8

Organizing an Asynchronous Network of Processors for Distributed Computation

We have considered so far synchronous and asynchronous algorithms for the solution of a variety of problems on a network of processors. The focus has been on questions of convergence and rate of convergence, and on the computation and communication complexity of these algorithms. These questions can, to a large extent, be viewed separately from the issue of organizing the processor network itself to execute distributed algorithms in either a synchronous or an asynchronous mode. There are a number of questions related to this issue. For example, how does one start a distributed algorithm in an asynchronous processor network? How does one abort a distributed algorithm that is in progress? How does one detect the termination of a distributed algorithm? How can a processor broadcast messages to all other processors? If an algorithm is synchronous, how does one synchronize its computations across the processors of the network? How does one schedule the use of scarce resources among different processors? How can the above be accomplished when the processors and/or the communication links that connect them are subject to failure and repair? We have touched upon some of these topics in previous chapters, and in this chapter, we will provide a more systematic discussion.

The subject of this chapter is very broad and we cannot address it comprehensively within the framework of this book. Our treatment is, therefore, selective and focuses on problems that are most relevant to the numerical computation methods of the earlier chapters. Our treatment is also somewhat nonrigorous because we often do not adhere to formal models of distributed computation. This serves the dual purpose of curtailing the length of the exposition and of emphasizing the more intuitive aspects of the algorithms discussed. We justify, however, our algorithms in sufficient detail to convince the reader of their essential correctness and to provide a starting point for rigorous proofs of their validity.

Throughout this chapter we restrict ourselves to message–passing systems. In Section 8.1, we discuss the problem of detecting the (global) termination of an algorithm based on local termination conditions at the processors of a distributed system. In Section 8.2, we present a snapshot algorithm that can be used to detect certain properties of the global state of a distributed system such as deadlock, for example. In Section 8.3, we consider algorithms for scheduling the resources of an asynchronous processor network when there are restrictions on the simultaneous use of some resources by several processors. In Section 8.4, we discuss a synchronization method, based on rollback, that provides an alternative to the global and local synchronization methods of Section 1.4, and is particularly relevant to simulation of discrete event dynamical systems. In Sections 8.1 through 8.4, we assume that the topology of the processor network does not change with time. In Section 8.5, we consider networks whose topology can change unpredictably due to processor or link failures and repairs. We discuss two fundamental problems within this framework: communication of all processors with a single special processor and communication of a single special processor with all other processors.

8.1 DETECTING TERMINATION OF A DISTRIBUTED ALGORITHM

In many of the algorithms considered in the preceding chapters, there are situations where the computation can naturally be viewed as terminated. For example, the Bellman–Ford algorithm comes to an end when Bellman's equation is satisfied for each node. Similarly one may terminate the iterative solution of a linear system of equations when relaxation of any one of the equations changes the corresponding variable by no more than a given $\epsilon > 0$. These are examples of situations where a global termination condition is decomposed into a collection of local termination conditions, one for each processor. In some cases, for example, when using the global synchronization method of Subsection 1.4.1 based on phase termination messages, there is a processor that can observe simultaneously all the local termination conditions, and can, therefore, detect the global termination condition. In an alternative scheme, a special processor checks (with some delay) whether the system satisfied the global termination condition at the end of some phase, in which case, it issues a command to stop all computation. In a related scheme, a processor issues a special "local termination" message when it reaches the local termination condition, and issues a special "restart" message when it exits that condition. A special processor collects the local termination and restart messages. When the difference between the number of local termination messages and the number of restart messages equals the number of processors, the special processor issues a command to stop the computation, perhaps after some delay to guard against the possibility that additional restart messages are forthcoming. There is no guarantee, however, that this algorithm will always detect termination correctly, particularly when the delays of messages along communication links are unpredictable. The preceding types of schemes are, nonetheless, often adequate in practice, despite the fact that their theoretical properties may not be fully satisfactory.

In this section, we describe, somewhat informally, an alternative method for detecting termination that is theoretically more sound than the methods just described. We have a network of processors connected with bidirectional communication links. We assume that each packet transmitted on a link is correctly received after a finite but unspecified time delay. We do not assume, however, that the links preserve the order of packet transmissions.

It is useful for our purposes to ignore the precise nature of the algorithm whose termination is to be detected and to focus instead on its communication aspects. We use the term "message" to refer to a special type of packet that is related in some way to the algorithm. The examples that follow illustrate the nature of messages in specific contexts. We assume that the algorithm is started when a special processor, referred to as the *initiator*, sends a message to one or more other processors. Subsequently, a number of messages are exchanged by the processors. The implication here is that each processor follows some rules according to which it generates and sends messages to other processors; however, the nature of these rules and the contents of the messages are immaterial for our purposes.

We consider a situation where during execution of the algorithm, each processor is able to monitor its own computations and decide whether a certain "local termination condition" holds. We do not need to specify the precise nature of this condition, but we assume the following:

Assumption 1.1. If the local termination condition holds at some processor, then no messages can be transmitted by that processor. Furthermore, once true, the local termination condition remains true until a message from some other processor is received.

We say that *termination has occurred* at some time t if:

(a) The local termination condition holds at all processors at time t.

(b) No message is in transit along any communication link at time t.

We say that *termination occurs* at time $\bar{t}$, if $\bar{t}$ is the smallest time t for which the above conditions (a) and (b) hold. Our objective is to detect the termination within finite time after it occurs. Notice that if termination has occurred at some time t, then the same is true for every subsequent time $t' > t$, since no messages will be transmitted after time t and the local termination condition will remain true at all processors.

We illustrate the nature of messages and of the termination condition by means of some examples:

Example 1.1. *Asynchronous Fixed Point Iterations*

Consider a network of n processors and the totally asynchronous execution of the iteration $x := f(x)$, where $f : \Re^n \mapsto \Re^n$, as discussed in Section 6.1 (cf. Example 1.1). Each processor i stores the vector

$$x^i(t) = \big(x_1^i(t), \ldots, x_n^i(t)\big), \tag{1.1}$$

it updates its ith coordinate at some times according to

$$x_i^i := f_i(x^i), \tag{1.2}$$

and it subsequently communicates this updated value to the other processors. Here the messages of the algorithm are the updated values of the coordinates. The preceding algorithm, as stated, will execute an infinite number of iterations. To convert it into a finitely terminating algorithm, we assume that the update (1.2) is not executed (and the attendant communication of the updated coordinate does not take place) if

$$|x_i^i - f_i(x^i)| \le \epsilon, \tag{1.3}$$

where ϵ is a given positive scalar. This is the local termination condition at i. Thus, termination occurs when Eq. (1.3) holds simultaneously for all processors i and there is no message in transit along any communication link. Note the nature of the difficulty here; each processor i can verify its own local termination condition (1.3), but cannot easily detect whether this condition holds simultaneously for all processors and whether there is any message in transit.

Example 1.2. *Asynchronous Bellman–Ford Algorithm*

In the context of the preceding example, consider the asynchronous Bellman–Ford algorithm discussed in Section 6.4 for finding the shortest distances of all nodes to node 1. The algorithm updates the shortest distance estimate of a node $i \neq 1$ according to

$$x_i := \min_{j \in A(i)} (a_{ij} + x_j), \tag{1.4}$$

where $A(i)$ is the set of all nodes such that (i, j) is an arc, and the updated value is communicated to the processors j such that $i \in A(j)$. Thus, the messages of this algorithm are the updated values of the shortest distance estimates. We assume that when the iteration (1.4) does not change the value of x_i, this value is not communicated to any other processor. The local termination condition here holds at i if execution of iteration (1.4) does not change the value of x_i. Based on the results of Section 6.4, the number of messages generated by the algorithm is finite, and termination occurs within finite time after all processors have found their shortest distance to the destination.

Our termination detection procedure is based on message acknowledgments. In particular, messages received by processor i from processor j, are acknowledged by i by sending to j special acknowledgment packets (abbreviated ACKs). Note that ACKs are distinct from messages and are not acknowledged by further ACKs. We assume that each message and each ACK arrives at its destination after some positive (but finite) time from the time it was transmitted. Furthermore, each processor can transmit only a finite number of messages and ACKs within any bounded time interval, can transmit at most one message or ACK to another prossesor at any one instant of time, can receive at most one message or ACK at any one instant of time, and is not allowed to transmit (a message or ACK) and simultaneously receive (a message or ACK) at any instant of time. These assumptions do not diminish the practical relevance of our analysis. For example, they would be appropriate for a practical situation where there can be multiple simultaneous transmissions and receptions at a processor, but the processor has a way of establishing unambiguously the relative temporal order of these transmissions and receptions. Our assumptions are used to resolve in a simple manner ambiguities about the algorithmic rules that will be described shortly. They also allow us to simplify the presentation by assuming without loss of generality that communication events, that is, transmissions and receptions of messages and ACKs, occur at integer times $t \geq 0$ and that the initial transmission of the initiator occurs at time $t = 0$. Accordingly, in all subsequent references to communication events, we imply that *these events occur instantaneously at integer times*. The expression "just after time t", where t is an integer, will refer to all times that are larger than t and smaller than $t + 1$.

There are two possible states for a processor at any one time: the *inactive state* (in which the processor is called *inactive*) and the *active state* (in which the processor is called *active*). We describe these states, the restrictions that a processor must obey at each state, and the circumstances under which a processor changes states. All changes of state occur at integer times, and we will adopt the convention that when a processor changes its state from A to B at time t, then the state is A at time t and it is B just after time t.

In the inactive state, a processor i can send no messages or ACKs. It will move from the inactive state to the active state at a time t if it receives a message at time t from some other processor j. This message and the corresponding processor j play a special role for the period between t and the next time t' at which processor i returns to the inactive state. During this period, the message is called the *critical message*, j is called the *parent* of i, whereas i is called a *child* of j.

In the active state a processor may transmit any number of messages to any one of its neighbors (subject to the limitation of one per neighbor and time instant assumed earlier). It must also acknowledge within a finite number of time units each message that it receives except for the critical message. An active processor becomes inactive simultaneously with transmitting an ACK for its critical message. This ACK is transmitted at the first time t for which the following conditions hold:

(a) No message is received by the processor at time t.

(b) The local termination condition holds at the processor at time t.

(c) The processor has transmitted prior to t an ACK for each message it has received except for the critical message.

(d) The processor has received prior to t an ACK for each message it has transmitted.

The preceding rule for changing to the inactive state is also imposed on an active processor without a parent, except that the references to the critical message are unnecessary. It will be seen shortly that under our assumptions, the initiator is the only processor that is ever active without having a parent. Note that the parent i of a processor j must be active, since i must still be awaiting the ACK for the critical message it sent to j. A diagram summarizing the algorithm is given in Fig. 8.1.1.

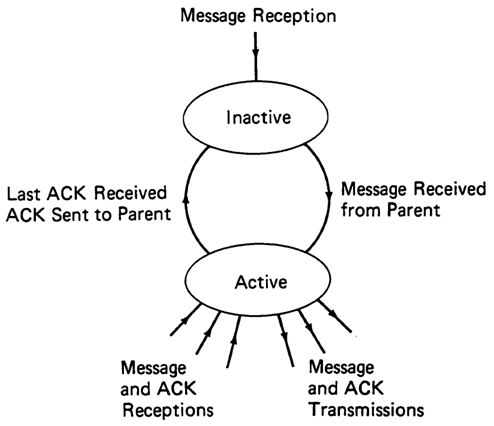


Figure 8.1.1 States of a processor during the termination detection algorithm. In the inactive state a processor transmits no messages or acknowledgments (ACKs), and changes to the active state upon receiving a message from some other processor, which then becomes the processor's parent. In the active state, a processor trasmits messages and sends an ACK for each message it receives. The processor changes its state to inactive upon sending an ACK to its parent, and is allowed to send this ACK only after it receives an ACK for each of the messages it has transmitted, and in addition, its local termination condition holds.

Initially, at time $t = 0$, the initiator is active and all other processors are inactive. Furthermore, at $t = 0$, the local termination condition holds at all processors except for the initiator. It can be seen that based on the rules of the algorithm, if a processor is inactive at time t, the following are true:

(a) Its local termination condition holds at t.

(b) It has transmitted ACKs for all the messages it has received prior to t.

(c) It has received ACKs for all the messages it has transmitted prior to t.

If the processor is active at time t, at least one of the above conditions is violated.

We say that *termination is detected*, at the first time when the initiator becomes inactive. It follows that up to the time that termination is detected, the initiator will never receive a critical message and acquire a parent, whereas every other processor will always have a parent while it is active.

We will show that the procedure we have described has the following two properties:

(a) If termination is detected at time t', then termination occurred at some time $t \leq t'$.

(b) If termination occurs at time t, then termination is detected at some time $t' \geq t$.

The key concept for showing these properties is the *activity graph* (abbreviated AG), which at each time t, consists of all the processors that are active together with the directed arcs that connect the parents of these processors with the processors themselves. We make the following observations regarding the AG:

(1) At all times prior to termination detection, the initiator is active and therefore belongs to the AG.

(2) At all times, the parent of any processor that belongs to the AG (other than the initiator) is unique and must also belong to the AG.

(3) If a processor j belongs to the AG just after time t, then its parent i belongs to the AG at all times in the interval $(t-2, t+2)$. (The reason is that the critical message sent from i to j must have been transmitted at a time $t' \leq t-1$ and the corresponding ACK will not be transmitted by j prior to time $t+1$ and will not be received by i prior to time $t+2$; we are using here the assumption that all messages and ACKs take at least one time unit to reach their recipients. Based on this observation, we see that only childless processors are added to or removed from the AG at any one time.)

An important fact is that at all times prior to termination detection, *the AG is a tree of directed arcs that is rooted at the initiator* (meaning that it has no cycles, it contains the initiator and also contains a unique positive path from the initiator to every one of its other processors; see Fig. 8.1.2). To show this, it will be sufficient, in view of the above observations (1) and (2), to show that at all times prior to termination detection the AG does not contain a positive cycle. Indeed, initially the AG consists of just the initiator and if the AG first acquired a positive cycle C just after time t, then each processor of C is the parent of another processor of C which is active just after time t. By the above observation (3), it follows that each processor of C is active just after time $t-1$. Since the AG is acyclic just after time $t-1$, it follows that some arc of the cycle C was added to the AG at time t, joining two already active processors. This, however, is impossible since an arc (i,j) is added to the AG only when the processor j changes from inactive to active.

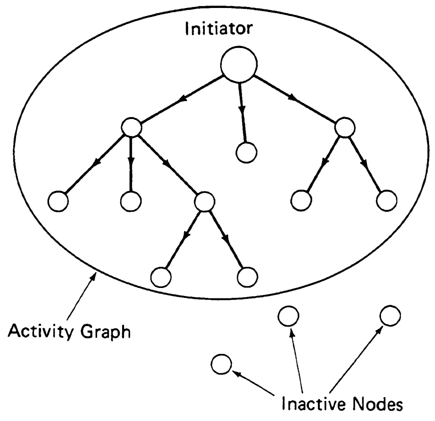


Figure 8.1.2 The activity graph at a given time consists of all the processors that are active at that time together with directed arcs connecting the parents of the active processors with the processors themselves. At all times, the graph is a tree of directed arcs that is rooted at the initiator in the sense that it has no cycles, it contains the initiator and also contains a unique positive path from the initiator to every one of its other nodes.

Assume now that termination is detected at time t'. We claim that the activity graph becomes empty at t' and that termination must have occurred at time t'. To show this, we first observe that just after time $t' - 1$, the initiator must be the only processor in the AG since otherwise, the initiator would be the parent of some processor just after time $t' - 1$ and by the above observation (3), the initiator would not become inactive at time t'. We next observe that a processor cannot become active at time t' since by the above observation (3), its parent must be active in the interval $(t' - 2, t' + 2)$; the parent cannot be different than the initiator because it must be inactive just after time $t' - 1$ as shown earlier, and it cannot be the initiator which turns inactive at time t' by assumption. Therefore, at time t', the activity graph becomes empty and all processors are inactive. From the rules by which processors become inactive, it follows that the local termination condition holds at all processors and that there are no messages in transit at time t'. This means that termination has occurred at time t'.

Suppose finally that termination occurs at time t. We will show that termination is detected (necessarily at some time $t' \geq t$, based on what has been proved so far). Indeed after t, no processor can change its state from inactive to active because no further messages will be received by any processor, so if termination is never detected, the activity graph must not change after some time while being nonempty. This, however, cannot happen because each processor i that is active and has no child in the final activity graph will eventually receive an ACK for each message it sent to the other processors (once termination occurs and the activity graph stops changing, processor i will never again have a child, so each processor must either have sent an ACK to i or will send eventually an ACK to i for every message it has received from i). Since the local termination condition holds at processor i after termination has occurred, i must eventually become inactive, thereby contradicting the hypothesis that it stays active indefinitely. We see, therefore, that the activity graph will eventually become empty at which time termination is detected.

We now consider the communication overhead that can be attributed to the termination detection procedure. Since there is one ACK per message, we see that when the procedure is used, the total number of packet transmissions is doubled. On the other hand, the ACKs can be incorporated in some way into an existing data link control scheme such as those discussed in Subsection 1.3.2, in which case, the communication overhead for termination detection is almost negligible.

Consider next the delay associated with the termination detection procedure, that is, the difference between the time when termination is detected and the time when termination occurs. Let us consider the activity graph at the time when termination occurs. It can be seen that the delay for termination detection is the time needed for the ACKs to propagate to the initiator along the links of the AG, starting from the childless nodes and proceeding toward the initiator. Thus the delay for termination detection is $O(r)$, where r is the maximum number of links on a path of the AG that connects the initiator with a node of the AG. Since r is less than the number p of processors of the distributed system, the delay for termination detection can be estimated as $O(p)$.

There are a number of variations of the termination detection procedure just described. For example, it is possible that a single ACK can acknowledge several messages

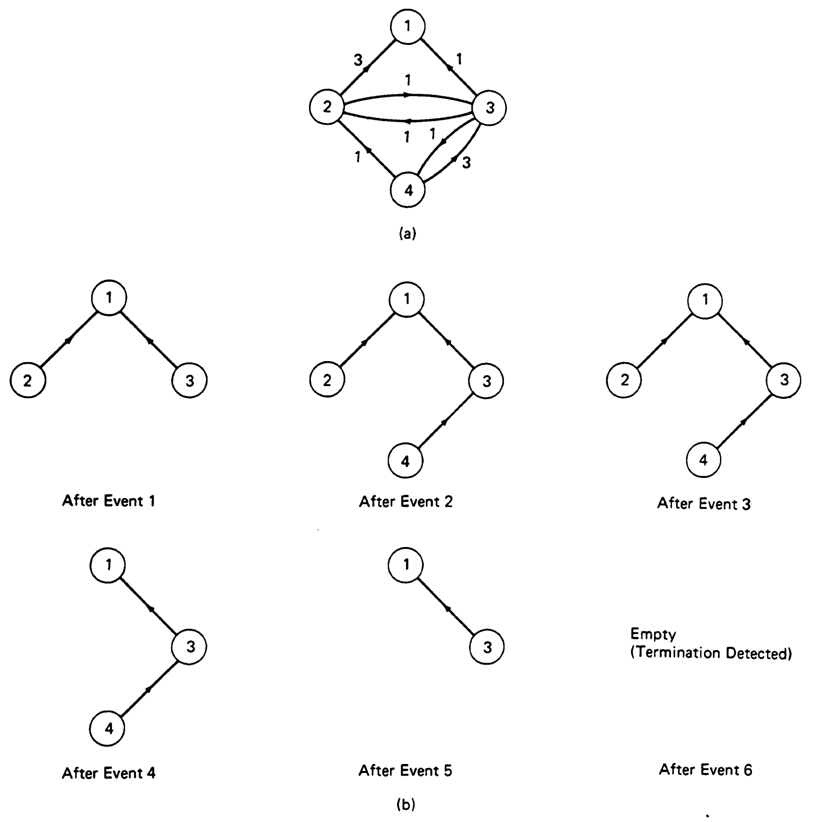


Figure 8.1.3 Illustration of the termination detection procedure for the asynchronous Bellman–Ford algorithm. (a) Shortest path problem data. The number next to each arc is the length of the arc. (b) The activity graphs corresponding to the following sequence of events involving updates according to the Bellman–Ford iteration, and message and ACK transmissions. Each event may require several time units.

Event 1: Nodes 2 and 3 receive message "$x_1 = 0$" from node 1. Node 2 becomes active due to the message from node 1, declares node 1 as its parent, and sends message "$x_2 = 3$" to nodes 3 and 4. Node 3 becomes active due to the message from node 1, declares node 1 as its parent, and sends message "$x_3 = 1$" to nodes 2 and 4. All these messages are received before event 2 begins. The 3–to–4 message is received before the 2–to–4 message (so node 4 will declare node 3 as its parent).

Event 2: Node 3 sends ACK to node 2. Node 2 sends message "$x_2 = 2$" to nodes 3 and 4, and then sends ACK to node 3. Node 3 sends (a second) ACK to node 2. Node 4 becomes active, declares node 3 as its parent, sends message "$x_4 = 4$" to node 3, and sends ACK to node 2. All messages and ACKs are received before event 3 begins.

Event 3: Node 3 sends ACK to node 4. Node 4 sends message "$x_4 = 3$" to node 3, and sends ACK to node 2. All messages and ACKs are received before event 4 begins.

Event 4: Node 2 sends ACK to node 1 and becomes inactive. Node 3 sends ACK to node 4. These ACKs are received before event 5 begins.

Event 5: Node 4 sends ACK to node 3 and becomes inactive. The ACK is received before event 6 begins.

Event 6: Node 3 sends ACK to node 1 and becomes inactive. Node 1 receives the ACK of node 3, becomes inactive, and termination is detected.

simultaneously. Furthermore, it is not crucial that each ACK is tied to a specific message. What is important for the execution of the algorithm is that each node keeps track of the difference between the number of messages transmitted and ACKs received along each link. When this difference becomes zero, the node knows that no more ACKs are pending from the opposite end processor of that link.

Figure 8.1.3 illustrates the termination detection procedure in the context of the asynchronous Bellman–Ford algorithm (Example 1.2) for a particular sequence of message and ACK receptions. The following is another example of a termination detection algorithm.

Example 1.3. *Verifying the Reception of Broadcast Information*

Consider a network of processors connected with bidirectional communication links. Suppose that some processor, referred to as the *initiator*, wishes to send some information, call it I, to all other processors and to verify the reception of I by all other processors. The following algorithm uses two types of packets, denoted M and A, which are assumed to be received within finite time from the start of their transmission. The packets M carry the value of I and the packets A play the role of acknowledgments of reception of packets M. The algorithm operates as follows:

The initiator starts the algorithm by sending M to all its neighbor processors.

When processor i receives M for the first time, say from processor $f(i)$, it stores the identity number of $f(i)$ and sends M to all its neighbors except for $f(i)$, if it has at least one such neighbor; otherwise it sends A to $f(i)$.

When processor i receives M for the second and subsequent times, say from processor j, it sends A to j.

When processor i receives A from all neighboring processors other than $f(i)$, it sends A to $f(i)$.

By associating M with messages and A with ACKs, and by viewing $f(i)$ as the parent of processor i, we see that the preceding algorithm is a special case of the termination detection procedure. It follows that the initiator will eventually receive A from all its neighbors and that at that time, all processors will have received M. Figure 8.1.4 illustrates the operation of the algorithm.

Algorithms such as the one of the preceding example are often useful in data networks. For instance, the reader may wish to construct a similar algorithm that allows the initiator to determine the number and/or identities of all the processors in the network.

