 (Complexity upper bounds for the LCR algorithm). *For a ring network $\mathcal{S}$ of order n, the* LCR ALGORITHM *has communication complexity in $\Theta(n^2)$, time complexity equal to n, and space complexity in $\Theta(1)$.*

1.5.5 Shortest-paths tree computation

Finally, we consider the shortest-paths tree problem in a weighted digraph: in Section 1.4.4 we presented the DIJKSTRA ALGORITHM to solve this problem in a centralized setting; we present here the BELLMAN-FORD ALGORITHM for the distributed setting. We consider a synchronous network associated to a weighted digraph, that is, we assume that a strictly positive weight is associated to each communication edge. We aim to compute a tree containing shortest paths from a source, say node 1, to all other nodes. As for the computation of a BFS tree, we aim to obtain a distributed representation of a directed tree with bounded memory at each node:

> *[Informal description]* Each agent maintains in its memory an estimate `dist` of its weighted distance from the source, and an estimate `parent` of the in-neighbor corresponding to the (weighted) shortest path from the source. The `dist` estimate is initialized to 0 for the source and to $+\infty$ for all other nodes. In each communication round, each agent performs the following tasks: (1) it transmits its `dist` value estimate to its out-neighbors, (2) it computes the smallest quantity among "the `dist` value received from an in-neighbor summed with the edge weight corresponding to that in-neighbor," and (3) if the agent's estimate `dist` is larger than this quantity, then the agent updates its `dist` and its estimate `parent`.

The algorithm is formally stated as follows:

`Synchronous Network with Weights:` $\mathcal{S} = (\{1, \ldots, n\}, E_{\text{cmm}}, A)$

`Distributed Algorithm:` DISTRIBUTED BELLMAN-FORD

`Alphabet:` $\mathbb{A} = \mathbb{R}_{>0} \cup \texttt{null} \cup \{+\infty\}$

`Processor State:` $w = (\texttt{parent}, \texttt{dist})$, where

$\texttt{parent} \in \{1, \ldots, n\}$,	initially:	$\texttt{parent}^{[j]} = j$ for all j
$\texttt{dist} \in \mathbb{A}$,	initially:	$\texttt{data}^{[1]} = 0$,
		$\texttt{data}^{[j]} = +\infty$ for all $j = 1$

`function` msg(w, i)

1: **if** `round` $< n$ **then**
2: **return** `dist`
3: **else**
4: **return** `null`

`function` stf(w, y)

1: $i :=$ processor UID
2: $k := \text{arginf}\{y_j + a_{ji} \mid \text{for all } y_j = \texttt{null}\}$
3: **if** $(\texttt{dist} < k)$ **then**
4: **return** $(\texttt{parent}, \texttt{dist})$
5: **else**
6: **return** $(k, y_k + a_{ki})$

In other words, if we let $d_i \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$ denote the `dist` variable for each processor i, then the BELLMAN-FORD ALGORITHM is equivalent to the following discrete-time dynamical system:

$$d_i(\ell + 1) = \inf\left\{d_i(\ell)\,,\, \inf\{d_j(\ell) + a_{ji} \mid (j, i) \in E_{\text{cmm}}\}\right\},$$

with initial conditions $d(0) = (1, +\infty, \ldots, +\infty)$. (Recall that E_{cmm} is the edge set and that the weights a_{ij} are strictly positive for all $(i, j) \in E_{\text{cmm}}$.)

The following formal statements may be made about the evolution of this algorithm. If there exists a directed spanning tree rooted at vertex 1, then all variables d_i will take a final value in time equal to their topological distance from vertex 1. After k communication rounds, the estimated distance at node i equals the shortest path of topological length at most k from the source to node i. Therefore, after $n - 1$ communication rounds, all possible distinct topological paths connecting source to node i have been investigated.

The complexity properties of the DISTRIBUTED BELLMAN-FORD ALGORITHM are described as follows.

Lemma 1.58 (Complexity upper bounds for the distributed Bellman-Ford algorithm). *For a network $\mathcal{S}$ of order n containing a spanning tree rooted at v, the* DISTRIBUTED BELLMAN-FORD ALGORITHM *has communication complexity in $\Theta(n|E_{\text{cmm}}|)$, time complexity equal to $n-1$, and space complexity in $\Theta(1)$.*

Figure 1.18 shows an execution of the DISTRIBUTED BELLMAN-FORD ALGORITHM in a weighted digraph with four nodes and six edges.

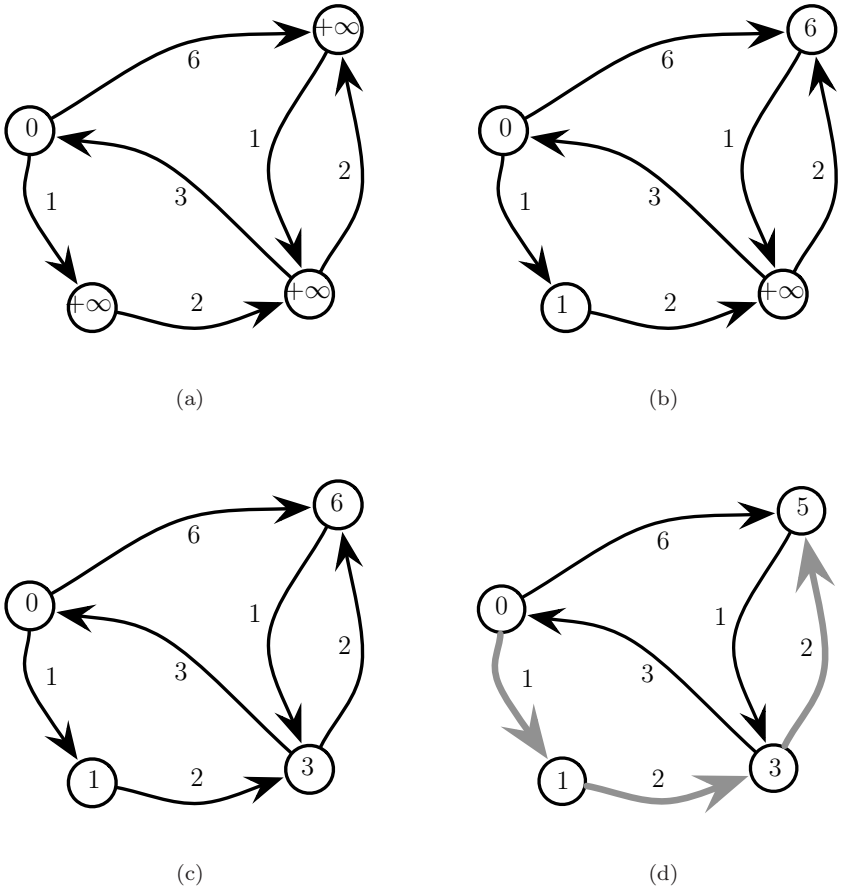


Figure 1.18 Execution of the DISTRIBUTED BELLMAN-FORD ALGORITHM. (a) The processor state initialization. The vertex 1 is the only one whose variable `dist` is 0. After three iterations, as guaranteed by Lemma 1.58, (d) depicts the resulting shortest-paths tree of the digraph rooted at vertex 1. This tree is represented in the last frame, with edges colored in gray.

--continued--