8.2 SNAPSHOTS

Each processor in a distributed computing system typically has access only to local information, that is, its own state of computation as well as the messages it sends and receives. Now suppose that we wish to detect whether the global state of the distributed system has certain properties. For instance, we might wish to detect whether the system

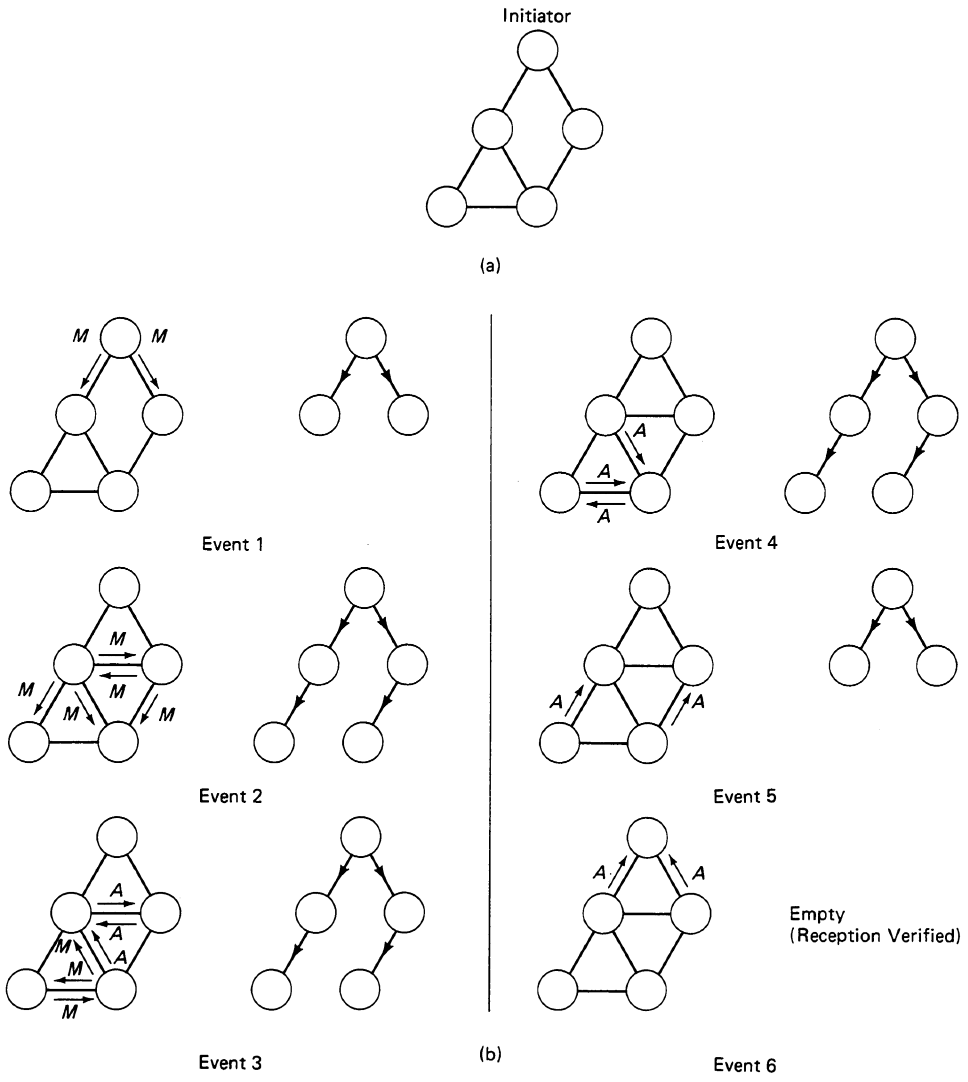


Figure 8.1.4 Illustration of the algorithm for verification of the reception of broadcast information of Example 1.3. (a) Graph over which the information is broadcast by the initiator. (b) A sequence of message and ACK transmissions (shown on the left) together with the corresponding activity graphs following these transmissions (shown on the right). M indicates a message transmission and A indicates an ACK transmission.

is deadlocked, whether the vector generated by an asynchronous iterative algorithm has come sufficiently close to a solution, or to monitor the progress of some distributed algorithm. Ideally, such a task might be accomplished by having the processors simultaneously record and then transmit their local states to a central processor. However, simultaneous recording of local states is generally impossible if the processors do not have access to a global clock and if messages (that could be used for synchronization) are subject to unknown delays. For this reason, we will settle for the lesser goal of having the processors record their local states at a set of times that are *possibly simultaneous*, meaning that the information available to the processors does not contradict the simultaneity of these times. We provide an algorithm (known as the snapshot algorithm) that achieves this goal and we then show how the recorded information can be used to detect certain properties of the true global state of the system.

We start by stating the assumptions on the operation of the distributed system. Let $G = (N, A)$ be a directed graph. The set N of nodes corresponds to the processors and each arc (i, j) represents a unidirectional, never failing, and error–free communication link. We assume that G is strongly connected, meaning that there exists a positive path from every processor to every other processor. Concerning the communication links, we assume that for each $(i, j) \in A$, a message sent by processor i to processor j reaches its destination after a positive but finite amount of time, and, furthermore, messages are received by processor j in the order in which they were transmitted. The latter assumption (often called FIFO, for first in, first out) is crucial for the correct operation of the snapshot algorithm to be presented later. It can be enforced by using an appropriate protocol for data link control (Subsection 1.3.2).

We associate a local clock with each processor. The current value t_i of the local clock of processor i will be called i's *local time*, as opposed to *global time*, denoted by τ, which is the time in the clock of an external observer. We make the assumption that distinct global times correspond to distinct local times, and conversely. In particular, the global time can be written as $h_i(t_i)$, where t_i is the local time of processor i and h_i is a strictly increasing function. The material in this section could have been presented without introducing the notion of global time. This is because global time (equivalently, the function h_i) is not known by any processor and its value cannot have any effect on the algorithms being implemented. Nevertheless, the concept of global time simplifies the presentation.

We assume that the processors are engaged in some collective computation and that at any time, the state of computation of processor i is specified by a *local state* x_i belonging to some state space X_i. [For example, the local state variable x_i could consist of the values of all variables in the memory of the ith processor that are related to the collective computation. In the special case of an asynchronous iteration of the form $y := f(y)$, the local state variable x_i is naturally identified with the value of the vector y kept in the memory of processor i.] The computation involves the exchange of messages. For any $(i, j) \in A$, let M_{ij} be the set of all possible messages from i to j. The *state of a communication link* $(i, j) \in A$, at some global time, is defined to consist of all messages that have been transmitted through that link and have not yet been received. A collection of local states (a variable x_i for each processor i) and of link states [a sequence of messages m_{ij} for every $(i, j) \in A$] is called a *global state*. If the processor and link states in a global state s are the actual states at some global time τ, we say that s is *the global state at (global) time* τ. We use $\mathcal{S}$ to denote the set of all global states.

The computations performed by each processor are modeled as a sequence of *events*. When an event occurs at processor i, the state x_i of that processor may change and certain messages m_{ij} to other processors may be generated. Events can be of two types: they can be *self–induced*, in which case, the state x_i changes spontaneously to a new value, or they can be *message–induced*; in the latter case, an event occurs upon reception of a message $m_{ji} \in M_{ji}$ from some processor j with $(j, i) \in A$. In the sequel, the following assumption will be in effect. It is somewhat stronger than necessary, but helps in simplifying the discussion.

Assumption 2.1.

(a) Events are instantaneous: each event at any processor i takes a zero amount of time. (Accordingly, it is meaningful to talk about "the time that an event occurs" or *event time*, for brevity.)

(b) Distinct events at the same processor occur at different local times.

(c) Only a finite number of events can occur during a time interval of finite length.

(d) Different messages received by the same processor are received at distinct local times. In particular, a message–induced event is caused by exactly one message.

Let us consider local times t_i, t_j, and t_k at three respective processors i, j, and k. Suppose that a message m_{ij} is transmitted by processor i at a local time larger than t_i, and that this message is received by j at a local time smaller than t_j. In such a case, there is sufficient information to determine that $h_i(t_i) < h_j(t_j)$. If, in addition, some other message m_{jk} from j to k testifies to the fact that $h_j(t_j) < h_k(t_k)$, then it can be inferred that $h_i(t_i) < h_k(t_k)$. If, on the other hand, two local times t_i and t_j are such that no inference of the form $h_i(t_i) < h_j(t_j)$ can be made, we will say that the local times are possibly simultaneous. We shall make this notion precise and then present an algorithm that records the state of each processor, and of each link, at possibly simultaneous times.

We define a collection $\{t_i \mid i \in N\}$ of local times to be *possibly simultaneous* if every message m_{ij} sent by i at or after t_i is received by j after t_j, and this property holds for every $(i, j) \in A$. We define the *snapshot* corresponding to a collection $\{t_i \mid i \in N\}$ of possibly simultaneous times to be the global state consisting of the local states x_i of each processor i at local time t_i and, for each $(i, j) \in A$, of the sequence of all messages sent by i before t_i and received by j after t_j (Fig. 8.2.1). This definition could be ambiguous if some t_i is an event time at processor i (do we record the state x_i immediately before or immediately after the event?) and for this reason, we shall subsequently ensure that none of the times t_i is an event time.

We now present an algorithm by means of which a snapshot of the system is taken. The algorithm uses special *marker* messages. We assume that these marker messages do not interfere with the underlying collective computation carried out by the processors. In particular, any sequence of events and message transmissions related to the underlying computation that was possible in the absence of the marker messages is also assumed to be possible when the marker messages are present, and conversely.

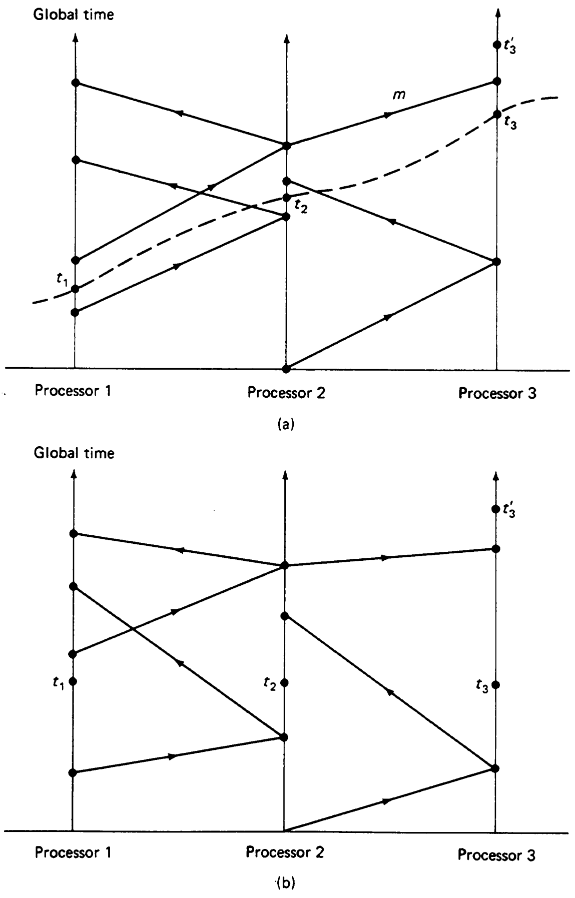


Figure 8.2.1 Illustration of a set of possibly simultaneous times and of a corresponding snapshot. Each axis represents the progress of each processor as global time increases. Each node in the diagram represents an event and each arc represents a message. Although the vertical axes stand for global time, the labels t_1, t_2, t_3, and t_3' are the local times of the respective processors at the indicated points. Times t_2 and t_3' are not possibly simultaneous because of the presence of the message m. However, the times t_1, t_2, and t_3 are possibly simultaneous. In part (b), we show an alternative timing of the different events that is indistinguishable from the one in part (a) as far as the sequence of events in each processor is concerned and in which the local times t_1, t_2, and t_3 correspond to the same global time.

It can be shown that a timing diagram as in (a) can be always redrawn to an equivalent diagram (meaning that for each processor, the same events occur in the same order) so that a given set of possibly simultaneous times is transformed into truly simultaneous times.

The dashed line in (a) illustrates a snapshot. The state of each processor i is recorded at local time t_i and the link states consist of all message that cross the dashed line. In this example, the recorded states of links $(1, 2)$ and $(2, 3)$ consist of the empty sequence and the recorded states of links $(2, 1)$ and $(3, 2)$ consist of one message.

Notice that messages crossing the dashed line can only start below the dashed line and end above it; otherwise the possible simultaneity of the times t_1, t_2, and t_3 would be contradicted.

The Snapshot Algorithm [ChL85]

(A1) (*Initialization*) One or more processors initiate the algorithm by recording their respective local states and immediately sending a marker message to each of their neighbors. For every initiating processor i, let t_i^* be its local time upon initiation. It is assumed that t_i^* is not an event time at processor i.

(A2) (*Recording of processor states*) Each processor i that is not an initiator records its local state at the first local time t_i^* that it receives a marker message. (If the first reception of a marker message by processor i occurs at an event time, we assume that the marker reception is artificially delayed by a negligible amount, thus ensuring that t_i^* is not an event time.)

(A3) (*Marker generation*) If processor i is not an initiator, it sends a marker message along each of its outgoing arcs at local time t_i^*.

(A4) (*Recording of link states*) For any $(i, j) \in A$, let t_{ij}^* be the first local time that processor j receives a marker message from processor i. Processor j records the state of link (i, j) by recording the sequence of all messages received by j on that link between local times t_j^* and t_{ij}^*.

Figure 8.2.2 illustrates the algorithm and shows that the global state recorded by a snapshot can be different from any one of the true global states at the different global times.

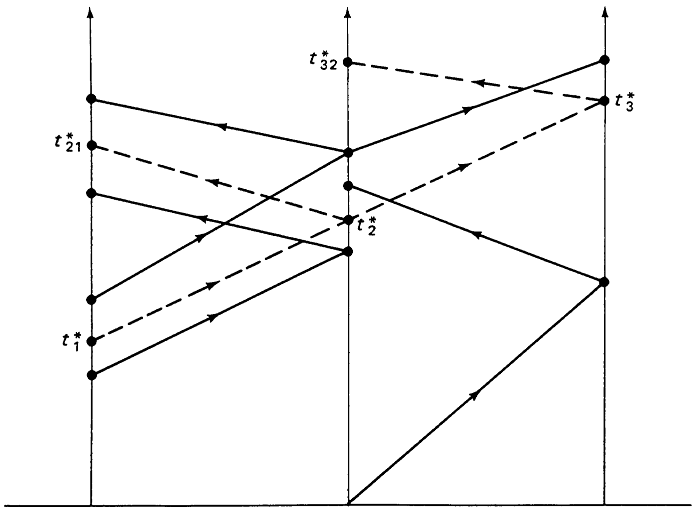


Figure 8.2.2 Illustration of the snapshot algorithm for the sequence of events shown in Fig. 8.2.1. The algorithm is initiated at local time t_1^* by processor 1. The dashed lines indicate the travel of marker messages. Processor 1 records the state of link (2, 1) to consist of all messages received between local times t_1^* and t_{21}^*. This is the set of all messages that were generated by processor 2 in the "past" (that is, before time t_2^*) and received by processor 1 in the "future" (that is, after time t_1^*).

We now show that the snapshot algorithm performs as desired. We first show that the algorithm eventually terminates with every processor having recorded its local state as well as the states of its incoming links. Because of the marker generation rule (A3), and because the graph G has been assumed strongly connected, it follows that every processor that is not an initiator eventually receives a marker message. In particular, t_i^* is well defined for each i, and every processor eventually records its local state. We then see [cf. (A1) and (A3)] that exactly one marker message is transmitted along each arc $(i,j) \in A$. In particular, t_{ij}^* is well defined and finite for every $(i,j) \in A$. Thus, each link's state is recorded. We conclude that the snapshot algorithm eventually terminates.

Next we show that the local times t_i^* at which the processors record their states are possibly simultaneous. Suppose that $(i,j) \in A$ and that a message m_{ij} is transmitted from processor i to processor j after time t_i^*. We see that this message is transmitted after the transmission of a marker message from i to j. Using the FIFO assumption, the message m_{ij} is received by processor j subsequent to the reception time t_{ij}^* of the marker message. Since $t_{ij}^* \geq t_j^*$, it follows that m_{ij} is received after time t_j^*. This proves that the times $\{t_i^* \mid i \in N\}$ are possibly simultaneous.

In order to complete the proof that the states recorded by the algorithm form a snapshot, it remains to show that the recorded state of a link (i,j) is the sequence of messages sent along that link before t_i^* and received after t_j^*. According to (A4), the recorded messages are those received by processor j between local times t_j^* and t_{ij}^*. From the FIFO assumption, this is the same as the set of messages sent by i before time t_i^* and received by j after t_j^*, which completes the proof.

As just shown, the snapshot algorithm is guaranteed to terminate. Termination can be detected by the processors as follows. As soon as a processor records its own state and the states of its incoming links, it broadcasts to all processors a termination message. As soon as a processor receives a termination message from all other processors, it knows that the snapshot algorithm has terminated. Furthermore, the processors can send the recorded states to a particular processor, designated as a center, which could analyze the data provided by the snapshot and draw inferences on the global state of the distributed system. It was indicated earlier that the states recorded in a snapshot need not correspond to a true global state of the distributed system. Despite that, it will be shown that the information contained in a snapshot is sufficient for detecting certain interesting properties of the true global state.

Let $\mathcal{S}_0$ be a subset of the set $\mathcal{S}$ of all global states. We say that $\mathcal{S}_0$ is *invariant* if it has the following property: if the global state at some global time τ belongs to $\mathcal{S}_0$, then the same is true for every possible global state at any subsequent global time $\tau' > \tau$.

It has been shown [ChL85] that a snapshot can be employed to detect membership in an invariant set, under certain assumptions. We are interested, in particular, in the following two properties:

Property P1. If the global state of the system at global time τ belongs to $\mathcal{S}_0$ and if the snapshot algorithm is initiated after global time τ, then the global state recorded by the snapshot algorithm also belongs to $\mathcal{S}_0$.

Property P2. If the global state recorded by the snapshot algorithm belongs to $\mathcal{S}_0$, then the global state of the system also belongs to $\mathcal{S}_0$ at any global time τ subsequent to the termination of the snapshot algorithm.

We will prove the above two properties for the special case where the invariant set $\mathcal{S}_0$ has a simple Cartesian product structure. A more general result can be found in [ChL85].

Let there be given a subset X_i^* of X_i and a subset M_{ij}^* of M_{ij} for each $i \in N$ and each $(i,j) \in A$, respectively.

Assumption 2.2.

(a) If the local state x_i of processor i belongs to X_i^* prior to a self–induced event, then it belongs to X_i^* after the event as well. Furthermore, any messages m_{ij} generated by that event belong to the set M_{ij}^*.

(b) Part (a) of this assumption holds for message–induced events as well, provided that the message m_{ji} causing the event belongs to M_{ji}^*.

Let $\mathcal{S}_0$ be the set of all global states for which the local state x_i of each processor i belongs to X_i^* and every message in the state of a link $(i,j) \in A$ belongs to M_{ij}^*. We see that under Assumption 2.2, the set $\mathcal{S}_0$ is invariant.

We now verify Properties P1 and P2 when $\mathcal{S}_0$ is defined as above and Assumption 2.2 holds. Property P1 is obvious because if the global state at some global time τ belongs to $\mathcal{S}_0$, then every subsequent local state of processor i belongs to X_i^* and every subsequently generated message belongs to a set M_{ij}^*, which establishes that the global state recorded by a subsequent snapshot belongs to $\mathcal{S}_0$.

We now assume that a snapshot corresponding to a set of possibly simultaneous times $\{t_i^* \mid i \in N\}$ has recorded a state in $\mathcal{S}_0$, and we verify Property P2. We will say that an event at processor i has Property P if the state of processor i after the event belongs to X_i^* and any message m_{ij} generated by that event belongs to M_{ij}^*. We assume that Property P2 does not hold, in order to derive a contradiction. Then there exists an event that occurs at some processor i at some local time $t_i > t_i^*$ that does not have Property P. Let us choose a processor i and such an event for which the corresponding global time $h_i(t_i)$ is as small as possible. If this event is self–induced, then Assumption 2.2(a) implies that the state of processor i did not belong to X_i^* before the event. However, x_i belonged to X_i^* at time t_i^* and we conclude that some event must have occurred between local times t_i^* and t_i that does not have Property P. This contradicts our definition of t_i. We now consider the case where the event at time t_i is message–induced. Using Assumption 2.2(b) and the same reasoning as before, we conclude that the message m_{ji} inducing this event does not belong to M_{ji}^*. We distinguish two subcases. If the message m_{ji} was transmitted by processor j before local time t_j^*, then this message is one of the messages recorded in the state of link (j,i) and, therefore, belongs to M_{ji}^*, a contradiction. In the second subcase, the message m_{ji} was transmitted after time t_j^*. It follows that this message was generated by some event at processor j at some (local) time t_j satisfying $t_j > t_j^*$ and, furthermore, since $m_{ji} \notin M_{ji}^*$, the event at processor j did not have property P. On the other hand, we have $h_j(t_j) < h_i(t_i)$, which again contradicts our choice of i and t_i, and completes the proof.

Example 2.1. *Termination Detection*

For each processor i, let X_i^* be the set of states x_i from which any self–induced event leaves x_i unchanged and generates no messages. (The condition $x_i \in X_i^*$ is to be interpreted as a local termination condition for processor i.) For every link $(i,j) \in A$, let M_{ij}^* be the empty set. It is clear that the sets X_i^* and M_{ij}^* satisfy Assumption 2.2, and the corresponding set $\mathcal{S}_0$ is invariant. It is seen that the global state of the system belongs to $\mathcal{S}_0$ if and only if the underlying computation has terminated. We conclude that a snapshot can be usefully employed for termination detection.

Another interesting application of snapshots will be seen in Section 8.4 on asynchronous simulation.

EXERCISES

2.1. **[Gaf86]** Suppose that each one of the sets $\{t_i \mid i \in N\}$ and $\{t_i' \mid i \in N\}$ is a collection of possibly simultaneous local times. Show that the same is true for each one of the sets $\{\max\{t_i, t_i'\} \mid i \in N\}$ and $\{\min\{t_i, t_i'\} \mid i \in N\}$.

8.3 RESOURCE SCHEDULING

We consider an asynchronous network of processors, described by an undirected graph $G = (N, A)$, where $N = \{1, \ldots, n\}$ is the set of processors, and A is the set of undirected arcs. Each arc represents a perfectly reliable bidirectional communication link joining processors i and j. We assume that the processors participate in some computational task and that they perform certain operations once in a while. We assume, however, that each arc (i,j) is associated with some shared resource R_{ij}, which is necessary for either processor i or j to perform an operation, and that this resource can only be possessed by one processor at a time. Therefore, for the computation to proceed, we need a processor i to gain control of all resources associated with its incident arcs. As shown in Fig. 8.3.1, it is possible that the processors become deadlocked if each one is idle waiting for some needed resource. We will present a simple algorithm under which deadlock is avoided and all processors have a chance to operate an infinite number of times.

This problem, also known as the *dining philosophers problem*, is of fundamental importance in synchronization and deadlock avoidance in distributed systems, and applies to a variety of situations. The example closest to the applications discussed in this book relates to the distributed execution of an iteration of the form

$$x_i := f_i(x_1, \ldots, x_n), \qquad i = 1, \ldots, n. \tag{3.1}$$

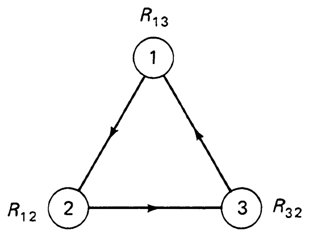
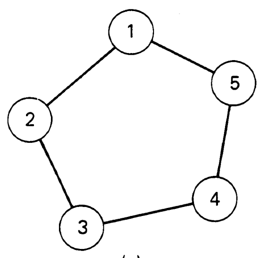


Figure 8.3.1 Illustration of deadlock. Each processor has taken hold of one of the two resources it needs and waits to take hold of the second. A direction is assigned to each arc (i, j) to indicate the processor holding the corresponding resource R_{ij}. As the situation of each processor is completely symmetrical, they are faced with a deadlock.

Here we let G be the corresponding undirected dependency graph, that is, an arc (i, j) is present if and only if f_i depends on x_j or f_j depends on x_i. Consider a serial execution of iteration (3.1), whereby at each time t, some processor $i(t)$ executes Eq. (3.1) and informs the other processors about the new value of $x_{i(t)}$. Suppose that such a serial algorithm is convergent for all possible choices of $i(1)$, $i(2), \ldots$, provided that each processor executes an infinite number of times. This is the case, for example, for the dual relaxation algorithms for network flow problems, which were studied in Chapter 5. The same is true in several situations in which there is an underlying cost function that decreases whenever a single processor performs an update. Coordinate descent methods, such as the Gauss–Seidel algorithm for linear equations or the nonlinear Gauss–Seidel algorithm for nonlinear problems, are some examples. In some such algorithms, convergence can be lost if two neighboring processors i and j [with $(i, j) \in A$] are allowed to perform an update simultaneously. For example, the nonlinear Gauss–Seidel algorithm for strictly convex optimization problems is guaranteed to converge, but this is not always the case for the nonlinear Jacobi algorithm in which all processors update simultaneously. We conclude that there are several circumstances in which we wish iteration (3.1) to be executed in a way that is mathematically equivalent to a serial execution (one processor at a time). To guarantee this, it is sufficient to require that no two adjacent processors operate simultaneously. This, in turn, will be accomplished by means of a resource R_{ij} that cannot be simultaneously held by processors i and j.

A synchronous solution to our problem is obtained by using a coloring scheme, as discussed in Subsection 1.2.4. Here we associate a particular "color" with each processor in the graph, subject to the constraint that no two adjacent processors have the same color. Let the different colors be numbered from 1 to K. The computation proceeds in stages. At the first stage, all processors with the first color operate, then processors with the second color, all the way to the last color, and then we restart with the first color. Such a scheme is *fair* in the sense that the number of operations performed by different processors can differ by at most one. The example in Fig. 8.3.2 shows that the coloring approach is not necessarily the most efficient possible synchronous scheduling method. In an inherently asynchronous system, the previously outlined synchronous solution can be implemented using a synchronization mechanism (Section 1.4). In particular, a global synchronization method could be used, but the overhead involved is unwarranted. We do not discuss the use of a local synchronization method for this problem because it leads to an algorithm similar to the one to be presented.

We motivate our algorithm in terms of the resource allocation problem mentioned in the beginning of this section. With each arc $(i, j) \in A$, we associate a resource R_{ij}.



(a)

Processors Operating	1 3	2 4	5	1 3	2 4	5	. . .
Time	1	2	3	4	5	6	

(b)

Processors Operating	1 3	2 4	3 5	4 1	5 2	. . .
Time	1	2	3	4	5	

(c)

Figure 8.3.2 An example where optimal coloring cannot achieve the concurrency attainable by an optimal schedule. We consider the dependency graph in (a) and we require that no two neighbors operate simultaneously. Let processors 1 and 3 have color 1, processors 2 and 4 have color 2, and processor 5 have color 3. The synchronous schedule based on this coloring is shown in (b) and the average number of processors operating per time unit is 5/3. With the schedule in (c), however, there are two processors operating at each time unit.

At any point in time, either one of the two processors i and j is in control of R_{ij} or the resource is traveling from one processor to the other. When a processor i takes control of the resources R_{ij} associated with all of its incident arcs, it is allowed to perform an operation, and we assume that it will do so in finite time. Subsequently, and for each j such that $(i,j) \in A$, the resource R_{ij} is transferred to processor j. This is done by sending an appropriate message to each adjacent processor j. The resource is said to be traveling while this message is in transit and as soon as that message reaches processor j, the resource R_{ij} is under the control of processor j. The state of this scheduling procedure, at any given time, can be described compactly by assigning a particular direction to each arc $(i,j) \in A$, thus converting G to a directed graph. In particular, we orient arc (i,j) to point toward processor j if and only if the resource R_{ij} is under the control of j or is traveling toward j. Given a current orientation of the arcs in the graph G, we say that processor i is a *sink* if and only if all of its incident arcs $(i,j) \in A$ are oriented to point toward i. It is seen that a processor that is a sink eventually has control of all relevant resources and performs an operation. The subsequent transfer of these resources to the neighbors of i is then equivalent to reversing the directions of all arcs incident to i. We notice that if the previously described directed graph has no sink, then the graph remains forever unchanged. In that case, no processor ever becomes a sink, no processor ever operates, and the system is deadlocked. (This is the case, for example, in Fig. 8.3.1.) It will be shown that we can guarantee the existence of a sink at any given time, provided that the directions with which the scheduling algorithm is initialized make G a *directed acyclic graph*, that is, a directed graph with no cycles consisting exclusively of forward arcs.

In terms of the directed graphs just defined, the algorithm can be simply described as follows (see Fig. 8.3.3 for an illustration).

Arc Reversal Algorithm. The algorithm is initialized by assigning a direction to each arc $(i, j) \in A$ so that the resulting directed graph is acyclic. Any processor who is a sink performs an operation, within a finite amount of time, and reverses the directions of its incident arcs.

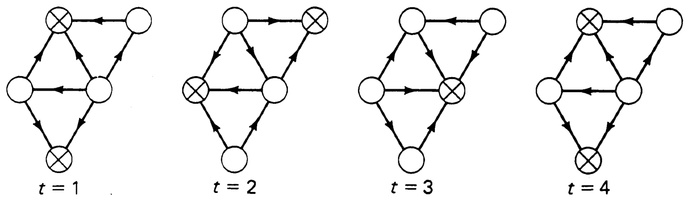


Figure 8.3.3 Illustration of the arc reversal algorithm. Processors marked with an X are those that are ready to operate and reverse the directions of their incident arcs.

We now verify that if a directed graph is acyclic and if the orientation of each arc incident to a sink is reversed, then the resulting directed graph is also acyclic. Let i be a sink that has performed an arc reversal. After the arc reversal, there can be no positive cycle going through i because there are no arcs oriented toward i. Furthermore, the directions of all arcs not incident to i are unchanged by the arc reversal. Assuming that the directed graph was acyclic before the arc reversal, we see that no positive cycles not involving i are created by the reversal. We conclude that the directed graph maintained by the arc reversal algorithm is at all times acyclic.

We next verify that any directed acyclic graph has a sink. To see this, suppose that there was no sink. Then we could construct an arbitrarily long path by starting at an arbitrary node and choosing each time an outgoing path from the current node. In particular, if the path has at least n arcs, then some node has to be visited at least twice, thereby contradicting acyclicity and proving the existence of a sink.

This discussion shows that at any given time, the directed graph maintained by the algorithm has a sink. It follows that for any given time t, there exists a subsequent time t' at which some processor operates. In particular, deadlock is never reached. We continue by verifying a fairness property of the algorithm. Let $X_i(t)$ be the number of operations performed by processor i until time t. Between consecutive operations by processor i, all of its neighbors must operate at least once, in order to reset the corresponding arcs so that they point toward i. It follows that $|X_i(t) - X_j(t)| \leq 1$ for all arcs (i, j). Thus, if the graph G is connected, we have

$$|X_i(t) - X_j(t)| \leq d, \qquad \forall i, j, \tag{3.2}$$

where d is the diameter of the graph. Hence, no processor can be arbitrarily far ahead of the others, as far as the number of operations is concerned.

In the context of the distributed execution of iteration (3.1), the scheduling algorithm is implemented as follows. Each arc is initially assigned a direction, so that the resulting directed graph is acyclic. Each processor i that is initially a sink learns the initial value of x_j for each neighbor j and updates x_i according to Eq. (3.1). It then transmits the new value of x_i to all its neighbors. When a neighbor j of i receives a value of x_i, it interprets this as a signal that the direction of the arc (j, i) has now been reversed to point toward j. Eventually, processor j becomes a sink, performs an update of x_j, etc. To summarize, it is seen that a processor performs an update as soon as it has received new values from all of its neighbors and transmits to them its new value as soon as it has performed its own update. The similarity with the local synchronization method of Section 1.4 should be evident. The only difference is that here we are dealing with a Gauss–Seidel rather than a Jacobi iteration, and that the algorithm uses the *undirected* dependency graph as a starting point.

We now investigate the choice of the directed acyclic graph with which the scheduling algorithm is initialized. For the sake of analysis, let us assume that the algorithm starts at time 1, that an operation by a processor takes exactly one time unit, and that communication between processors is instantaneous. (In particular, processors adjacent to a sink are instantaneously informed about the arc reversal when it occurs.) We finally assume that a processor starts its operation as soon as it becomes a sink. Let $G(t)$ denote the directed acyclic graph at time t. We notice that the set of all possible graphs $G(t)$ is finite, since there is only a finite number of possible orientations of the arcs in the graph G. Since $G(t)$ changes deterministically, it must eventually become a periodic function of time. We let $M(t)$ be the number of sinks in $G(t)$. We see that $M(t)$ eventually becomes periodic and, therefore, the limit

$$M = \lim_{t\to\infty} \frac{\sum_{\tau=1}^{t} M(\tau)}{t}$$

exists. The number M is the average number of operations per time unit and should be thought of as a measure of concurrency. We notice that

$$\sum_{\tau=1}^{t} M(\tau) = \sum_{i=1}^{n} X_i(t+1).$$

We divide both sides by nt, take the limit as t tends to infinity, and use inequality (3.2) to obtain

$$\frac{M}{n} = \lim_{t\to\infty} \frac{X_i(t)}{t}, \qquad \forall i.$$

The number M depends strongly on the directed acyclic graph $G(1)$ with which the algorithm is initialized. In particular, for the same underlying undirected graph G, M could be as large as $n/2$ and as small as 1 (see Fig. 8.3.4 for an example). It turns out that with an optimal choice of $G(1)$, the scheduling method under consideration achieves the maximum concurrency over all schemes where a processor must operate once between two consecutive operations by any neighbor [BaG87]. Unfortunately, the problem of choosing $G(1)$ so as to maximize M is an intractable combinatorial problem (NP–hard) [BaG87], although it is often easy to solve when G has a special structure (Exercise 3.2).

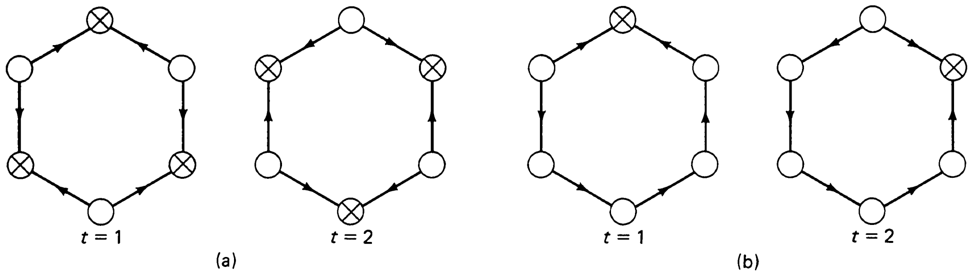


Figure 8.3.4 The effects of the initial assignment of directions to the arcs on the resulting measure of concurrency M. (a) A good choice of $G(1)$. Here $M = 3$. (b) A bad choice of $G(1)$. Here there is only one sink at a time and $M = 1$. More generally, if we have a ring of n nodes, with n even, a best choice of $G(1)$ leads to $M = n/2$, whereas a bad choice leads to $M = 1$.

EXERCISES

3.1. Let $G(1) = \big(N, A(1)\big)$ be the directed acyclic graph with which the arc reversal algorithm is initialized. Suppose that there exist nodes $i_1, \ldots, i_K \in N$ such that $(i_1, i_K) \in A(1)$ and $(i_k, i_{k+1}) \in A(1)$ for $k = 1, \ldots, K-1$. Show that the concurrency measure M is bounded above by n/K. *Hint:* See Figure 8.3.4.

3.2. Suppose that the graph G is a d–dimensional mesh. Assign initial orientations to the arcs of G so that the value of the concurrency measure M in the resulting scheduling algorithm, is as large as possible. *Hint:* Generalize the red–black ordering of Subsection 1.2.4 to the d–dimensional case.

8.4 SYNCHRONIZATION USING ROLLBACK: ASYNCHRONOUS SIMULATION

We have seen in Subsection 1.4.1 that a synchronous algorithm can be implemented in an inherently asynchronous parallel or distributed computing system, using either global or local synchronization methods. However, these methods have drawbacks that in certain contexts can be significant. For example, in global synchronization, several processors can be idle, waiting for other processors to complete the computations of the current phase. Also, the local synchronization method works well if each processor knows from which processors it is going to receive a message at the current phase, but involves substantial communication overhead (dummy messages) in case such knowledge is missing (see the discussion in Subsection 1.4.1). The synchronization method presented here is based on an entirely different principle: each processor keeps computing at its own pace under the assumption that no message is going to be received from other processors; in the case that such a message is actually received, the processor invalidates its computations and starts over again, taking the received message properly into account. This mechanism, called *rollback*, is a general purpose synchronization procedure and can be used in the implementation of any synchronous algorithm in an asynchronous computing system. It has been applied mainly in concurrency control of distributed databases and in the simulation of dynamical systems. We use simulation as our working example in this section, but it should be kept in mind that the method is of more general validity.

A Model of Discrete Time Dynamical Systems

We wish to simulate a discrete time dynamical system (referred to as the *physical system*) consisting of n interacting subsystems. It is natural to carry out the simulation using a parallel computing system with a different processor assigned to the task of simulating a different subsystem. The processors have to exchange messages, in the course of the simulation, in order to handle the interactions between subsystems.

To be more specific, let $\mathcal{S}_i$ be the ith subsystem and let its state at time t be described by a state variable $x_i(t)$ belonging to some state space X_i. Here t is an integer time variable, running from 0 to some final time T, which represents *physical* time in the system being simulated. It should be distinguished from *real* time, which is the time in the clock of an observer looking at the computing system used for the simulation. The initial states $x_i(0)$ in the physical system are assumed to be known. Let $G = (N, A)$ be a directed graph. The set N of nodes is equal to $\{1, \dots, n\}$, with node i representing subsystem $\mathcal{S}_i$. The presence of an arc (i, j) indicates the possibility that the state of subsystem $\mathcal{S}_i$ influences the state of subsystem $\mathcal{S}_j$. For any $(i, j) \in A$ and any (physical) time t, let $m_{ij}(t)$ be a variable, taking values in a space M_{ij}, which models the interaction of subsystems $\mathcal{S}_i$ and $\mathcal{S}_j$ in a way to be made precise shortly. We postulate a functional dependence of the form

$$m_{ij}(t) = g_{ij}\big(x_i(t)\big), \qquad (i, j) \in A, \tag{4.1}$$

where g_{ij} is a function from X_i into M_{ij}. We wish to allow for the possibility that even if $(i, j) \in A$, subsystems $\mathcal{S}_i$ and $\mathcal{S}_j$ do not interact at every time instance. We thus assume that m_{ij} can take a special (null) value, denoted by π. The equality $m_{ij}(t) = \pi$ is interpreted as the absence of any interaction from $\mathcal{S}_i$ to $\mathcal{S}_j$ at physical time t. Let $z_i(t)$ be the vector consisting of all interaction variables $m_{ji}(t)$, where $(j, i) \in A$. We assume that we have a model of the form

$$x_i(t + 1) = f_i\big(x_i(t), z_i(t)\big), \tag{4.2}$$

for each subsystem $\mathcal{S}_i$. Equations (4.1) and (4.2) define completely the computations that we would like to carry out. In fact, such equations can be used to describe a broad variety of synchronous algorithms and for this reason, the discussion that follows is of much more general applicability.

A Synchronous Simulation Algorithm

We assign a separate processor P_i to each subsystem S_i. The computation proceeds as follows. If processor P_i knows the value of $x_i(t)$ and has received a message from processor P_j containing the value of $m_{ji}(t)$ for every j such that $(j, i) \in A$, then processor P_i computes $x_i(t+1)$ and $m_{ik}(t+1)$ for each k such that $(i, k) \in A$, using Eqs. (4.2) and (4.1), respectively. Subsequently, for every k such that $(i, k) \in A$, the message $m_{ik}(t+1)$ is transmitted to the corresponding processor P_k. [Such a message is transmitted even if $m_{ik}(t+1)$ has the null value π.] We allow for the possibility that messages are not received in the order in which they were transmitted. In order for the receiver to be able to interpret correctly the received messages, we assume that the messages also contain a timestamp (see Fig. 8.4.1). For concreteness, let us assume that messages have the format (t, m, i, j). Reception of such a message informs the recipient that $m_{ij}(t) = m$. We refer to t as the *timestamp* of the message.

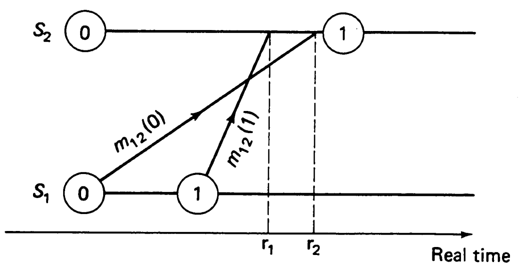


Figure 8.4.1 Illustration of the need for timestamps. Here there are two subsystems and subsystem S_1 affects S_2, but not conversely $[A = \{(1,2)\}]$. The nodes in this figure indicate the real times at which the values of the variables $x_i(t)$ are computed. Processor P_2 receives two messages at real times r_1 and r_2, respectively, but if these messages carry no timestamps, it has no way of telling which one is $m_{12}(0)$ and which one is $m_{12}(1)$.

Assuming that all messages eventually reach their destinations, it is evident that the previously described algorithm correctly simulates the system described by Eqs. (4.1) and (4.2). Consider the special case where each stage requires C time units of computation by each processor and assume that each message reaches its destination with a delay of D time units. Then the total time until the completion of the simulation is equal to $(C + D)T$. This is the best possible if the subsystems interact all of the time.

The method just described is precisely the local synchronization method of Subsection 1.4.1. It is an appropriate method if subsystems S_i and S_j interact at every physical time step for each i and j such that $(i, j) \in A$. On the other hand, if such interactions are rarely present, then $m_{ij}(t)$ is equal to the null value π, most of the time. Then processor P_j will often have to wait to receive a message m_{ij}, only to discover that it carries the uninformative value π. The objective of the synchronization method to be presented is precisely to avoid waiting for these null messages.

The Rollback Mechanism

Let us consider the following scheme. Each processor continues updating according to Eq. (4.2) and keeps sending messages containing values of the interaction variables, generated according to Eq. (4.1). We assume again that messages have the generic format (t, m, i, j) and that they are guaranteed to reach their destination after some finite but unknown real time and not necessarily in the order that they were sent. Whenever processor P_i needs [in order to execute Eq. (4.2)] the value of some interaction variable $m_{ji}(t)$ and a message (t, m, j, i) has not been received, then processor P_i uses the default value π. In effect, processor P_i takes a gamble that interactions will be absent. If interactions are indeed absent for all times and for all pairs of subsystems, then it is evident that the simulation is completed in the least possible time, since each processor continues computing at full speed. Now suppose that during the simulation, a nonnull interaction variable $m_{ij}(t) \neq \pi$ is generated. When processor P_j receives the corresponding message (t, m, i, j), there are two distinct possibilities that we now discuss.

Suppose that at the real time instance when $m_{ij}(t)$ is received, $x_j(t+1)$ has not been computed. Then when processor P_j eventually reaches the point where $x_j(t+1)$ is to be generated, it will have available and will be able to use the correct value of $m_{ij}(t)$.

Now suppose that processor P_j has already computed $x_j(t+1)$, before receiving $m_{ij}(t)$, while assuming, incorrectly, that $m_{ij}(t) = \pi$. Then the value of $x_j(t+1)$ and all subsequent values $x_j(t')$, $t' > t$, that have already been computed are incorrect and must be recomputed. (We then say that processor P_j is *rolling back*; see Fig. 8.4.2.) In order to recompute these values, processor P_j must remember the value of $x_j(t)$ and all messages $m_{kj}(t')$, $t' \geq t$, it has received. Furthermore, all messages $m_{jk}(t')$, $t' > t$, sent by processor P_j are incorrect because they were evaluated from Eq. (4.1) on the basis of an incorrect value of $x_j(t')$. Therefore, processor P_j has to transmit special messages to the other processors, informing them that certain messages transmitted in the past are invalid. We refer to these special messages as *antimessages*. For concreteness, we assume that the antimessage invalidating the message (t, m, j, k) has the format $(t, m, j, k, *)$. Notice that the transmission of appropriate antimessages requires each processor to keep a record of the messages it has transmitted in the past. Whenever a processor P_k receives an antimessage $(t, m, j, k, *)$, some of the results of earlier computations by that processor can be invalidated, which can invalidate some messages that have been already sent by P_k. Thus, P_k may have to send antimessages as well. In particular, each antimessage can trigger the transmission of an arbitrary number of further antimessages (see Fig. 8.4.3). Finally, notice that a processor can receive an antimessage $(t, m, i, j, *)$ without first receiving the corresponding message (t, m, i, j), because we allow messages to be received out of order, and we must specify how such a situation is to be handled.

We continue with a full description of the simulation algorithm, but we first make the key observation that null messages of the form (t, π, i, j) are inconsequential: processor P_j performs the same computations whether or not such a message has been received. The reason is that π is the default value to be used in the absence of a message. It follows that processor P_i does not have to transmit a message if the value of the interaction variable $m_{ij}(t)$ is equal to π. This modification can save a substantial amount of communication and is one of the reasons for introducing the rollback mechanism.

We assume that for each $(i, j) \in A$, there is a buffer B_{ij} in which messages from P_i to P_j are placed upon reception until they are processed by P_j. We also assume that each processor P_i keeps the following in its memory:

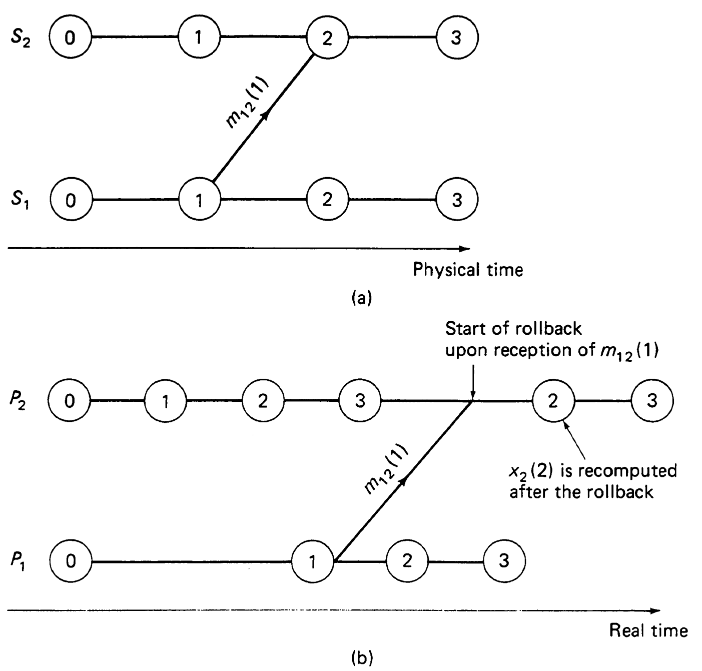

S_2
S_1
$m_{12}(1)$
Physical time
(a)
Start of rollback upon reception of $m_{12}(1)$
P_2
P_1
$x_2(2)$ is recomputed after the rollback
Real time
(b)

Figure 8.4.2 Illustration of a rollback. (a) A possible scenario for the physical system being simulated. Here the state of subsystem S_1 at time 1 affects the state of S_2 at time 2. (b) A possible scenario for the simulation. A node labeled t stands for the point in time that $x_i(t)$ was computed by processor P_i. In this scenario, processor P_1 is slow and the communication delay of the message $m_{12}(1)$ is large enough so that this message reaches processor P_2 after $x_2(3)$ is computed. Receipt of such a message invalidates the computation of $x_2(2)$ and $x_2(3)$, which have to be recomputed.

(a) An integer variable τ_i initialized with $\tau_i = -1$. [This variable is interpreted as the largest value of t such that $x_i(t)$ has been computed and the computed value has not been invalidated.]

(b) A record containing a value $x_i(\tau)$ for every τ such that $0 \le \tau \le \tau_i$.

(c) A record of all messages (t, m, j, i) it has received and processed and for which a corresponding antimessage $(t, m, j, i, *)$ has not been received and processed.

(d) A record of all antimessages $(t, m, j, i, *)$ it has received and processed and for which a corresponding message (t, m, j, i) has not been received and processed.

(e) A record of all messages (t, m, i, j) it has sent and for which it has not sent a corresponding antimessage.

Initially, the records (b) to (e) are empty. During the algorithm, processor P_i executes the following three instructions. (Several improvements are possible; see e.g. Exercise 4.2.)

1. If a message (t, m, j, i) exists in one of the buffers B_{ji}, remove it from the buffer and do the following:

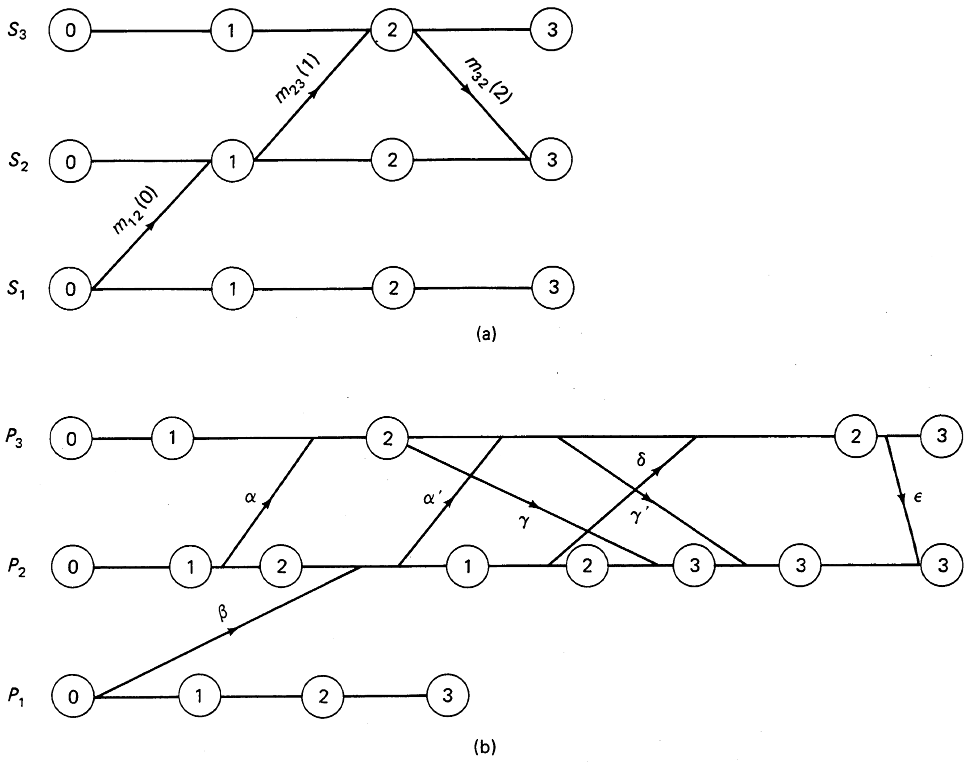


Figure 8.4.3 Illustration of a rollback that causes a further rollback at another processor. (a) A possible scenario for a physical system consisting of three subsystems. Here the arrows stand for nonnull interactions. For example, $x_1(0)$ affects $x_2(1)$ by means of an interaction $m_{12}(0)$. (b) A corresponding possible scenario for the simulation. Processor P_2 computes $x_2(1)$ and sends a message $\alpha = m_{23}(1)$ to processor P_3. However, the message $\beta = m_{12}(0)$ reaches processor P_2 after $x_2(2)$ has been computed, causes a rollback, and invalidates the values of $x_2(1)$ and $x_2(2)$. This rollback triggers an antimessage α' from processor P_2 to processor P_3 to invalidate the message α. In the meantime, processor P_3 has sent a message γ to processor P_2 based on its computation of $x_3(2)$. However, when the antimessage α' is received, the value of $x_3(2)$ is invalidated and an antimessage γ' is sent to processor P_2 to cancel the message γ. Eventually, $x_2(1)$ is recomputed and a new message δ is sent to processor P_3, carrying the correct value of $m_{23}(1)$. Then processor P_3 eventually computes the correct value of $x_3(2)$ and sends a message ϵ with the correct value of $m_{32}(2)$. This message causes one more rollback at processor P_2 which recomputes $x_2(3)$.

1.1. If the antimessage $(t, m, j, i, *)$ is found in the record (d) of antimessages, then delete it and discard the message (t, m, j, i).

1.2. If the antimessage $(t, m, j, i, *)$ is not found in the record (d), then place the message (t, m, j, i) in the record (c) of messages received and processed. If $t \geq \tau_i$, no further action is required. If $t < \tau_i$, then do the following:

1.2.1. (Rolling back) Assign to the variable τ_i the new value t and delete $x_i(t')$ from record (b) for all $t' > t$.

1.2.2. (Canceling incorrect messages) Remove all messages of the form (t', m, i, k), $t' > t$, from the record (e) of messages sent and send corresponding antimessages $(t', m, i, k, *)$.

2. If an antimessage $(t, m, j, i, *)$ exists in one of the buffers B_{ji}, remove it from the buffer and do the following:

2.1. If the message (t, m, j, i) is not found in the record (c) of messages, then place the antimessage in the record (d) of antimessages received and processed.

2.2. If the message (t, m, j, i) is found in the record (c), then delete it from that record and discard the antimessage. If $t \geq \tau_i$, no further action is required. If $t < \tau_i$, then do the following:

2.2.1. Assign to the variable τ_i the new value t and delete $x_i(t')$ from record (b) for all $t' > t$.

2.2.2. Remove all messages of the form (t', m, i, k), $t' > t$, from the record (e) of messages sent and send corresponding antimessages $(t', m, i, k, *)$.

3. If $\tau_i < T$, then do the following:

3.1. Compute $x_i(\tau_i + 1)$ and $m_{ik}(\tau_i + 1)$ for each k such that $(i, k) \in A$, using Eqs. (4.1) and (4.2). [In the special case where $\tau_i = -1$, instead of using Eq. (4.2), simply fetch the value of $x_i(0)$, which is viewed as an external input.] If the computation of $x_i(\tau_i + 1)$ requires the value of $m_{ji}(\tau_i)$ [i.e., if $(j, i) \in A$], look for it in the record of messages received and processed. If it is not found, use the default value π. If several messages from P_j with the same timestamp τ_i are found in that record, choose one of them arbitrarily.

3.2. Increment τ_i to $\tau_i + 1$.

3.3. If $\tau_i < T$, then for each k such that $(i, k) \in A$ and $m_{ik}(\tau_i) \neq \pi$, send a message (τ_i, m, i, k) to processor P_k, with $m = m_{ik}(\tau_i)$, and place this message in the record (e) of messages sent.

The algorithm terminates when $\tau_i = T$ for each i, and there are no messages or antimessages that have been transmitted but have not yet been received and processed.

We assume that each processor keeps executing instructions 1, 2, and 3 at its own pace and in an arbitrary order. For example, if several messages and antimessages have been placed in the buffers B_{ji} of processor P_i, then this processor has the option of executing either instruction 1 or 2, and handle any one of the received and yet unprocessed messages and antimessages. Still, we assume that every transmitted message or antimessage is received correctly and is processed after a finite amount of time, in the course of some execution of instruction 1 or 2. We also assume that if $\tau_i < T$, then processor P_i will eventually execute instruction 3. It is assumed that each one of the three instructions is indivisible: for example, processor P_i will never interrupt the execution of instruction 2 to start executing instruction 1.

We now argue that the algorithm eventually terminates with the correct values of $x_i(t)$ for each i and $t \in \{0, 1, \ldots, T\}$, where a value is called *correct* if it is the one generated by a synchronous execution of Eqs. (4.1) and (4.2). To reach this conclusion, we will show that for each physical time t, there exists some real time r_t such that the following are true:

(a) We have $\tau_i \geq t$ for each i.

(b) There are no messages or antimessages in transit (i.e., sent but not yet received and processed) with a timestamp smaller than t.

We start by noticing that if (a) and (b) are true at some real time r_t, then they remain true for all real times $r > r_t$. To see this, we examine the nature of instructions 1 to 3 and verify that there is no possibility for (a) or (b) to become false. For example, if instruction 3 is executed, τ_i becomes $\tau_i + 1 \geq t + 1 \geq t$, messages generated have the timestamp $\tau_i + 1 \geq t + 1$, and no antimessages are generated. If instruction 1 is executed, it involves a message with a timestamp no smaller than t. Thus, even if a rollback occurs, the new value of τ will be no smaller than t, and any antimessages caused by this rollback have a timestamp no smaller than $t + 1$. If instruction 2 is executed, the argument is similar.

We now proceed by induction. We observe that for $t = 0$, condition (b) always holds and condition (a) becomes and remains true as soon as instruction 3 is executed at least once by each processor. Consider now some $t < T$ and suppose that (a) and (b) are true at some real time r_t. The argument of the preceding paragraph shows that (a) and (b) remain forever true and, furthermore, no new messages or antimessages with a timestamp smaller than $t+1$ will be generated in the future. In particular, any messages and antimessages with a timestamp t will eventually reach their destinations and be processed. (This argument is correct if we assume that there is only a finite number of messages or antimessages in transit. Otherwise, it might take an infinite amount of time until all of them are received and processed, even though each one is received and processed in finite time. This issue is, however, of no concern because it can be proved that only a finite number of messages and antimessages is generated in the course of the algorithm; see Exercise 4.1.) We conclude that there exists some real time r'_t after which condition (b) will be satisfied for $t + 1$. Similarly, after real time r_t, the value of each τ_i never becomes smaller than t. Moreover, at the first time after r'_t that instruction 3 is executed, τ_i is incremented to a value no smaller than $t + 1$ and is never going to decrease below $t + 1$ because no message or antimessage with a timestamp smaller than $t + 1$ will be received in the future. Therefore, there exists a real time $r_{t+1} > r'_t$ such that condition (a) is true for every real time $r \geq r_{t+1}$.

We conclude that there exists some time r_T at which $\tau_i = T$, for each i, no messages or antimessages are in transit, and, therefore, the algorithm has terminated. It remains to show that at termination, all the values $x_i(t)$ and $m_{ij}(t)$ in the records of the processors are correct for each t. This is obviously true for $t = 0$ and we proceed again by induction. Assuming that it is true for some physical time t, the messages $m_{ij}(t)$ in the buffers of processor P_j have the correct values and this implies that the value of $x_j(t + 1)$ is also correct. Therefore, the only messages $m_{jk}(t + 1)$ that have been sent and have not been invalidated by subsequent antimessages have the correct values. Since, at termination, there are no messages or antimessages in transit, it follows that the messages $m_{jk}(t + 1)$ in the buffers of all processors P_k have the correct values, which completes the induction.

Let us now make the additional assumption that messages and antimessages are received in the same order as they are transmitted. In this case, the antimessages invalidating past erroneous values of $m_{ji}(t)$ reach processor P_i before the final correct message. Therefore, as soon as the correct messages are received, processor P_i can compute the correct value of $x_i(t)$ without any further delay. We conclude that in this case, the algorithm proceeds as fast as possible: correct values are computed at the first instance of time when the required data have been received. To be more specific, suppose that computations take zero time and the transmission delay of a message or antimessage from P_i to P_j takes T_{ij} time units. Let $r_{i,t}$ be the real time of the last (and, therefore, correct) computation of $x_i(t)$. Assume that $r_{i,0} = 0$ and it can be seen that

$$r_{i,t+1} = \max\Big\{r_{i,t}, \max\Big\{r_{j,t} + T_{ji} \mid (j,i) \in A,\ m_{ji}(t) \neq \pi\Big\}\Big\},$$

where we use the convention that $\max\{c \mid c \in C\} = -\infty$ when C is the empty set. On the other hand, given our assumptions on communication delays, no other algorithm could evaluate $x_i(t)$ before real time $r_{i,t}$, even if we knew ahead of time which of the interaction variables $m_{ij}(t)$ are absent (null). We therefore have an optimal algorithm as far as time is concerned.

The argument in the preceding paragraph refers to an idealized situation. It is based on the assumption that communication delays are independent of the particular algorithm being used and ignores the overhead associated with the rollback mechanism. In fact, rollback involves several types of overhead:

(a) Computational overhead due to invalid computations: computational resources are wasted in computing often incorrect values of $x_i(t)$ to be invalidated later. Suppose that a processor can stop a computation immediately upon reception of an invalidating message and start a new computation using the message value received. Then the computational overhead does not slow down the computation because any processor engaged in a computation to be invalidated later only has the alternative option of staying idle. On the other hand, if computations cannot be interrupted and take a long time to be completed, this overhead should be taken into account.

(b) Communication overhead: this can be quite severe in the worst case. One can envisage multiple rollbacks triggering further rollbacks, each one generating large numbers of antimessages that result in queueing delays. Even if communication resources are abundant, excessive communication can drain the computational resources of the processors, since each message requires some amount of processing. On the other hand, it is hoped that for a class of interesting systems, rollbacks will exhibit some temporal and spatial locality: if a processor rolls back, it may only roll back a few time units, and if any further rollbacks are triggered, these will involve only a few neighboring processors. However, this hypothesis has not been established theoretically or experimentally as yet.

(c) Finally, the algorithm has significant memory requirements. Processors have to keep records of all messages sent and received, and of the values of $x_i(t)$ that have been computed. There are two partial remedies to this problem that we now discuss.

Assuming that messages are relatively infrequent, the main memory requirement comes from having to remember old values of $x_i(t)$. This requirement is reduced if a processor saves $x_i(t)$ only once in a while, say once every A time units, where A is some integer. This introduces a new problem: if a processor has to roll back to some time t for which the value of $x_i(t)$ has not been saved, then it must go even further back, use an earlier value of x_i, and reconstruct $x_i(t)$. So, the savings in memory are offset by increased computational requirements each time that a rollback occurs. A reasonable compromise is to save all values $x_i(t)$ for t close to τ_i, and save a few old values just in case that a rollback to the distant past occurs. Again, whether this scheme would work well depends on the validity of the hypothesis of temporal locality of rollbacks.

Another reduction of the memory requirements is suggested by the following observation. If $\tau_i \geq t$ for all i, and if there are no messages or antimessages in transit (transmitted but not received and processed) with timestamps smaller than t, then there is nothing that could cause a rollback resetting τ_i to a value smaller than t. This means that the values of $x_i(\tau)$ and $m_{ij}(\tau)$ for $\tau < t$ can be discarded. Unfortunately, such a memory management mechanism cannot be easily implemented in a distributed manner, because it involves global properties of the algorithm, as opposed to the local information available to each processor. It can be implemented, however, by employing the snapshot algorithm of Section 8.2 (under the FIFO assumption). To see this, we say that the state of computation of processor P_i belongs to the set X_i^* if $\tau_i \geq t$ and any messages or antimessages in the buffers B_{ji} (that is, received but not processed) have timestamps greater than or equal to t. Also, we let M_{ij}^* be the set of all possible messages or antimessages from i to j with timestamps greater than or equal to t. These sets have the invariance properties considered in Section 8.2, and, therefore, the snapshot algorithm can be used to detect whether the local state of computation of each processor P_i belongs to X_i^* and each message or antimessage in transit from i to j belongs to M_{ij}^*.

A related problem with the algorithm is that the processors cannot detect termination based on their own local information. Rather, termination has to be detected either by a host computer supervising the simulation or by a distributed termination detection algorithm of the type discussed in Section 8.1. Alternatively, if the snapshot algorithm is used to monitor the progress of the algorithm, the snapshots can also be used for the purpose of termination detection.

Simulation of Continuous Time Discrete Event Systems

Asynchronous distributed simulation with rollback has been actually proposed not for discrete time systems described by equations of the form (4.1)–(4.2), but for so called *discrete event* systems. These latter systems evolve in continuous time, but their state changes only at a discrete set of times. (An example of a discrete event system is a queueing network with random arrivals and service times.) Some new difficulties arise because the times at which these discrete events can occur are not a priori known.

We describe a model of a discrete event system and suggest some algorithms. The issues are similar to those discussed in the context of the simulation of discrete time systems, and for this reason, the presentation here will be less detailed.

We again have n subsystems $\mathcal{S}_1, \ldots, \mathcal{S}_n$ and a directed graph $G = (N, A)$ indicating the possible interactions between subsystems. Let $x_i(t)$ be the state of the ith subsystem at time t. Here t is a continuous time variable, taking values in $[0, T]$. We constrain $x_i(t)$ to be a piecewise constant function of time, and for concreteness, we assume that it is right–continuous. The times of discontinuity of $x_i(t)$ are called *event times*. Whenever an event occurs at subsystem $\mathcal{S}_i$, it can trigger an interaction with another subsystem $\mathcal{S}_j$, assuming that $(i, j) \in A$. Whether this is the case or not depends on the state $x_i(t)$ just after the event. To keep the discussion simple, we assume that interactions are not instantaneous, that is, if an event occurs at subsystem $\mathcal{S}_i$ at time t, the resulting interaction can affect another subsystem $\mathcal{S}_j$ only after some positive amount of time, although we allow this time to be arbitrarily small. At the time that such an interaction affects subsystem $\mathcal{S}_j$, an event is triggered at the latter subsystem. Events can also be self–induced. In particular, if an event occurs at subsystem $\mathcal{S}_i$ at time t, then a new event time t' is generated, as a function of the state $x_i(t)$ right after the event. The value of t' is the time of the next event at subsystem $\mathcal{S}_i$, unless an event is triggered earlier due to an interaction from another subsystem.

The above description of a discrete event system is not very rigorous. The following sequential algorithm simulates a discrete event system, and can also be taken as a definition of the dynamics of such a system. Assume that the system has been simulated up to a certain time t, including t itself. Each event that has already occurred determines the times at which certain events will occur in the future. Choose that event that is scheduled to occur first among all those future events. (For simplicity, we prohibit simultaneous events at different subsystems. This is no loss of generality: since we do not allow simultaneous interactions, if two events were to occur simultaneously, they could not be causally dependent and their times could be perturbed by an infinitesimal amount to make them appear non–simultaneous.) Suppose that the first future event is scheduled to occur at subsystem $\mathcal{S}_i$ at time $t' > t$. We can then simulate all subsystems up to time t' as follows. No events occur at any of the remaining subsystems during the interval $[t, t']$, and, therefore, the state of these subsystems remains unchanged. Concerning subsystem $\mathcal{S}_i$, we have to simulate the particular event under consideration. That is, we have to generate the new state of $\mathcal{S}_i$ after the event, determine the times of interaction with other subsystems caused by this event (if any), and determine the time of the next self–generated event at subsystem $\mathcal{S}_i$. We can then proceed to the simulation of the next event.

The above algorithm can be used to simulate an arbitrary number of events. In particular, if only a finite number of events can occur during a finite time interval, the algorithm can simulate the system for any arbitrary length of time. On the other hand, this algorithm is inherently sequential, because after the occurrence of each event, we need to determine the minimum event time over all events due to occur in the future and all subsystems. Several modifications of this algorithm which are more parallelizable have been studied. However, they are susceptible to deadlock and require, in general, special deadlock resolution procedures or special assumptions on the nature of interactions between subsystems [Mis86].

An alternative distributed simulation algorithm is based on the rollback scheme and is almost identical to the algorithm provided earlier for discrete time systems. Each processor P_i keeps simulating its subsystem $\mathcal{S}_i$ arbitrarily far into the future. At each stage, it determines the time of the next event due to occur at $\mathcal{S}_i$, using the information available. Whenever an event is simulated by P_i, messages are transmitted to other processors P_j to inform them about events to be triggered at $\mathcal{S}_j$ caused by the event simulated at $\mathcal{S}_i$. If processor P_j receives a message stating that an event is to occur at some time t at $\mathcal{S}_j$ and processor P_j has already simulated $\mathcal{S}_j$ up to some time $t' > t$, then it has to undo and repeat its simulation for the interval $[t, t']$. As before, appropriate antimessages should be sent to invalidate any incorrect messages. The details are the same as for the simulation of discrete time systems. In particular, if we assume that all messages and antimessages reach their destinations and are processed in finite (real) time, and if only a finite number of events can occur during a finite (physical) time interval, then the rollback algorithm eventually simulates the system correctly, arbitrarily far into the future.

An Example: Shortest Path Computation

In the example to be discussed here, the physical system to be simulated involves a directed graph $G = (N, A)$. With each arc (i, j), there is an associated positive arc length a_{ij}. The physical system operates as follows: initially, all nodes have a state $x_i(0) = \infty$, except for node 1 for which $x_i(0) = 0$. Node 1 emits a signal along its outgoing arcs. When a node receives for the first time a signal along one of its incoming arcs, it immediately retransmits it along all of its outgoing arcs. Signals to be received later are not retransmitted. Let t_i be the time at which subsystem $\mathcal{S}_i$ receives a signal for the first time. We define the dynamics of the state variables x_i: we have $x_i(t) = \infty$ for $t < t_i$, and when a signal is first received by node i at time t_i, the value of x_i becomes t_i and stays at that value thereafter. Assume that the travel time of the signal along an arc (i, j) is equal to the arc length a_{ij}. It is then obvious that t_i is equal to the length of a shortest path from node 1 to node i.

This physical system can be viewed as a discrete event system. The event times are the times at which the nodes receive signals. Notice that the state of a node changes only at the first event at that node. We now describe a simulation of the system using the rollback algorithm.

We use a network of n processors interconnected according to the topology determined by the graph G. The algorithm to be executed by processor P_i is the following: initially, $x_i(0) = \infty$ for $i \neq 1$, and $x_1(0) = 0$. Messages along an arc of the computing system simulate signals along the same arc in the physical system. The only useful information in these signals is the physical time at which the signal is received. Accordingly, messages only carry a timestamp indicating the time that the corresponding signal should reach its destination in the physical system.

Recall that the essence of the rollback scheme is that each processor simulates its own subsystem, [the time function $x_i(\cdot)$] as far into the future as it can under the assumption that all messages it has received are correct, and that no further messages are to be received. Notice that in the present example, the time function $x_i(\cdot)$ corresponding to a set of messages is straightforward to determine: $x_i(t)$ is infinite for t less than the smallest timestamp s_i of the received, processed, and not invalidated messages, and $x_i(t)$ is equal to s_i thereafter. Thus, the time function $x_i(\cdot)$ is uniquely determined by the single number s_i, and we can assume that processor P_i only keeps s_i in its memory, rather than the time function $x_i(\cdot)$. It follows that the rollback scheme amounts to the following. Each processor P_i, $i \neq 1$, keeps a number s_i initialized at infinity. Whenever a processor P_i receives a message, it reads the timestamp t. If $t \geq s_i$, it leaves s_i unchanged. If $t < s_i$, then s_i is reset to t. The messages sent by P_i, triggered by the reception of a message with timestamp t, are determined as follows. If $t \geq s_i$, no message is sent, corresponding to the fact that a signal is emitted from the subsystem $\mathcal{S}_i$ only at the first time that the subsystem receives a signal. If on the other hand, $t < s_i$, then the old value of s_i is invalidated, which invalidates the signals already sent by P_i. Thus, processor P_i must sent antimessages as well as new messages based on the new value of s_i. The message sent to processor P_j should be equal to the sum of the new value of s_i and a_{ij}; this is the time at which a signal is received by $\mathcal{S}_j$ if this signal was emitted from $\mathcal{S}_i$ at a time equal to the new value of s_i. In fact, it is not hard to see that antimessages are redundant in this context. If processor P_j receives a new message with a smaller timestamp it automatically knows that the old messages are invalid.

An equivalent and more compact description of the above scheme is the following. Each processor P_i, $i \neq 1$, maintains a number s_i initialized at infinity. Whenever processor P_j receives a message from processor P_i with a timestamp $s'_j = s_i + a_{jj}$ which is less than s_j, it resets s_j to be equal to s'_j and sends a message $s'_j + a_{jk}$ to every processor k such that $(j,k) \in A$. It is assumed that all messages are received after some unspecified delay. We recognize this as being equivalent to the asynchronous Bellman–Ford algorithm analyzed in Section 6.4. We now recall that the asynchronous Bellman–Ford algorithm has, under a worst case scenario, exponential communication complexity (see Fig. 6.4.2 in Section 6.4). This proves, in particular, that it is possible for the rollback scheme to generate a number of messages and antimessages that is an exponential function of the number of nonnull interactions in the physical system being simulated.

EXERCISES

4.1. Show that the total number of messages and antimessages generated during the asynchronous simulation of a discrete time system over a finite time interval $[0, T]$, using the rollback mechanism, is bounded by some function of T and the number of subsystems. *Hint:* Show that the number of rollbacks that reset τ_i to a value less than or equal to t is bounded by a function of the number of messages and antimessages with a timestamp smaller than or equal to t. Then bound the number of messages and antimessages with a timestamp equal to $t+1$ and proceed by induction.

4.2. Consider a processor P_i that computes $x_i(t+1)$ by executing instruction 3. Suppose, furthermore, that processor P_i finds two messages (t, m, j, i) and (t, m', j, i) in the record of messages received and processed, and that the message (t, m, j, i) is used to compute $x_i(t+1)$. Suppose that the antimessage $(t, m', j, i, *)$ is received later. Such an antimessage causes a rollback at processor P_i and the recomputation of $x_i(t+1)$ (cf. instruction 2). On the other hand, it is intuitively clear that if a message has not been used in computations, its cancellation by a corresponding antimessage does not invalidate any computations and should not cause a rollback.

(a) Modify the algorithm so that no rollback occurs in a situation such as the one described above and explain why the algorithm still produces the correct results.

(b) Show that the situation under consideration cannot arise if messages and antimessages are received in the order in which they are transmitted.

8.5 MAINTAINING COMMUNICATION WITH A CENTER

We consider an asynchronous network of processors described by an undirected graph $G = (N, A)$, where $N = \{0, 1, \ldots, n\}$ is the set of processors, and A is the set of undirected arcs. Each arc (i, j) indicates the existence of a bidirectional communication link between processors i and j. Processor 0 represents a center and we are interested in an algorithm that guarantees that each processor maintains a set of simple (loop–free) paths through which it can communicate with the center. Furthermore, we would like such an algorithm to be able to adapt itself to unpredictable topological changes such as the removal or the addition of communication links. Finally, the algorithm should be distributed, with the actions of each processor depending only on locally available information.

The above problem has applications in a few different contexts. For example, in certain data networks, such as mobile packet radio networks, topological changes can be very frequent and the task of maintaining communication with a center can be quite challenging. Another context is provided by geographically distributed sensor networks in which data obtained by the sensors must be relayed to a central processing station. Finally, one might envisage loosely coupled distributed computing systems operating in an uncertain environment with failure–prone communications.

A simple solution to our problem could be a *centralized* one. Here the processors transmit topological information to the center. Then the center, based on its knowledge of the topology of the network, computes a set of simple paths, from each processor to the center, and communicates these paths to the individual processors. This is not as straightforward as it seems because for the processors to transmit topological information to the center, they need to have already established communication paths, which is the problem that we were trying to solve in the first place. For this reason, the task of transmitting the topological information to the center becomes rather complex. It can be solved by flooding or by running a topology broadcast algorithm (see the end of this section). Both of these solutions can be undesirable, however, because they can involve an excessive amount of message transmissions and considerable overhead.

A distributed alternative is based on the solution of a shortest path problem. Let us assign a length of 1 to every arc in the network and let the processors execute the asynchronous Bellman–Ford algorithm, with processor 0 playing the role of the destination. If the topology of the network does not change and if the network is connected, this algorithm will eventually terminate and a set of shortest paths from every processor to the center will be obtained. Since the arc lengths are positive, such shortest paths will be simple, as desired (Fig. 8.5.1). Furthermore, if topological changes occur,

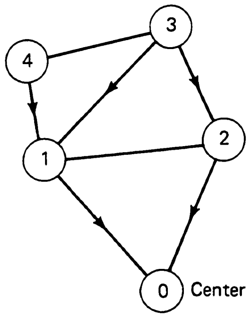


Figure 8.5.1 Simple paths from every processor to the center obtained by solving a shortest path problem. An arrow on the arc (i, j) indicates that this arc belongs to a shortest path from i to the center.

the processors will keep executing the Bellman–Ford algorithm, which is then guaranteed to converge to a set of shortest paths for the new topology of the network.

While the shortest path approach provides a distributed algorithm for our problem, it has two undesirable properties:

(a) The paths obtained when the algorithm reaches quiescence are guaranteed to be simple, but this is not necessarily true while the algorithm is executing. If a processor attempts to communicate with the center before the algorithm has terminated, the messages it sends might travel in a cycle before they reach their destination (Fig. 8.5.2).

(b) If a topological change (say the addition of an arc) does not disrupt previously established paths from the processors to the center, it might be reasonable to continue using these paths. Nevertheless, the shortest path algorithm may lead to major changes in the paths (Fig. 8.5.3).

We now proceed to develop a class of algorithms for the problem under consideration. In these algorithms, starting with the original undirected graph G, we convert it into a directed graph G' by assigning a direction to each one of its arcs. If the assigned directions are such that G' is acyclic (i.e., there are no cycles consisting exclusively of forward arcs), we say that G' is a directed acyclic graph (*DAG*). If G' is a DAG and node 0 is the only sink, we say that G' is *oriented* toward the center; otherwise, we call it *disoriented*. Notice that if we manage to assign directions to the arcs so that G' is acyclic and oriented, we have obtained a solution to our problem. In particular, starting from any node i, consider any path obtained by traversing consecutive arcs in accordance to their assigned directions. Such a path must eventually come to an end because G' is acyclic. When it comes to an end, a sink must have been reached and, since G' is oriented, that sink must be node 0 (Fig. 8.5.4).

The algorithms to be presented maintain at all times a direction for each arc and rely on the following properties:

Property A. The algorithm is initialized by assigning directions to the arcs so that the resulting directed graph is acyclic.

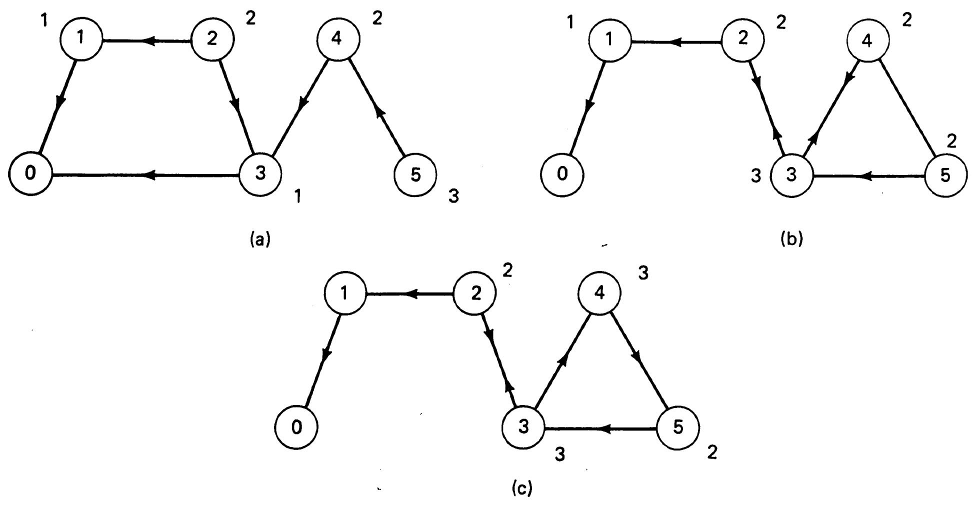


Figure 8.5.2 Formation of a cycle when the paths are chosen by a shortest path algorithm. (a) Original shortest paths. The labels next to each node indicate estimates of the distance from the center. Also, an arrow on arc (i, j) pointing toward j indicates that node i perceives this arc as belonging to a shortest path to the center. (b) Arc (3,0) is removed from the network and arc (5,3) is added. Processors 3 and 5 perform simultaneously an iteration of the Bellman–Ford algorithm to obtain new distance estimates of 3 and 2, respectively. (c) Processor 4 performs an iteration to obtain a distance estimate of 3. The shortest paths perceived by each processor are such that a cycle is formed. In particular, if processor 4 attempts to send a message to the center, that message could circulate in the cycle until eventually the processors perform more iterations and obtain the correct shortest distances.

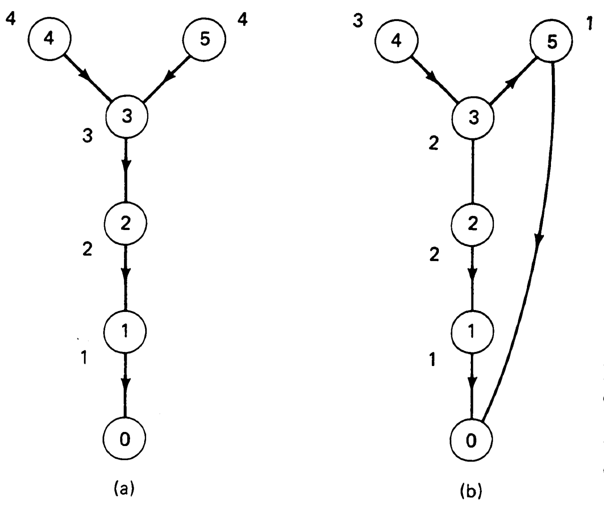


Figure 8.5.3 (a) A network and corresponding shortest distances. (b) Arc $(5, 0)$ is added to the network. Although the old shortest paths can still be used for communication with the center, the new shortest paths are different.

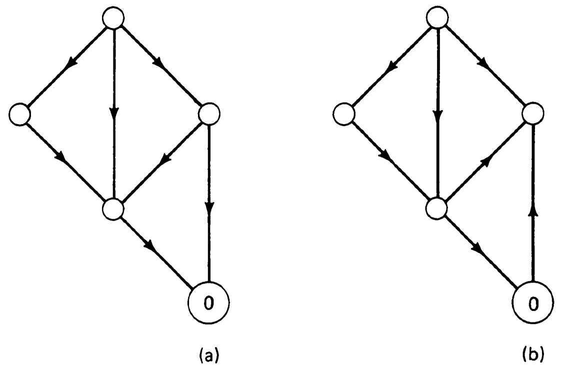


Figure 8.5.4 (a) An oriented DAG. (b) A disoriented DAG.

Property B. The direction of any arc can only be changed in a way that preserves acyclicity.

Property C. In the absence of topological changes and assuming that the undirected graph G is connected, the algorithm eventually terminates and, at termination, processor 0 is the only sink.

Property D. When an arc is added, it is assigned a direction so that the new directed graph is still acyclic.

Clearly, any algorithm with these four properties provides a solution to our problem. Acyclicity is preserved throughout, resulting in loop–free paths, and at termination, we have an oriented DAG.

A simple way for initializing any algorithm is by letting arc (i, j) point toward processor j if and only if $i > j$. This results in a DAG because any path in the directed graph can only go toward processors with smaller identity numbers and cannot, therefore, contain a cycle.

It is assumed in the sequel that the direction of an arc is under the control of the processor to which it points from the time that this processor learns this direction until the time that the direction is changed. In particular, if a processor is in control, it knows the correct direction. Also, our subsequent algorithms are such that a processor that is not in control remains passive. It follows that no actions will ever by taken by processors with incorrect knowledge of the direction of an arc. In order to keep the discussion simple, we will assume that every processor always knows the current directions of its incident arcs (i.e., arc reversal information is instantaneously transmitted and received along an arc whenever its direction is reversed), but the above discussion implies that this assumption is not really necessary.

Our first algorithm is very simple:

Full Reversal Algorithm. The algorithm is initialized by assigning arc directions so that the resulting directed graph is acyclic. Each processor knows the directions of its incident arcs. If all arcs incident to processor i point toward i (i.e., if i is a sink) and if $i \neq 0$, then processor i will, after some finite amount of time, reverse the directions of all these arcs.

Arc reversals by a processor i in this algorithm are assumed to be instantaneous operations and, in particular, all arcs involved are simultaneously reversed and the neighbors of i are informed about this reversal after a finite amount of time. We now verify that the full reversal algorithm has Properties B and C. Concerning Property B, it has already been shown in Section 8.3 that an arc reversal by a sink in a directed acyclic graph preserves acyclicity. For Property C, let $X_i(t)$ be the number of times that processor i has performed a full arc reversal until time t. Let j be a neighbor of i. After an arc reversal by processor i, the arc (i, j) points toward processor j. In order for processor i to perform another arc reversal, it must become a sink and this is possible only if processor j performs an arc reversal in between. This shows that a full arc reversal is performed by processor j between any two consecutive full arc reversals by processor i. Hence $|X_i(t) - X_j(t)| \leq 1$ for all times t and for all pairs i and j, of neighboring processors. Furthermore, processor 0 never performs an arc reversal and we have $X_0(t) = 0$ for all t. Assuming that the underlying graph is connected, we obtain $X_i(t) \leq d$ for all times t, where d is the largest distance from processor 0 to any other processor i. We thus see that in the absence of topological changes, each processor performs at most d full arc reversals and the algorithm terminates after a total of less than nd arc reversals. After termination, no processor $i \neq 0$ can be a sink because if it were a sink, it would eventually perform an arc reversal, thereby contradicting termination. Figure 8.5.5 illustrates the algorithm for a particular example and Exercise 5.1 shows that the worst case estimate $O(nd)$ on the total number of full arc reversals is tight.

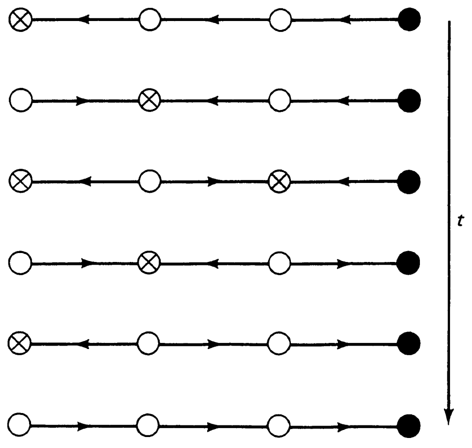


Figure 8.5.5 Illustration of the full reversal algorithm. The black node corresponds to the center and an X denotes a sink ready to perform an arc reversal.

We have not yet mentioned how the full reversal algorithm copes with topological changes, additions of new arcs in particular. It can be seen (Exercise 5.2) that there always exists a direction that can be assigned to a new arc, so that acyclicity is preserved, but this cannot be done, in general, using only the local information available to the processors (Fig. 8.5.6). For this reason, we modify the algorithm so that more information is available to the processors.

Numbered Full Reversal Algorithm. This is the same as our earlier full reversal algorithm except that each processor i maintains an integer number a_i. Initially, these integers satisfy $a_i > a_j$ for every arc (i, j) that points toward j. Whenever processor $i \neq 0$ becomes a sink, it performs an arc reversal and updates a_i by letting

$$a_i := 1 + \max_{j \in A(i)} a_j,$$

where $A(i)$ is the set of neighbors of i. Furthermore, when a neighbor of i becomes aware of an arc reversal by i, it also learns the new value of a_i. Finally, whenever a new arc (i, j) is added to the network, it is oriented to point to processor j if and only if $a_i > a_j$ or $a_i = a_j$ and $i > j$.

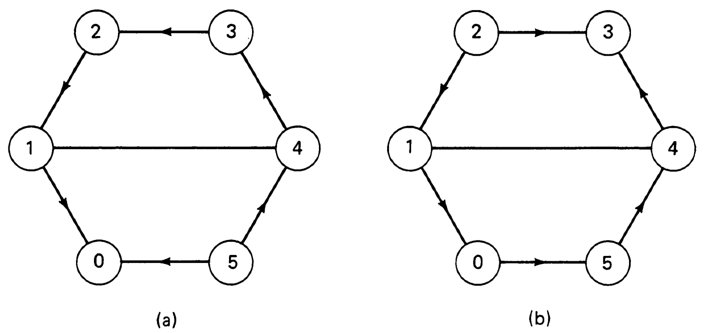


Figure 8.5.6 Difficulties in assigning a direction to a newly created arc using only local information. In (a) and (b), we show two directed acyclic graphs. Arc $(1, 4)$ has just been added to the graph and is to be assigned a direction while preserving acyclicity. In (a), this arc must point toward node 1, whereas in (b), it must point toward node 4. However, in both examples, the local information available to processors 1 and 4 is the same and is therefore insufficient for assigning a direction to arc $(1, 4)$.

The initialization of the numbers a_i is easy. For example, we can let $a_i = i$ for all i, and orient every arc (i, j) to point to the processor with the smaller identity. We call the pair $v_i = (a_i, i)$ the *value* of processor i. We order values lexicographically, that is, we let $(a_i, i) > (a_j, j)$ if $a_i > a_j$ or if $a_i = a_j$ and $i > j$. With this ordering, the set of all possible values is totally ordered and, furthermore, no two processors can ever have the same value. We notice that the algorithm maintains throughout the property that an arc $(i, j) \in A$ is oriented toward processor j if and only if $v_i > v_j$. In particular, this property is preserved when arcs are added to or removed from the network, which guarantees that we have an acyclic graph at all times and Property D holds. Properties B and C remain valid by our earlier discussion. We conclude that this algorithm satisfies all of our requirements.

As the numbered full reversal algorithm continues to be executed and as more topological changes occur (resulting in more and more arc reversals) the numbers a_i will grow unbounded. It may thus be necessary to have the center obtain a global view of the network and reassign new and relatively small numbers to the processors once in a while. Alternatively, a processor can reduce its number a_i on its own, provided that this does not change the direction of any arc.

We continue with a modification of the full reversal algorithm that, in the absence of topological changes, tends to terminate with a substantially smaller number of arc reversals.

Partial Reversal Algorithm. The algorithm is initialized by assigning arc directions so that the resulting directed graph is acyclic. Each processor $i \neq 0$ maintains a list of its incident arcs (i, j) that have been recently reversed by processor j. (Initially this list is empty.) If processor i becomes a sink, then it reverses the directions of all incident arcs that *do not* appear in the list and empties the list. An exception arises if the list contains all arcs incident to i, in which case, all of them are reversed and the list is emptied.

In the absence of topological changes, the partial reversal algorithm tends to involve fewer arc reversals than the full reversal algorithm (see Fig. 8.5.7 for an illustration), even though the best bound we have available is exponential in d. Exercise 5.3 provides such a bound and also proves that the algorithm eventually terminates. The difficult part in the proof of correctness of this algorithm is in verifying that acyclicity is preserved throughout. Furthermore, as discussed earlier, in the context of the full reversal algorithm, locally available information is not sufficient to handle topological changes. For this reason, a set of auxiliary integer variables will be introduced. Each processor i is assumed to maintain a value v_i that is a triple $v_i = (a_i, b_i, i)$. Values are again ordered lexicographically. That is, $v_i > v_j$ if and only if one of the following three occurs: (i) $a_i > a_j$, or (ii) $a_i = a_j$ and $b_i > b_j$, or (iii) $a_i = a_j$, $b_i = b_j$, and $i > j$. The processors' values are to be updated as in the following algorithm.

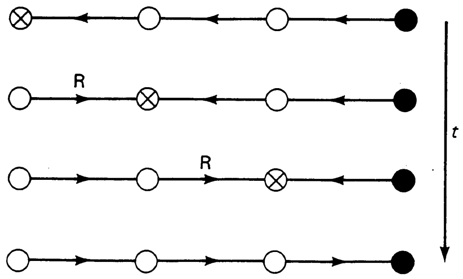


Figure 8.5.7 Illustration of the partial reversal algorithm for the same graph and initialization as in Fig. 8.5.5. An arc labeled R and pointing toward processor i belongs to the list (maintained by processor i) of recently reversed arcs.

Numbered Partial Reversal Algorithm. The algorithm is initialized by assigning arc directions so that the resulting directed graph is acyclic. Each processor i maintains a triple $v_i = (a_i, b_i, i)$; an arc (i, j) is oriented toward j if and only if $v_i > v_j$. If processor $i \neq 0$ becomes a sink [i.e., if $v_i < v_j$ for all $j \in A(i)$], then processor i lets

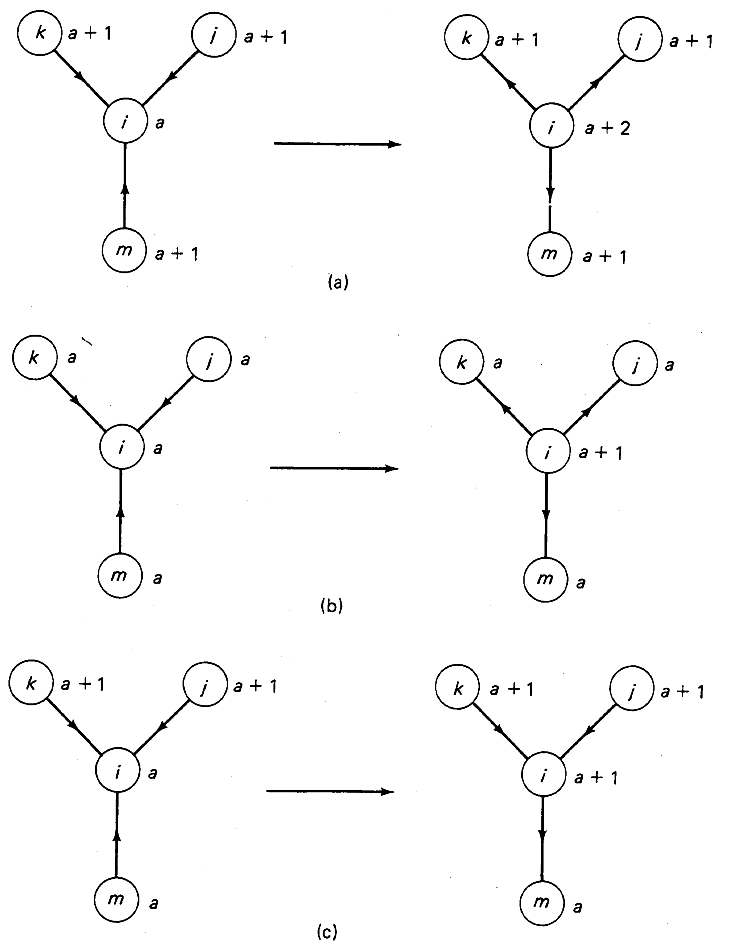


Figure 8.5.8 The relation between the partial reversal algorithm and the numbered partial reversal algorithm, in the absence of topological changes. Suppose that the numbered partial reversal algorithm is initialized with $a_i = b_i = c$ for all $i \neq 0$, where c is some constant. We show that throughout the algorithm we have (i) $|a_i - a_j| \leq 1$ for any two neighboring processors i and j, and (ii) $a_i = a_j + 1$ if and only if processor i has recently reversed the direction of arc (i, j) and processor j has not yet performed a partial arc reversal. Indeed these properties are true at initialization. Suppose that these properties are true before a partial arc reversal by processor i. Parts (a), (b), and (c) consider the three possible cases. All arcs are reversed in cases (a) and (b). In (c), we have right after the partial arc reversal, $a_i = a_j = a_k$, $b_i < b_j$, and $b_i < b_k$. Thus, arcs (j, i) and (k, i) are not reversed. This shows that properties (i) and (ii) remain valid throughout the algorithm. Given that property (ii) holds, we conclude that the set $\{i \mid a_i = a_j + 1\}$ in the numbered partial reversal algorithm is equal to the list of recently reversed arcs maintained by processor j in the partial reversal algorithm, and the two algorithms are identical.

$$a_i := 1 + \min_{j \in A(i)} a_j, \tag{5.1}$$

$$b_i := \min_{j \in A(i)} b_j - 1, \tag{5.2}$$

reverses the directions of its adjacent arcs (i, j) for which the new value of v_i is larger than v_j, and communicates the new value of v_i to its neighbors. If a new arc (i, j) is added to the network, it is oriented toward j if and only if $v_i > v_j$.

This algorithm coincides with our original partial reversal algorithm, provided that it is initialized appropriately and no topological changes occur (Fig. 8.5.8). The algorithm can also be interpreted as a version of the Bellman–Ford shortest path algorithm [cf. Eq. (5.1)]. The role of the coefficients b_i is to provide a tie breaking mechanism under which acyclicity is preserved. The number of partial arc reversals until the algorithm terminates behaves similarly with the Bellman–Ford algorithm: it tends to be small when the coefficients a_i are initialized appropriately (e.g., very large; see Section 4.1), but can also be arbitrarily large under certain circumstances (Fig. 8.5.9).

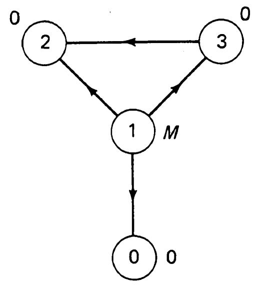


Figure 8.5.9 An example of slow performance of the numbered partial reversal algorithm. Suppose that the algorithm is initialized with $a_0 = 0$, $a_1 = M$, and $a_2 = a_3 = 0$, where M is a large integer. Processor 1 never becomes a sink and the value of a_1 stays constant. The values of a_2 and a_3 increase by steps of size $\Theta(1)$, and it is seen that the algorithm undergoes a total of $\Theta(M)$ partial arc reversals before it terminates.

We now verify that this algorithm has the desired properties. In our arguments, we will assume that processors adjacent to i receive all relevant information instantaneously after an arc reversal by processor i. Similarly, if a topological change involving arc (i, j) occurs, both processors i and j are instantaneously informed. The proof for the case where these instantaneity assumptions fail to hold follows the same lines, but the details are more tedious. Property A holds by definition. The values v_i of successive processors along any positive path in the directed graph are strictly decreasing, which shows that acyclicity is maintained throughout, and, therefore, Properties B and D hold.

To verify Property C, we assume that there are no topological changes and that the graph is connected, and we first prove that the algorithm eventually terminates. Let us denote by $a_i(t)$ the value of a_i at time t. Suppose that the function $a_i(t)$ is unbounded above for some i. From Eq. (5.1), it follows that $a_j(t)$ is also unbounded above for every j that is a neighbor of i. Since the graph is connected, $a_j(t)$ must be unbounded above for all processors. But this is impossible because processor 0 never performs an arc reversal and a_0 stays constant at its initial value. We conclude that each $a_i(t)$ is bounded above. We also notice from Eq. (5.1) that the quantity $\min_i a_i$ cannot decrease when a processor performs an arc reversal. We conclude that the functions $a_i(t)$ are also bounded below. In particular, each function $a_i(t)$ has a smallest limit point to be denoted by c_i.

We denote by NT the set of processors that perform an infinite number of partial arc reversals, and let T be its complement. We assume, in order to derive a contradiction, that NT is nonempty. Let t_0 be some time such that $a_i(\tau) \geq c_i$ for all $\tau \geq t_0$ and all i. Such a time exists because otherwise some function $a_i(t)$ would have a limit point smaller than or equal to $c_i - 1$. Notice that for every $i \in T$, the function $a_i(t)$ eventually becomes and stays equal to c_i. Thus, by choosing t_0 sufficiently large, we have $a_i(t) = c_i$ for all $i \in T$ and $t \geq t_0$. Let $i^* \in NT$ be a processor such that $c_{i^*} \leq c_i$ for all $i \in NT$. We consider two cases:

(i) If processor i^* has a neighbor $j \in T$ with $c_j < c_{i^*}$, then $a_{i^*}(t) \geq c_{i^*} > c_j = a_j(t)$ for all $t \geq t_0$. Thus, processor i^* cannot be a sink at any time after t_0, contradicting our assumption that $i^* \in NT$.

(ii) Now suppose that $c_j \geq c_{i^*}$ for all neighbors $j \in T$ of i^*. Since $c_j \geq c_{i^*}$ for all $j \in NT$ (this follows from the definition of i^*), we obtain $a_j(t) \geq c_j \geq c_{i^*}$ for all neighbors j of i^* and for all times greater than t_0. In particular, at each time after t_0 that processor i^* performs an arc reversal, the new value of a_{i^*} is at least as large as $c_{i^*} + 1$. In between consecutive arc reversals by processor i^*, the value of a_{i^*} remains constant. We conclude that $a_{i^*}(t) \geq c_{i^*} + 1$ subsequent to the first arc reversal by processor i^* that occurs after time t_0. In particular, every limit point of $a_{i^*}(t)$ is larger than c_{i^*}. This contradicts the definition of c_{i^*}.

We have thus contradicted the existence of an element of NT. It follows that the algorithm eventually terminates. At termination, processor 0 is by necessity the only sink, and we conclude that the algorithm has Property C and performs as desired.

We now show that the reversal algorithms do not have the undesirable properties of the shortest path methods that were mentioned in the beginning of this section. Suppose that a reversal algorithm has not yet terminated and that a processor i sends a message to a neighboring processor j, attempting to communicate with the center. If the direction of the arc (i, j) is reversed by processor j before the message is forwarded by j, it is conceivable that the message is returned to processor i. Such a scenario can be repeated, thus demonstrating that a message can circulate in a cycle as long as the reversal algorithm has not terminated. It can be shown, however, that the number of arcs traversed by such a message is bounded by a function of the number of arc reversals performed by the processors. In the absence of further topological changes, the number of arc reversals is bounded, and, therefore, the total number of arcs traversed by a message is also bounded (Exercise 5.4). This is in contrast to the shortest path method, in which a cycle can be formed and a message can circulate an unbounded number of times in that cycle (cf. Fig. 8.5.2).

For the second property, we claim that if a numbered reversal algorithm has terminated, resulting in a particular DAG and a set P of simple positive paths, and if a topological change occurs, then any path in P that is unaffected by the topological change is also unaffected by future iterations of the algorithm. To see this, first suppose that a new arc is added. Since at termination, processor 0 was the only sink and no new sinks can be created by the addition of an arc, we conclude that the new DAG is also oriented and therefore does not change in the future. Now suppose that an arc is removed, but that a path $p = (i_1, i_2, \dots, i_K, 0) \in P$ from a processor i_1 to 0 is unaffected. Since the arc $(i_K, 0)$ has not been removed and processor 0 never reverses its arcs, processor i_K never becomes a sink. Thus, processor i_K does not reverse its incident arcs and, therefore, processor i_{K-1} never becomes a sink. Proceeding backwards, we see that none of the processors on the path p ever becomes a sink and the path p remains unaffected. This is a major difference with our earlier algorithm that was based on the shortest path problem (compare with Fig. 8.5.3).

Information Broadcast in a Failure–Prone Network

We have considered so far the problem of establishing loop–free paths along which the nodes of the network can communicate with the center. These paths can also be used to solve the reverse problem, whereby the center wishes to communicate with the processors. In particular these paths can be used to broadcast some information from the center to every node. We discuss briefly some approaches to this problem.

Suppose that the center has some information that we model as a variable V taking values from some set. The value of this variable changes with time, and the center wishes, roughly speaking, that every other processor is to know, after a while, the current value of this variable. For example, V could represent the status of one of the outgoing communication links from the center; in this case, when the information broadcast algorithm is executed with every node independently playing the role of the center, it is usually called a *topology broadcast algorithm*. Alternatively, V could be a command from the center to start a certain distributed algorithm together with the information needed by each processor to execute this algorithm.

More precisely, we assume that after some initial time, several changes of V and several changes of the network topology occur, but there is some time after which there are no changes of either V or the network topology. We wish to design an algorithm by which all processors, after a finite time from the last change, are to know the correct value of V.

The simplest solution to the broadcast problem, widely used in data communication networks, is the so called *flooding* scheme, whereby the center (which holds the value of V) sends an update message with the most recent value of V to all its neighbors following each change of V. The neighbors send the message to their neighbors except for the one that they heard the message from, etc. To avoid infinite circulation of messages, a sequence number is appended to the value of V. This number is incremented with each new message issued by the center. A processor then accepts a new value of V and propagates it further only if it carries a sequence number that is larger than the one stored in its memory; otherwise it simply discards the corresponding message. To cope with situations where the network becomes disconnected, we require that when a link becomes operational after being down, the end processors exchange their stored values of V together with the corresponding sequence numbers. This flooding scheme works fairly well in practice, and requires less than $2|A|$ messages per new value of V, where

$|A|$ is the number of bidirectional network links. In fact, the scheme does not require the assumption that links preserve the order of transmission of messages; the sequence numbers can be used to recognize the correct order of messages.

One drawback of this flooding scheme is the extra overhead required for the sequence number. Usually, this number is encoded in a binary field of fixed length, which must be large enough to ensure that wraparound will never occur between startup and shutdown of the system. A second drawback is that when a processor crashes, it must remember the last sequence number it used at the time of the crash if there is a possibility that the processor will be repaired before the entire system is shut down.

We now discuss briefly an alternative to the use of sequence numbers. Here each processor i stores in its memory the value V_j^i latest received from each of its neighbors j and also maintains in its memory the value of V_i it thinks is the correct one. Each time V_i changes, its value is transmitted to all neighbors of i along each of the currently operating links incident to i; also, when a link (i, j) becomes operational after being down, the end nodes i and j exchange their current values V_i and V_j, that is, i sends V_i to j, which is stored as V_i^j, and j sends V_j to i, which is stored as V_j^i.

The algorithm description will be completed once we give the rule for changing the value V_i of each processor i. There are a number of possibilities along these lines, but the main idea is as follows.

Suppose we have an algorithm running in our system that maintains a tree of directed paths from all processors to the center (i.e., a tree rooted at the center; see Fig. 8.5.10). What we mean here is that this algorithm constructs such a tree within finite time following the last change in the network topology. Every node i except for the center has a unique successor $s(i)$ in such a tree, and we assume that this successor eventually becomes known to i. The rule for changing V_i then is for node i to make it equal to the value $V_{s(i)}^i$ latest transmitted by the successor $s(i)$ within finite time after either the current successor transmits a new value or the successor itself changes. It is evident, under these assumptions, that each node i will have the correct value V_i within finite time following the last change in V or in the network topology.

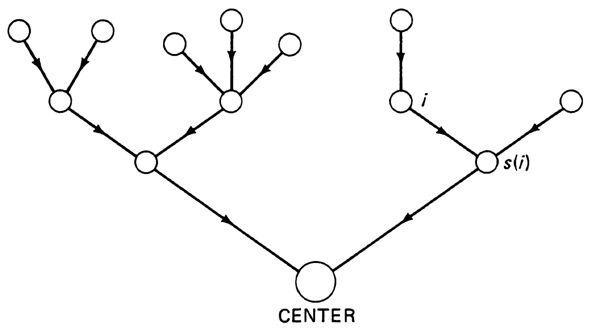


Figure 8.5.10 Information broadcast over a tree rooted at the center that initially holds the value V. Here there is a unique directed path from every processor to the center and a unique successor $s(i)$ of every node i on the tree. The idea of the algorithm to broadcast V is to maintain such a tree (in the presence of topological changes) and to require each node i to (eventually) adopt the latest received message from its successor $s(i)$ (i.e., $V_i = V_{s(i)}^i$). This guarantees that the correct information will eventually be propagated and adopted along the tree even after a finite number of topological changes that may require the restructuring of the tree several times.

It should be clear that there are many candidate algorithms that maintain a tree rooted at the center in the face of topological changes. Two possibilities are the full and partial reversal algorithms of this section, where the successor $s(i)$ of a node i is selected arbitrarily among the neighbors j for which the arc (i, j) is oriented from i to j. Another scheme is given in [Spi85c] and [SpG87], where the idea of information broadcast without sequence numbers was first introduced. A detailed presentation, together with additional discussion of topological change broadcasting is given in Section 5.3 of [BeG87].

EXERCISES

5.1. Consider the graph $G = (N, A)$, where $N = \{0, 1, \ldots, n\}$, and $A = \{(i, i+1) \mid 0 \leq i \leq n-1\}$. Suppose that each arc $(i, i+1)$ is initially oriented to point toward node $i+1$. Calculate the total number of full arc reversals to be performed by the processors before the full reversal algorithm terminates. Check your answer with Fig. 8.5.5 for the case $n = 3$.

5.2. Let $G = (N, A)$ be a directed acyclic graph and suppose that $(i, j) \notin A$ and $(j, i) \notin A$ for some $i, j \in N$. Let $A' = A \cup \{(i, j)\}$ and $A'' = A \cup \{(j, i)\}$. Show that at least one of the graphs (N, A') and (N, A'') is acyclic.

5.3. Consider the partial reversal algorithm on a connected graph with a fixed topology. Let $X_i(t)$ be the number of times, up to time t, that processor i has performed a partial arc reversal.

(a) Show that if $(i, j) \in A$, then between any three consecutive partial arc reversals by processor i, processor j must have performed at least one partial arc reversal.

(b) Show that $X_i(t) \leq 2X_j(t) + 2$ for all $(i, j) \in A$.

(c) Show that the algorithm eventually terminates and bound the total number of partial arc reversals until termination.

5.4. Suppose that the processors are using either the full or the partial reversal algorithm and that some processor sends a message, attempting to communicate with the center, before the algorithm terminates. This message is forwarded by other processors along arcs that are oriented away from the forwarding processor. Assuming that no topological changes occur, show that the total number of arcs traversed by the message, until it reaches the center, is bounded by some function of the number of processors. *Hint:* For the partial reversal algorithm, use the result of Exercise 5.3.

NOTES AND SOURCES

8.1 The idea of detecting termination through the use of acknowledgments is implicit in some distributed shortest path routing and network resynchronization algorithms developed for use in data communication networks (cf. Example 8.3; see [Gal76], [SMG78], [Fin79], and [Seg83]). The termination detection procedure of this section is due to [DiS80]. For related work see [Apt86], [ChM82], [DFV83], [Eri88], [Fra80], [HaZ87], [MiC82], [SSP85], and [Ver87].

8.2 The snapshot algorithm is from [ChL85]. A discussion of local and global clocks, and of the issues related to the comparison of the times at which different events take place at different processors can be found in [Lam78]. See [GaB87] for the use of snapshots in the context of a distributed implementation of simulated annealing.

8.3 A version of the resource allocation problem has been first formulated in [Dij71]. A probabilistic algorithm for this problem was given in [LeR81]. The algorithm presented in this section is from [ChM84] and is closely related to the arc reversal algorithm of Section 8.5. See [Bar86] and [BaG87] for the analysis and the optimization of the concurrency measure M.

8.4 Methods for the simulation of discrete event systems that do not use rollback are surveyed in [Mis86]. See also [ChM81] for a particular such method. Synchronization by rollback has been introduced in [Jef85]. The rate of progress of a simulation when the rollback mechanism is used has been studied in a probabilistic framework in [LMS83] and [MiM84], for the case of two processors. The use of the snapshot algorithm for monitoring the progress of the simulation is discussed in [Sam85b] and [Gaf86]. The relation between rollback and the asynchronous Bellman–Ford algorithm, and its communication complexity implications, are new.

8.5 The arc reversal algorithms of this section are from [GaB81]. The topology broadcast problem is discussed in [Spi85c], [Gaf86], [HuS87a], [HuS87b], and [SpG87]. The idea of maintaining a tree in the face of topological changes along which to broadcast information, is implicit in [Spi85c] and [SpG87], and was articulated by E. M. Gafni (private communication). Flooding is used in many data networks for information broadcast, including the ARPANET. Some difficulties with the ARPANET algorithm are discussed in [Per83] and [BeG87]. For a complexity analysis of flooding, see [Top87].

A

Linear Algebra and Analysis

In this appendix, we collect definitions, notational conventions, and several results on linear algebra and analysis that are used extensively in the book. We omit some proofs that are either elementary or too involved. In any case, it is assumed that the reader has some basic background on the subject. Related and additional material can be found in [Gan59], [HoK71], [LaT85], and [Str76] (linear algebra), and in [Rud76] and [Ash72] (analysis).

We will be considering both real and complex vectors and matrices. Many results are stated for the complex case, but the reader should have no difficulty in converting them to the real case.

Notation

If S is a set and x is an element of S, we write $x \in S$. A set can be specified in the form $S = \{x \mid x \text{ satisfies } P\}$, as the set of all elements satisfying property P. The union of two sets S and T is denoted by $S \cup T$ and their intersection by $S \cap T$. The symbols $\exists$ and $\forall$ have the meanings "there exists" and "for all," respectively.

Let $\Re$ be the set of real numbers and let $\bar{\Re}$ be the set of real numbers together with ∞ and $-\infty$. If $a, b \in \bar{\Re}$ and $a \leq b$, the notation $[a, b]$ stands for the set of all $x \in \bar{\Re}$ such that $a \leq x \leq b$. Similarly, if $a < b$, the notation (a, b) stands for the set of all $x \in \Re$ such that $a < x < b$. Notations such as $[a, b)$ and $(a, b]$ are to be interpreted accordingly.

If f is a function, we use the notation $f : A \mapsto B$ to indicate the fact that f is defined on a set A (its *domain*) and takes values in a set B (its *range*).

Let A be some subset of $\Re$ and let $f : A \mapsto \Re$ and $g : A \mapsto \Re$ be some functions. The notation $f(x) = O\big(g(x)\big)$ [respectively, $f(x) = \Omega\big(g(x)\big)$] means that there exists a positive constant c and some x_0 such that for every $x \in A$ satisfying $x \geq x_0$, we have $|f(x)| \leq cg(x)$ [respectively, $f(x) \geq cg(x)$]. The notation $f(x) = \Theta\big(g(x)\big)$ means that both $f(x) = O\big(g(x)\big)$ and $f(x) = \Omega\big(g(x)\big)$ are true.

Throughout the text, we use $\log x$ to denote the base 2 logarithm of x and $\ln n$ to denote the natural logarithm of x, that is, $x = 2^{\log x} = e^{\ln x}$.

Vectors and Matrices

Let C be the set of complex numbers. For any $c \in C$, let $\bar{c}$ be its complex conjugate, and let $|c|$ be its magnitude. Let $\Re^n$ (respectively, C^n) be the set of n–dimensional real (respectively, complex) vectors. For any $x \in C^n$, we use x_i to indicate its ith *coordinate*, also called its ith *component*. Vectors in C^n will be viewed as column vectors, unless the contrary is explicitly stated. For any $x \in C^n$, we let x' be its transpose, which is an n–dimensional row vector; we also let x^* be the conjugate transpose of x. For any two vectors $x, y \in C^n$, their *inner product* $\langle x, y \rangle$ is defined to be equal to $x^*y = \sum_{i=1}^n \bar{x}_i y_i$. For real vectors $x, y \in \Re^n$, their inner product is equal to $x'y$. Any two vectors $x, y \in C^n$ satisfying $x^*y = 0$ are called *orthogonal.*

For any matrix A, we use A_{ij}, $[A]_{ij}$, or a_{ij} to denote its ijth entry. The *transpose* of A, denoted by A', is defined by $[A']_{ij} = a_{ji}$ and its complex conjugate transpose, denoted by A^*, is defined by $[A^*]_{ij} = \bar{a}_{ji}$. For any two matrices A and B of compatible dimensions, we have $(AB)' = B'A'$ and $(AB)^* = B^*A^*$.

Let A be a square matrix. We say that A is *symmetric* if $A' = A$. We say that A is *diagonal* if $[A]_{ij} = 0$ whenever $i \neq j$. It is *tridiagonal* if $[A]_{ij} = 0$ for all i and j such that $|i - j| > 1$. It is *lower triangular* if $[A]_{ij} = 0$ whenever $i < j$ and *strictly lower triangular* if $[A]_{ij} = 0$ whenever $i \leq j$. It is *upper* (respectively, *strictly upper*) *triangular* if its transpose is lower (respectively, strictly lower) triangular. It is *triangular* if it is upper or lower triangular. We use I to denote the identity matrix. The *determinant* of A is denoted by $\det(A)$.

Positive Vectors and Matrices

If w is a vector in $\Re^n$, the notations $w > 0$ and $w \geq 0$ indicate that all coordinates of w are positive or nonnegative, respectively. Similarly, if A is a real matrix, the notations $A > 0$ and $A \geq 0$ indicate that all entries of A are positive or nonnegative, respectively. For any two vectors w, v, the notation $w > v$ means $w - v > 0$. The notations $w \geq v$, $w < v$, $A < B$, etc. are to be interpreted accordingly.

Given a vector $w \in C^n$, we denote by $|w|$ the vector whose ith coordinate is equal to the magnitude of the ith coordinate of w. Similarly, for any matrix A, we denote by $|A|$ the matrix whose entries are equal to the magnitudes of the corresponding entries of A.

Subspaces and Linear Independence

A subset S of C^n is called a *subspace* of C^n if $ax + by \in S$ for every $x, y \in S$ and every $a, b \in C$. If S is a subspace of C^n and x is some vector in C^n, then the set $x + S = \{z \in C^n \mid z - x \in S\}$ is called a *linear manifold.* Given a finite collection $F = \{x^1, \ldots, x^K\}$ of elements of C^n, the *span* of F is the subspace of C^n defined as the set of all vectors y of the form $y = \sum_{k=1}^{K} a_k x^k$, where each a_k is a complex scalar. The vectors $x^1, \ldots, x^K \in C^n$ are called *linearly independent* if there exists no set of complex coefficients $a_1, \ldots, a_K$ such that $\sum_{k=1}^{K} a_k x^k = 0$, unless $a_k = 0$ for each k. An equivalent definition is that $x^1 \neq 0$ and for every $k > 1$, the vector x^k does not belong to the span of $x^1, \ldots, x^{k-1}$. It can be seen that if the vectors $x^1, \ldots, x^K$ are nonzero and mutually orthogonal, then they are automatically linearly independent. Given a subspace S of C^n, a *basis* for S is a collection of vectors that are linearly independent and whose span is equal to S. Every basis of a given subspace has the same number of vectors and this number is called the *dimension* of S. In particular, the dimension of C^n is equal to n. Another important fact is that every subspace has an *orthogonal basis*, that is, a basis consisting of mutually orthogonal vectors.

Vector Norms

Definition A.1. A *norm* $\|\cdot\|$ on C^n is a mapping that assigns a real number $\|x\|$ to every $x \in C^n$ and that has the following properties:

(a) $\|x\| \geq 0$ for all $x \in C^n$.
(b) $\|cx\| = |c| \cdot \|x\|$ for every $c \in C$ and every $x \in C^n$.
(c) $\|x\| = 0$ if and only if $x = 0$.
(d) $\|x + y\| \leq \|x\| + \|y\|$ for all $x, y \in C^n$.

A norm on $\Re^n$ is defined by replacing C and C^n by $\Re$ and $\Re^n$, respectively, in the above.

The *Euclidean norm* $\|\cdot\|_2$ is defined by

$$\|x\|_2 = (x^* x)^{1/2} = \left(\sum_{i=1}^{n} |x_i|^2\right)^{1/2}, \tag{A.1}$$

and $\Re^n$, equipped with this norm, is called a *Euclidean space.* The Euclidean norm satisfies the *Schwartz inequality*

$$|x^* y| \leq \|x\|_2 \cdot \|y\|_2, \tag{A.2}$$

and the following theorem:

Proposition A.1. (*Pythagorean Theorem*) If x and y are orthogonal then $\|x + y\|_2^2 = \|x\|_2^2 + \|y\|_2^2$.

The *maximum norm* $\|\cdot\|_\infty$ (also called *sup–norm* or *ℓ_∞–norm*) is defined by

$$\|x\|_\infty = \max_i |x_i|. \tag{A.3}$$

For any positive vector w ($w > 0$), we define the *weighted maximum norm* $\|\cdot\|_\infty^w$ by

$$\|x\|_\infty^w = \max_i \left|\frac{x_i}{w_i}\right|. \tag{A.4}$$

The *ℓ_1–norm* $\|\cdot\|_1$ is defined by

$$\|x\|_1 = \sum_{i=1}^n |x_i|. \tag{A.5}$$

Proposition A.2. For any $x \in C^n$, we have:

(a) $\|x\|_\infty \leq \|x\|_2 \leq n^{1/2}\|x\|_\infty$.
(b) $\|x\|_1 \leq n^{1/2}\|x\|_2 \leq n^{1/2}\|x\|_1$.

Proof.

(a) This is a straightforward consequence of Eqs. (A.1) and (A.3).
(b) Let e be the vector with all coordinates equal to 1. Using Eqs. (A.5) and (A.2), we have

$$\|x\|_1 = e'|x| \leq \|e\|_2 \cdot \|x\|_2 = n^{1/2}\|x\|_2.$$

The inequality $\|x\|_2 \leq \|x\|_1$ is an easy consequence of Eqs. (A.1) and (A.5). **Q.E.D.**

Sequences, Limits, and Continuity

A sequence $\{x^k \mid k = 1, 2, \ldots\}$ (or $\{x^k\}$ for short) of complex numbers is said to *converge* to a complex number x if for every $\epsilon > 0$ there exists some K such that $|x^k - x| < \epsilon$ for every $k \geq K$. A real sequence $\{x^k\}$ is said to converge to ∞ (respectively, $-\infty$) if for every A there exists some K such that $x^k \geq A$ (respectively, $x^k \leq A$) for all $k \geq K$. If a sequence converges to some x (possibly infinite), we say that x is the *limit* of x^k; symbolically, $\lim_{k\to\infty} x^k = x$.

A sequence $\{x^k\}$ is said to converge *geometrically* (or *at the rate of a geometric progression*) to x^* if there exist constants $A \geq 0$ and $\alpha \in [0, 1)$ such that $|x^k - x^*| \leq A\alpha^k$ for all k.

A sequence $\{x^k\}$ is called a *Cauchy sequence* if for every $\epsilon > 0$, there exists some K such that $|x^k - x^m| < \epsilon$ for all $k \geq K$ and $m \geq K$.

A real sequence $\{x^k\}$ is said to be *bounded above* (respectively, *below*) if there exists some real number A such that $x^k \leq A$ (respectively, $x^k \geq A$) for all k. A real sequence is said to be *nonincreasing* (respectively, *nondecreasing*) if $x^{k+1} \leq x^k$ (respectively, $x^{k+1} \geq x^k$) for all k. A complex sequence $\{x^k\}$ is called *bounded* if the sequence $\{|x^k|\}$ is bounded above.

Proposition A.3. Every nonincreasing or nondecreasing real sequence converges to a possibly infinite number. If it is also bounded, then it converges to a finite real number.

The *supremum* of a nonempty set $A \subset \Re$, denoted by $\sup A$, is defined as the smallest real number x such that $x \geq y$ for all $y \in A$. If no such real number exists, we say that the supremum of A is infinite. Similarly, the *infimum* of A, denoted by $\inf A$, is defined as the largest real number x such that $x \leq y$ for all $y \in A$, and is equal to $-\infty$ if no such real number exists. Given a sequence $\{x^k\}$ of real numbers, the supremum of the sequence, denoted by $\sup_k x^k$, is defined as $\sup\{x^k \mid k = 1, 2, \ldots\}$. The infimum of a sequence is similarly defined. Given a sequence $\{x^k\}$, let $y^m = \sup\{x^k \mid k \geq m\}$, $z^m = \inf\{x^k \mid k \geq m\}$. The sequences $\{y^m\}$ and $\{z^m\}$ are nonincreasing and nondecreasing, respectively, and therefore have a (possibly infinite) limit (Prop. A.3). The limit of y^m is denoted by $\limsup_{m\to\infty} x^m$ and the limit of z^m is denoted by $\liminf_{m\to\infty} x^m$.

Proposition A.4. Let $\{x^k\}$ be a real sequence.

(a) There holds

$$\inf_k x^k \leq \liminf_{k\to\infty} x^k \leq \limsup_{k\to\infty} x^k \leq \sup_k x^k.$$

(b) The sequence $\{x^k\}$ converges if and only if $\liminf_{k\to\infty} x^k = \limsup_{k\to\infty} x^k$ and, in that case, both of these quantities are equal to the limit of x^k.

A sequence $\{x^k\}$ of vectors in $\mathcal{C}^n$ is said to converge to some $x \in \mathcal{C}^n$ if the ith coordinate of x^k converges to the ith coordinate of x for every i. The notation $\lim_{k\to\infty} x^k = x$ is used again. Convergence is said to occur geometrically if each coordinate of x^k converges geometrically. Finally, a sequence of vectors is called a Cauchy sequence (respectively, bounded) if each coordinate is a Cauchy sequence (respectively, bounded).

Definition A.2. We say that some $x \in \mathcal{C}^n$ is a *limit point* of a sequence $\{x^k\}$ in $\mathcal{C}^n$ if there exists a subsequence of $\{x^k\}$ that converges to x. Let A be a subset of $\mathcal{C}^n$. We say that $x \in \mathcal{C}^n$ is a limit point of A if there exists a sequence $\{x^k\}$, consisting of elements of A, that converges to x.

Proposition A.5.

(a) A bounded sequence of vectors in $\mathcal{C}^n$ converges if and only if it has a unique limit point.

(b) A sequence in $\mathcal{C}^n$ converges if and only if it is a Cauchy sequence.

(c) Every bounded sequence in $\mathcal{C}^n$ has at least one limit point.

(d) If $\{x^k\}$ is a real sequence and $\limsup_{k\to\infty} x^k$ (respectively, $\liminf_{k\to\infty} x^k$) is finite, then it is the largest (respectively, smallest) limit point of the sequence $\{x^k\}$.

Definition A.3. A set $A \subset \mathcal{C}^n$ is called *closed* if it contains all of its limit points. It is called *open* if its complement is closed. It is called *bounded* if there exists some $c \in \Re$ such that the magnitude of any coordinate of any element of A is less than c. A closed and bounded subset of $\mathcal{C}^n$ is called *compact*. Let $\|\cdot\|$ be a vector norm on $\mathcal{C}^n$. If $A \subset \mathcal{C}^n$ and $x \in A$, we say that x is an *interior* point of A if there exists some $\epsilon > 0$ such that $\{y \in \mathcal{C}^n \mid \|x-y\| < \epsilon\} \subset A$.

Proposition A.6.

(a) The union of finitely many closed sets is closed.

(b) The intersection of closed sets is closed.

(c) The union of open sets is open.

(d) The intersection of finitely many open sets is open.

(e) A subset of $\mathcal{C}^n$ is open if and only if all of its elements are interior points.

Let A be a subset of $\mathcal{C}^m$ and let $f : A \mapsto \mathcal{C}^n$ be some function. Let x be a limit point of A. If the sequence $\{f(x^k)\}$ has a common limit z for every sequence $\{x^k\}$ of elements of A such that $\lim_{k\to\infty} x^k = x$, we write $\lim_{y\to x} f(y) = z$. If A is a subset of $\Re$ and x is a limit point of A, the notation $\lim_{y\uparrow x} f(y)$ [respectively, $\lim_{y\downarrow x} f(y)$] will stand for the limit of $f(x^k)$, where $\{x^k\}$ is any sequence of elements of A converging to x and satisfying $x^k \le x$ (respectively, $x^k \ge x$), assuming that the limit exists and is independent of the choice of the sequence $\{x^k\}$.

Definition A.4. Let A be a subset of $\mathcal{C}^m$.

(a) A function $f : A \mapsto \mathcal{C}^n$ is said to be *continuous* at a point $x \in A$ if $\lim_{y\to x} f(y) = f(x)$. It is said to be continuous on A if it is continuous at every point $x \in A$.

(b) A real valued function $f : A \mapsto \Re$ is called *upper semicontinuous* (respectively, *lower semicontinuous*) at a vector $x \in A$ if $f(x) \ge \limsup_{k\to\infty} f(x^k)$ [respectively, $f(x) \le \liminf_{k\to\infty} f(x^k)$] for every sequence $\{x^k\}$ of elements of A converging to x.

(c) Let A be a subset of $\Re$. A function $f : A \mapsto \mathcal{C}^n$ is called *right–continuous* (respectively, *left–continuous*) at a point $x \in A$ if $\lim_{y\downarrow x} f(y) = f(x)$ [respectively, $\lim_{y\uparrow x} f(y) = f(x)$].

It is easily seen that when A is a subset of $\Re$, a nondecreasing and right–continuous (respectively, left–continuous) function $f : A \mapsto \Re$ is upper (respectively, lower) semicontinuous.

Proposition A.7.

(a) The composition of two continuous functions is continuous.
(b) Any vector norm on $\mathcal{C}^n$ is a continuous function.
(c) Let $f : \mathcal{C}^m \mapsto \mathcal{C}^n$ be continuous, and let $A \subset \mathcal{C}^n$ be open (respectively, closed). Then the set $\{x \in \mathcal{C}^m \mid f(x) \in A\}$ is open (respectively, closed).

Proposition A.8. (*Weierstrass' Theorem*) Let A be a nonempty compact subset of $\mathcal{C}^n$. If $f : A \mapsto \Re$ is continuous, then there exist $x, y \in A$ such that $f(x) = \inf_{z \in A} f(z)$ and $f(y) = \sup_{z \in A} f(z)$.

Proof. Let $\{z^k\}$ be a sequence of elements of A such that $\lim_{k\to\infty} f(z^k) = \inf_{z\in A} f(z)$. Since A is bounded, this sequence has at least one limit point x [Prop. A.5(c)]. Since A is closed, x belongs to A. Finally, the continuity of f implies that $f(x) = \lim_{k\to\infty} f(z^k) = \inf_{z\in A} f(z)$. The proof concerning the supremum of f is similar. **Q.E.D.**

Proposition A.9. For any two norms $\|\cdot\|$ and $\|\cdot\|'$ on $\mathcal{C}^n$, there exists some positive constant $c \in \Re$ such that $\|x\| \le c\|x\|'$ for all $x \in \mathcal{C}^n$.

Proof. Let a be the minimum of $\|x\|'$ over the set of all $x \in \mathcal{C}^n$ such that $\|x\| = 1$. The latter set is closed and bounded and, therefore, the minimum is attained at some $\tilde{x}$ (Prop. A.8) that must be nonzero since $\|\tilde{x}\| = 1$. For any $x \in \mathcal{C}^n$, $x \ne 0$, the $\|\cdot\|$ norm of $x/\|x\|$ is equal to 1. Therefore,

$$0 < a = \|\tilde{x}\|' \le \left\| \frac{x}{\|x\|} \right\|' = \frac{\|x\|'}{\|x\|}, \qquad \forall x \ne 0,$$

which proves the desired result with $c = 1/a$. **Q.E.D.**

Proposition A.10. The set of interior points of a set $A \subset \mathcal{C}^n$ does not depend on the choice of norm.

Proposition A.11. Let $\|\cdot\|$ be an arbitrary vector norm. A sequence $\{x^k\}$ converges to x if and only if $\lim_{k\to\infty} \|x^k - x\| = 0$. In particular, x^k converges to x geometrically if and only if $\|x^k - x\|$ converges to zero geometrically.

Proof. By definition, x^k converges to x if and only if $\lim_{k\to\infty} \|x^k - x\|_\infty = 0$ and the first result follows from Prop. A.9. The result concerning geometric convergence follows similarly. **Q.E.D.**

Definition A.5. Let A be a subset of $\mathcal{C}^m$. Let $\{f^k\}$ be a sequence of functions from A into $\mathcal{C}^n$. We say that f^k converges *pointwise* to a function $f : A \mapsto \mathcal{C}^n$ if $\lim_{k\to\infty} f^k(x) = f(x)$ for every $x \in A$. We say that f^k converges to f *uniformly* if for every $\epsilon > 0$, there exists some K such that $\|f^k(x) - f(x)\| < \epsilon$ for all $k \ge K$ and $x \in A$.

(Using Prop. A.9, it is seen that uniform convergence is independent of the choice of the norm $\|\cdot\|$.)

Matrix Norms

A norm $\|\cdot\|$ on the set of $n \times n$ complex matrices is a mapping that assigns to any $n \times n$ matrix A a real number $\|A\|$ and that has the same properties as vector norms do when the matrix is viewed as an element of $\mathcal{C}^{n^2}$.

We are mainly interested in *induced norms*, which are constructed as follows. Given any vector norm $\|\cdot\|$, the corresponding induced matrix norm, also denoted by $\|\cdot\|$, is defined by

$$\|A\| = \max_{\{x \in \mathcal{C}^n \mid \|x\|=1\}} \|Ax\|. \tag{A.6}$$

The set over which the maximization takes place in Eq. (A.6) is closed [Prop. A.7(c)] and bounded; the function being maximized is continuous [Prop. A.7(b)] and therefore the maximum is attained (Prop. A.8). It is easily verified that for any vector norm, Eq. (A.6) defines a bona fide matrix norm having all the required properties. Induced norms have the following additional properties:

Proposition A.12. Let $\|\cdot\|$ be an induced norm on the set of $n \times n$ matrices. Then:

(a) $\|A\| = \max_{x \neq 0} \|Ax\|/\|x\|$.
(b) $\|Ax\| \leq \|A\| \cdot \|x\|$ for all $x \in \mathcal{C}^n$.
(c) $\|AB\| \leq \|A\| \cdot \|B\|$, where A and B are $n \times n$ matrices.

Proof. Part (a) follows because any nonzero vector x can be scaled so that its norm is equal to 1, without changing the value of the ratio $\|Ax\|/\|x\|$. Part (b) then follows immediately. For part (c), we use part (b) twice to obtain $\|ABx\| \leq \|A\| \cdot \|Bx\| \leq \|A\| \cdot \|B\| \cdot \|x\|$. Since this is true for all x, the result follows from Eq. (A.6). **Q.E.D.**

The examples of vector norms given earlier induce certain matrix norms and, by abuse of notation, we use the same symbols to denote them. They have the following properties:

Proposition A.13. Let A be an $n \times n$ matrix. Then:

(a) $\|A\|_\infty^w = \max_i \frac{1}{w_i} \sum_{j=1}^n |a_{ij}| w_j$. (A.7)
(b) $\|A\|_1 = \max_j \sum_{i=1}^n |a_{ij}|$. (A.8)
(c) $\|A\|_1 = \|A'\|_\infty$.
(d) $\|A\|_\infty \leq n^{1/2} \|A\|_2$.
(e) $\|A\|_1 \leq n^{1/2} \|A\|_2$.
(f) $\|A\|_\infty \cdot \|A\|_1 \leq n \|A\|_2^2$.

Proof.

(a) Let x be such that $\|x\|_\infty^w = 1$. We then have $|x_i| \leq w_i$ for all i and it follows that

$$\|Ax\|_\infty^w = \max_i \frac{1}{w_i} \left| \sum_{j=1}^n a_{ij} x_j \right| \leq \max_i \frac{1}{w_i} \sum_{j=1}^n |a_{ij}| w_j.$$

Since this is true for all such x, we conclude that $\|A\|_\infty^w$ is no larger than the right–hand side of Eq. (A.7). Let i be an index for which the maximum in the right–hand side of Eq. (A.7) is attained. Let x be a vector whose jth coordinate x_j satisfies $a_{ij}x_j = |a_{ij}|w_j$ for each j. In particular, $|x_j| = w_j$ for each j and $\|x\|_\infty^w = 1$. Furthermore, the ith coordinate of Ax is equal to $\sum_{j=1}^n |a_{ij}|w_j$. Therefore, $\|Ax\|_\infty^w$ is at least as large as the right–hand side of Eq. (A.7). This implies the same inequality for $\|A\|_\infty^w$ and completes the proof.

(b) The proof is similar. A little algebra shows that $\|A\|_1$ is no larger than the right–hand side of Eq. (A.8). Then consider a vector x with all entries equal to zero, except for the entry corresponding to the maximizing index in Eq. (A.8), which is equal to 1. For this vector, $\|x\|_1 = 1$ and $\|Ax\|_1$ is equal to the right–hand side of Eq. (A.8), which shows that $\|A\|_1$ is no smaller than the right–hand side of Eq. (A.8).

(c) This is immediate from Eqs. (A.7) and (A.8).

(d) Using Prop. A.2, we have

$$\|Ax\|_\infty \leq \|Ax\|_2 \leq \|A\|_2 \cdot \|x\|_2 \leq \|A\|_2 n^{1/2} \|x\|_\infty.$$

By dividing with $\|x\|_\infty$ and taking the maximum over all $x \neq 0$, the result is obtained.

(e) The result follows, similarly with part (d), from the inequalities

$$\|Ax\|_1 \leq n^{1/2}\|Ax\|_2 \leq n^{1/2}\|A\|_2 \cdot \|x\|_2 \leq n^{1/2}\|A\|_2 \cdot \|x\|_1.$$

(f) This is obtained by combining parts (d) and (e). **Q.E.D.**

A more general class of induced matrix norms is defined as follows. Let $\|\cdot\|$ and $\|\cdot\|'$ be vector norms on C^m and C^n, respectively. (The two norms could be different even if $m = n$.) This pair of vector norms induces a matrix norm defined by

$$\|A\| = \max_{\|x\|'=1} \|Ax\|,$$

where A is an $m \times n$ matrix. Parts (a) and (b) of Prop. A.12 remain valid, provided that $\|x\|$ is replaced by $\|x\|'$.

Eigenvalues

Definition A.6. A square matrix A is called *singular* if its determinant is zero. Otherwise it is called *nonsingular* or *invertible.*

Proposition A.14.

(a) Let A be an $n \times n$ matrix. The following are equivalent:
- **(i)** The matrix A is nonsingular.
- **(ii)** The matrix A' is nonsingular.
- **(iii)** For every nonzero $x \in C^n$, we have $Ax \neq 0$.
- **(iv)** For every $y \in C^n$, there exists a unique $x \in C^n$ such that $Ax = y$.
- **(v)** There exists a matrix B such that $AB = I = BA$.
- **(vi)** The columns of A are linearly independent.

(b) Assuming that A is nonsingular, the matrix B of statement (v) (called the *inverse* of A and denoted by A^{-1}) is unique. Furthermore, if A is real, then A^{-1} is also real.

(c) For any two square invertible matrices A and B of the same dimensions, we have $(AB)^{-1} = B^{-1}A^{-1}$.

Definition A.7. The *characteristic polynomial* ϕ of an $n \times n$ matrix A is defined by $\phi(\lambda) = \det(\lambda I - A)$, where I is the identity matrix of the same size as A. The n (possibly repeated) roots of ϕ are called the *eigenvalues* of A. A vector $x \in C^n$ such that $Ax = \lambda x$ is called an *eigenvector* of A associated with λ.

We note that the eigenvalues and eigenvectors of A could be complex even if A is real.

Proposition A.15.

(a) A complex number λ is an eigenvalue of a square matrix A if and only if there exists a nonzero eigenvector associated with λ.

(b) A square matrix A is singular if and only if it has an eigenvalue that is equal to zero.

Definition A.8.

(a) A matrix J of size $m \times m$ is said to be a *Jordan block* if it is of the form

$$J = \begin{bmatrix} \lambda & 1 & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & \lambda \end{bmatrix}.$$

That is, $J_{i,i+1} = 1$ $(1 \le i \le m-1)$, $J_{ii} = \lambda$ $(1 \le i \le m)$, and $J_{ij} = 0$ if $j \neq i$ and $j \neq i+1$.

(b) A square matrix J is said to be a *Jordan matrix* if it has the structure

$$J = \begin{bmatrix} J_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & J_L \end{bmatrix},$$

where each J_ℓ, $\ell = 1, \ldots, L$, is a Jordan block. That is, J is block–diagonal and each diagonal block is a Jordan block.

Proposition A.16. (*Jordan Normal Form*)

(a) Every square matrix A can be represented in the form $A = SJS^{-1}$, where S is a nonsingular matrix and J is a Jordan matrix.

(b) The Jordan matrix J associated with A is unique up to the rearrangement of its blocks.

(c) The diagonal entries of J are the eigenvalues of A repeated according to their multiplicities.

Proposition A.17.

(a) The eigenvalues of a triangular matrix are equal to its diagonal entries.

(b) If S is a nonsingular matrix and $B = SAS^{-1}$, then the eigenvalues of A and B coincide.

(c) The eigenvalues of $cI + A$ are equal to $c + \lambda_1, \ldots, c + \lambda_n$, where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A.

(d) The eigenvalues of A^k are equal to $\lambda_1^k, \ldots, \lambda_n^k$, where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A.

(e) If A is nonsingular, then the eigenvalues of A^{-1} are the reciprocals of the eigenvalues of A.

(f) The eigenvalues of A and A' coincide.

Proof.

(a) If a_{ii} is the ith diagonal entry of a triangular $n \times n$ matrix A, then $\det(\lambda I - A) = \prod_{i=1}^{n}(\lambda - a_{ii})$, which has roots $a_{11}, \ldots, a_{nn}$.

(b) Let $A = S_1 J S_1^{-1}$, where J is a Jordan matrix associated with A (Prop. A.16). Then $B = (SS_1)J(SS_1)^{-1}$. Thus, J is a Jordan matrix associated with B. From Prop. A.16(c), A and B have the same eigenvalues.

(c) If $A = SJS^{-1}$, where J is a Jordan matrix, then $cI + A = S(cI + J)S^{-1}$. It is easy to see that $cI + J$ is a Jordan matrix and its ith diagonal entry is equal to $\lambda_i + c$, where λ_i is the ith diagonal entry of J. The result follows from Prop. A.16(c).

(d) If $A = SJS^{-1}$, where J is a Jordan matrix, then $A^k = SJ^kS^{-1}$. From part (b) of the present proposition, the eigenvalues of A^k coincide with the eigenvalues of J^k. Since J is triangular, J^k is also triangular and its eigenvalues are equal to its diagonal entries; the latter are equal to the kth powers of the diagonal entries of J, that is, the kth powers of the eigenvalues of A.

(e) If $A = SJS^{-1}$, then $A^{-1} = SJ^{-1}S^{-1}$ and, by part (b), the eigenvalues of A^{-1} are equal to the eigenvalues of J^{-1}. It is easy to show that the inverse of a triangular matrix is also triangular and the diagonal entries of the inverse are the reciprocals of the diagonal entries of the original matrix. From part (a), the eigenvalues of J^{-1} are equal to the reciprocals of the diagonal entries of J; they are, therefore, equal to the reciprocals of the eigenvalues of A.

(f) Notice that $A' = (S^{-1})'J'S'$. From part (b), the eigenvalues of A' coincide with the eigenvalues of J'. From part (a), the eigenvalues of J' are its diagonal entries; these coincide with the diagonal entries of J, which are the eigenvalues of A. **Q.E.D.**

Given a polynomial ϕ and a square matrix A, we define $\phi(A)$ by substituting A for the free variable λ of the polynomial.

Proposition A.18. (*Cayley–Hamilton Theorem*) If ϕ is the characteristic polynomial of a square matrix A, then $\phi(A) = 0$.

Definition A.9. The *spectral radius* $\rho(A)$ of a square matrix A is defined as the maximum of the magnitudes of the eigenvalues of A.

It is known that the roots of a polynomial depend continuously on the coefficients of the polynomial. For this reason, the eigenvalues of a square matrix A depend continuously on A, and we obtain the following.

Proposition A.19. $\rho(A)$ is a continuous function of A.

Proposition A.20. For any induced matrix norm $\|\cdot\|$ and any $n \times n$ matrix A we have

$$\lim_{k\to\infty} \|A^k\|^{1/k} = \rho(A) \le \|A\|.$$

Proof. Let λ be an eigenvalue of A such that $|\lambda| = \rho(A)$. Let $x \neq 0$ be an eigenvector of A corresponding to the eigenvalue λ normalized so that $\|x\| = 1$. Then $\|A\| \ge \|Ax\| = \|\lambda x\| = |\lambda| = \rho(A)$, which proves the right–hand side inequality. Furthermore, $\|A^k\| \ge \|A^k x\| = \|\lambda^k x\| = |\lambda|^k = \rho(A)^k$, which shows that $\liminf_{k\to\infty} \|A^k\|^{1/k} \ge \rho(A)$. We will now prove a reverse inequality. Let $A = SJS^{-1}$, where J is a Jordan matrix. Then $A^k = SJ^kS^{-1}$ and

$$\|A^k\| \leq \|S\| \cdot \|J^k\| \cdot \|S^{-1}\|.$$

Thus,

$$\limsup_{k\to\infty} \|A^k\|^{1/k} \leq \limsup_{k\to\infty} \left(\|S\| \cdot \|S^{-1}\|\right)^{1/k} \limsup_{k\to\infty} \|J^k\|^{1/k} = \limsup_{k\to\infty} \|J^k\|^{1/k},$$

where we used the fact that $\lim_{k\to\infty} a^{1/k} = 1$ for any positive number a.

Let $\bar{J}$ be one of the Jordan blocks in J, of size $m \times m$. Let λ be the value of its diagonal entries [thus, $|\lambda| \leq \rho(A)$]. A direct computation shows that for $k \geq m$, the entries of $\bar{J}^k$ are given by

$$[\bar{J}^k]_{ij} = \begin{cases} 0, & 0 < j < i, \\ \binom{k}{j-i}\lambda^{k-(j-i)}, & i \leq j \leq m, \end{cases}$$

where

$$\binom{k}{\ell} = \frac{k!}{\ell!(k-\ell)!}$$

is the binomial coefficient. Using the inequality

$$\binom{k}{j-i} \leq k(k-1)\cdots(k-j+i+1) \leq k(k-1)\cdots(k-m+1) \leq k^m,$$

we see that each one of the entries of $\bar{J}^k$ is bounded in magnitude by $k^m|\lambda|^{k-(j-i)} \leq ck^n\rho(A)^k$, where c is a constant such that $|\lambda|^{-(j-i)} \leq c$ for every nonzero eigenvalue λ and any i and j between 1 and n. The same bound is obtained for all entries of J^k. Let c_{ij} be the norm of a matrix with all entries equal to 0, except for the ijth entry, which is equal to 1. Let $C = \sum_{i=1}^n \sum_{j=1}^n c_{ij}$. From the triangle inequality,

$$\|J^k\| \leq Cck^n\rho(A)^k \tag{A.9}$$

and

$$\limsup_{k\to\infty} \|J^k\|^{1/k} \leq \rho(A) \lim_{k\to\infty} (Cck^n)^{1/k} = \rho(A),$$

which completes the proof. We have used here the fact $\lim_{k\to\infty} k^{1/k} = 1$, which is easily proved by taking logarithms. **Q.E.D.**

As a corollary of Prop. A.20, we obtain:

Proposition A.21. Let A be a square matrix. We have $\lim_{k\to\infty} A^k = 0$ if and only if $\rho(A) < 1$.

Proof. If $\rho(A) \geq 1$, then $\|A^k\| \geq \rho(A^k) = \rho(A)^k \geq 1$ and A^k does not converge to zero. If $\rho(A) < 1$, let $\epsilon > 0$ be such that $\rho(A) + \epsilon < 1$. Using Prop. A.20, we have $\|A^k\|^{1/k} \leq \rho(A) + \epsilon < 1$ for all sufficiently large k. It follows that A^k converges to zero. **Q.E.D.**

Definition A.10. The *trace* of a square matrix A, denoted by tr(A), is defined as the sum of the diagonal entries of A.

Proposition A.22.

(a) The trace of A is equal to the sum of the eigenvalues of A counted according to their multiplicities.

(b) The trace of A^k is equal to the sum of the kth powers of the eigenvalues of A counted according to their multiplicities.

Proof. Consider the characteristic polynomial $\det(\lambda I - A) = c_0 + c_1\lambda + \cdots + c_{n-1}\lambda^{n-1} + \lambda^n$. From the formula for the determinant, it can be seen that the coefficient c_{n-1} is equal to the negative of the sum of the diagonal entries of A. On the other hand, $\det(\lambda I - A) = \prod_{i=1}^n (\lambda - \lambda_i)$, where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A. The coefficient of λ^{n-1} is seen to be equal to $-\sum_{i=1}^n \lambda_i$ and the result of part (a) follows. Part (b) follows from part (a) and Prop. A.17(d). **Q.E.D.**

Properties of Symmetric and Positive Definite Matrices

Proposition A.23. Let A be a real and symmetric $n \times n$ matrix. Then:

(a) The eigenvalues of A are real.

(b) The Jordan matrix associated to A is diagonal.

(c) The matrix A has a set of n mutually orthogonal, real, and nonzero eigenvectors $x^1, \ldots, x^n$, associated to its eigenvalues $\lambda_1, \ldots, \lambda_n$.

(d) Suppose that the eigenvectors in part (c) have been normalized so that $\|x^k\|_2 = 1$ for each k. Then

$$A = \sum_{k=1}^n \lambda_k x^k (x^k)'.$$

Proof.

(a) Let λ be an eigenvalue of A and let x be an associated nonzero eigenvector. Since A is real and symmetric, we have $(x^*Ax)^* = x^*A^*x = x^*Ax$. Therefore, the expression $x^*Ax = x^*\lambda x = \lambda\|x\|_2^2$ is real, which shows that λ is real.

(b) We represent A in the form $A = SJS^{-1}$, where S and J are as in Prop. A.16. Suppose, to derive a contradiction, that J is not diagonal. Consider the smallest i such that $[J]_{i,i+1} = 1$. Let $\lambda = [J]_{ii}$ and let x and y be the ith and $(i+1)$st column, respectively, of the matrix S. By reading the ith and $(i+1)$st column of the equation $AS = SJ$, we obtain $Ax = \lambda x$ and $Ay = x + \lambda y$. The first equation yields $y^*Ax = \lambda y^*x$. The second yields $x^*Ay = x^*x + \lambda x^*y$. We combine these two equalities and the assumption that A is real and symmetric to obtain

$$\lambda y^*x = y^*Ax = (x^*A^*y)^* = (x^*Ay)^* = (x^*x + \lambda x^*y)^* = x^*x + \lambda y^*x,$$

from which we conclude that $x^*x = 0$. Therefore, $x = 0$, which means that a column of S is zero. Using the equivalence of conditions (i) and (vi) in Prop. A.14(a), we see that S is singular, which is a contradiction.

(c) Suppose that λ is an eigenvalue of A and that it has multiplicity m as a root of the characteristic polynomial of A. Then m of the diagonal entries of the Jordan matrix J are equal to λ. Let $x^1, \ldots, x^m$ be the columns of the matrix S corresponding to these diagonal entries. Since J is diagonal and $AS = SJ$, we see that each x^k, $k = 1, \ldots, m$, is an eigenvector of A associated to λ. Furthermore, these eigenvectors are linearly independent because otherwise the matrix S would not be invertible.

Let Q be the span of the vectors $x^1, \ldots, x^m$; its dimension is m because these vectors are linearly independent. For $k = 1, \ldots, m$, let v^k, w^k, be the real and imaginary parts, respectively, of the vector x^k. Since A is a real matrix and since its eigenvalues are real, the equation $Ax^k = \lambda x^k$ implies that $Av^k = \lambda v^k$ and $Aw^k = \lambda w^k$. The span in $\mathcal{C}^n$ of the set of vectors $U = \{v^1, w^1, \ldots, v^m, w^m\}$ contains Q and therefore has dimension at least equal to m. Thus, the set U contains m linearly independent vectors. Furthermore, these vectors are real and they therefore span an m–dimensional subspace H of $\Re^n$. Let $z^1, \ldots, z^m$ be an orthogonal basis for the subspace H. We have already shown that each element of U is an eigenvector of A, with eigenvalue λ. The vectors $z^1, \ldots, z^m$ are, by construction, linear combinations of the elements of U and are themselves eigenvectors.

We have shown so far that to every eigenvalue of multiplicity m, we can associate m real, nonzero, and mutually orthogonal eigenvectors. The proof is completed by observing that eigenvectors associated to different eigenvalues are also orthogonal. Indeed, suppose that $Ax = \lambda_1 x$, $Ay = \lambda_2 y$, and $\lambda_1 \neq \lambda_2$. Then

$$\lambda_1 y^*x = y^*Ax = \left(x^*Ay\right)^* = \lambda_2(x^*y)^* = \lambda_2 y^*x,$$

which shows that $y^*x = 0$.

(d) Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of A and let $x^1, \ldots, x^n$ be associated real mutually orthogonal eigenvectors, normalized so that $\|x^k\|_2 = 1$ for each k. These eigenvectors are linearly independent and they therefore span $\Re^n$. Thus, any vector $y \in \Re^n$ can be expressed in the form $y = \sum_{k=1}^{n} c_k x^k$, where $c_1, \ldots, c_n$ are suitable real coefficients. Using the orthogonality of the eigenvectors, we obtain $c_k = y'x^k$. We now notice that

$$\sum_{k=1}^{n} \lambda_k x^k (x^k)' y = \sum_{k=1}^{n} \lambda_k c_k x^k = \sum_{k=1}^{n} c_k A x^k = Ay.$$

Since this equality is true for every $y \in \Re^n$, the proof of part (d) is complete. **Q.E.D.**

Proposition A.24. If A is a real and symmetric $n \times n$ matrix, then:

(a) $\|A\|_2 = \rho(A)$.

(b) $\|A\|_2 = \max_{\{x \in \Re^n \mid \|x\|_2 = 1\}} |x'Ax|$.

Proof.

(a) We already know that $\|A\|_2 \geq \rho(A)$ (Prop. A.20) and we need to show the reverse inequality. Let $x^1, \ldots, x^n$ and $\lambda_1, \ldots, \lambda_n$ be mutually orthogonal nonzero eigenvectors of A and the corresponding eigenvalues, respectively. We express an arbitrary vector $x \in C^n$ in the form $x = \sum_{i=1}^{n} c_i x^i$, where each c_i is a suitable complex scalar. Using the orthogonality of the vectors x^i and Prop. A.1, we obtain $\|x\|_2^2 = \sum_{i=1}^{n} |c_i|^2 \cdot \|x^i\|_2^2$. Using Prop. A.1 again, we obtain

$$\|Ax\|_2^2 = \left\| \sum_{i=1}^{n} \lambda_i c_i x^i \right\|_2^2 = \sum_{i=1}^{n} |\lambda_i|^2 \cdot |c_i|^2 \cdot \|x^i\|_2^2 \leq \rho^2(A) \|x\|_2^2.$$

Since this is true for every x, we obtain $\|A\|_2 \leq \rho(A)$ and the desired result follows.

(b) Let λ be an eigenvalue of A such that $|\lambda| = \rho(A) = \|A\|_2$ and let $y \neq 0$ be a corresponding real eigenvector normalized so that $\|y\|_2 = 1$. Then

$$\max_{\{x \in \Re^n \mid \|x\|_2 = 1\}} |x'Ax| \geq |y'Ay| = |\lambda| \cdot \|y\|_2^2 = \|A\|_2.$$

For the reverse inequality, let $x \in \Re^n$, with $\|x\|_2 = 1$, be arbitrary and decompose it as $x = \sum_{i=1}^{n} c_i x^i$, as in the proof of part (a). Using the orthogonality of the eigenvectors, we obtain

$$|x'Ax| = \sum_{i=1}^{n} |c_i|^2 \cdot |\lambda_i| \cdot \|x^i\|_2^2 \leq \rho(A) \|x\|_2^2 = \rho(A) = \|A\|_2.$$

Q.E.D.

Proposition A.25. Let A be a real square matrix. Then:

(a) $\|A\|_2 = \max_{\|y\|_2 = \|x\|_2 = 1} |y'Ax|$, where the maximization is carried out in $\Re^n$.

(b) $\|A\|_2 = \|A'\|_2$.

(c) If A is symmetric then $\|A^k\|_2 = \|A\|_2^k$ for any positive integer k.

(d) $\|A\|_2^2 = \|A'A\|_2 = \|AA'\|_2$.

(e) $\|A\|_2^2 \leq \|A\|_\infty \cdot \|A\|_1$.

(f) If A is symmetric and nonsingular, then $\|A^{-1}\|_2$ is equal to the reciprocal of the smallest of the absolute values of the eigenvalues of A.

Proof.

(a) It is easily seen that the Euclidean vector norm satisfies $\|x\|_2 = \max_{\|y\|_2=1} |y'x|$. It follows that $\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \max_{\|y\|_2=\|x\|_2=1} |y'Ax|$.

(b) This is an immediate consequence of part (a) and the fact $y'Ax = x'A'y$.

(c) If A is symmetric then A^k is symmetric. Using Prop. A.24(a), we have $\|A^k\|_2 = \rho(A^k)$. Using Prop. A.17(d), we obtain $\rho(A^k) = \rho(A)^k$, which is equal to $\|A\|_2^k$ by Prop. A.24(a).

(d) For any vector x such that $\|x\|_2 = 1$, we have, using the Schwartz inequality (A.2),

$$\|Ax\|_2^2 = x^* A'Ax \leq \|x\|_2 \cdot \|A'Ax\|_2 \leq \|x\|_2 \cdot \|A'A\|_2 \cdot \|x\|_2 = \|A'A\|_2.$$

Thus, $\|A\|_2^2 \leq \|A'A\|_2$. On the other hand, $\|A'A\|_2 \leq \|A'\|_2 \cdot \|A\|_2 = \|A\|_2^2$. Therefore, $\|A\|_2^2 = \|A'A\|_2$. The second equality is obtained by replacing A by A' and using the result of part (b).

(e) Using part (d) and Prop. A.24(a), we have $\|A\|_2^2 = \|AA'\|_2 = \rho(AA')$. We now recall Prop. A.20, which shows that $\rho(AA') \leq \|AA'\|_\infty$. Finally, we use Prop. A.13(c) to obtain $\|AA'\|_\infty \leq \|A\|_\infty \cdot \|A'\|_\infty = \|A\|_\infty \cdot \|A\|_1$, which completes the proof.

(f) This follows by combining Prop. A.17(e) with Prop. A.24(a). **Q.E.D.**

Definition A.11. A square matrix A of dimensions $n \times n$ is called *positive definite* if A is real and $x'Ax > 0$ for all $x \in \Re^n$, $x \neq 0$. It is called *nonnegative definite* if it is real and $x'Ax \geq 0$ for all $x \in \Re^n$.

Proposition A.26.

(a) For any real matrix A, the matrix $A'A$ is symmetric and nonnegative definite. It is positive definite if and only if A is nonsingular.

(b) A square symmetric real matrix is nonnegative definite (respectively, positive definite) if and only if all of its eigenvalues are nonnegative (respectively, positive).

(c) The inverse of a symmetric positive definite matrix is symmetric and positive definite.

Proof.

(a) Symmetry is obvious. For any vector $x \in \Re^n$, we have $x'A'Ax = \|Ax\|_2^2 \geq 0$, which establishes nonnegative definiteness. Positive definiteness is obtained if and only if the inequality is strict for every $x \neq 0$, which is the case if and only if $Ax \neq 0$ for every $x \neq 0$. This is equivalent to A being nonsingular.

(b) Let λ, $x \neq 0$, be an eigenvalue and a corresponding real eigenvector of a symmetric nonnegative definite matrix A. Then $0 \leq x'Ax = \lambda x'x = \lambda \|x\|_2^2$, which proves that $\lambda \geq 0$. For the converse result, let x be an arbitrary real vector. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of A, assumed to be nonnegative, and let $x^1, \ldots, x^n$ be a corresponding set of nonzero, real, and orthogonal eigenvectors. Let us express x in the form $x = \sum_{i=1}^n c_i x^i$. Then $x'Ax = (\sum_{i=1}^n c_i x^i)'(\sum_{i=1}^n c_i \lambda_i x^i)$. From the orthogonality of the eigenvectors, the latter expression is equal to $\sum_{i=1}^n c_i^2 \lambda_i \|x^i\|_2^2 \geq 0$, which proves that A is nonnegative definite. The proof for the case of positive definite matrices is similar.

(c) Let A be symmetric positive definite. We have $AA^{-1} = I$, which implies $I = (A^{-1})'A' = (A^{-1})'A$, which shows that $(A^{-1})' = A^{-1}$ and A^{-1} is symmetric. Since A is positive definite, we have for every real nonzero vector x, $x'A^{-1}x = x'A^{-1}AA^{-1}x = (A^{-1}x)'A(A^{-1}x) > 0$, which shows that A^{-1} is positive definite. **Q.E.D.**

Proposition A.27. Let A be a square symmetric nonnegative definite matrix.

(a) There exists a symmetric matrix $A^{1/2}$ with the property $A^{1/2}A^{1/2} = A$, called a *symmetric square root* of A.

(b) The matrix $A^{1/2}$ is invertible if and only if A is invertible; its inverse is denoted by $A^{-1/2}$.

(c) There holds $A^{-1/2}A^{-1/2} = A^{-1}$.

(d) There holds $AA^{1/2} = A^{1/2}A$.

Proof.

(a) Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of A and let $x^1, \ldots, x^n$ be corresponding nonzero, real, and orthogonal eigenvectors normalized so that $\|x^k\|_2 = 1$ for each k. We let

$$A^{1/2} = \sum_{k=1}^n \lambda_k^{1/2} x^k (x^k)',$$

where $\lambda_k^{1/2}$ is the nonnegative square root of λ_k. We then have

$$A^{1/2}A^{1/2} = \sum_{i=1}^n \sum_{k=1}^n \lambda_i^{1/2} \lambda_k^{1/2} x^i (x^i)' x^k (x^k)' = \sum_{\{(i,k) \mid i=k\}} \lambda_i^{1/2} \lambda_k^{1/2} x^i (x^k)'$$

$$= \sum_{k=1}^n \lambda_k x^k (x^k)' = A.$$

Here the second equality follows from the orthogonality of distinct eigenvectors; the last equality follows from Prop. A.23(d). We now notice that each one of the matrices $x^k(x^k)'$ is symmetric and it follows that $A^{1/2}$ is also symmetric.

(b) This follows from the fact that the eigenvalues of A are the squares of the eigenvalues of $A^{1/2}$ [Prop. A.17(d)].

(c) We have $(A^{-1/2}A^{-1/2})A = A^{-1/2}(A^{-1/2}A^{1/2})A^{1/2} = A^{-1/2}IA^{1/2} = I$.

(d) We have $AA^{1/2} = A^{1/2}A^{1/2}A^{1/2} = AA^{1/2}$. **Q.E.D.**

A symmetric square root of A is not unique. For example, let $A^{1/2}$ be as in the proof of Prop. A.27(a) and notice that the matrix $-A^{1/2}$ also has the property $(-A^{1/2})(-A^{1/2}) = A$. However, if A is positive definite, it can be shown that the matrix $A^{1/2}$ we have constructed is the only symmetric and positive definite square root of A.

Proposition A.28. Let G be a symmetric positive definite $n \times n$ matrix. For $x \in \Re^n$, let $\|x\|_G = (x'Gx)^{1/2}$. Let $G^{1/2}$ be a symmetric square root of G.

(a) There holds $\|x\|_G = \|G^{1/2}x\|_2$ for all $x \in \Re^n$.

(b) $\|\cdot\|_G$ is a vector norm.

(c) There exist positive constants K_1 and K_2, such that $K_1x'G^{-1}x \leq x'Gx \leq K_2x'G^{-1}x$ for all $x \in \Re^n$.

(d) There exists some $\alpha > 0$ such that $x'Gx \geq \alpha\|x\|_2^2$ for all $x \in \Re^n$.

(e) (*Schwartz inequality*) There holds $x'Gy \leq \|x\|_G \cdot \|y\|_G$ for all $x, y \in \Re^n$.

Proof. Part (a) is trivial and part (b) follows easily from part (a) and the fact that $\|\cdot\|_2$ is a norm. For part (c), notice that $(x'G^{-1}x)^{1/2}$ is also a vector norm, because G^{-1} is symmetric and positive definite [Prop. A.26(c)], and the result follows from Prop. A.9. Part (d) follows similarly from Prop. A.9. Part (e) follows from the Schwartz inequality applied to the vectors $G^{1/2}x$ and $G^{1/2}y$. **Q.E.D.**

Proposition A.29. Let G and H be symmetric positive definite matrices of dimensions $m \times m$ and $n \times n$, respectively. Consider the norm on $m \times n$ matrices defined by $\|A\| = \max_{\|x\|_H=1} \|Ax\|_G$, where $\|\cdot\|_G$ and $\|\cdot\|_H$ are defined as in Prop. A.28. Then $\|A\| = \|G^{1/2}AH^{-1/2}\|_2$.

Proof. Since $\|x\|_H = \|H^{1/2}x\|_2$, we obtain

$$\|A\| = \max_{\{x \mid \|H^{1/2}x\|_2=1\}} \|G^{1/2}Ax\|_2 = \max_{\{y \mid \|y\|_2=1\}} \|G^{1/2}AH^{-1/2}y\|_2 = \|G^{1/2}AH^{-1/2}\|_2.$$

Q.E.D.

Derivatives

Let $f : \Re^n \mapsto \Re$ be some function, fix some $x \in \Re^n$, and consider the expression

$$\lim_{\alpha \to 0} \frac{f(x + \alpha e^i) - f(x)}{\alpha},$$

where e^i is the ith unit vector. If the above limit exists, it is called a *partial derivative* of f at the point x and is denoted by $(\partial f/\partial x_i)(x)$. If all of these partial derivatives exist, we let $\nabla f(x)$, called the *gradient* of f, be the column vector whose ith coordinate, denoted by $\nabla_i f(x)$, is equal to the partial derivative $(\partial f/\partial x_i)(x)$. For any $y \in \Re^n$, we define $f'(x;y)$, the one-sided *directional derivative* of f in the direction y, to be equal to

$$\lim_{\alpha \downarrow 0} \frac{f(x+\alpha y) - f(x)}{\alpha},$$

provided that the limit exists. [Notice that if $f'(x;e^i) = -f'(x;-e^i)$, then $f'(x;e^i) = (\partial f/\partial x_i)(x)$.] If all directional derivatives of f at a point x exist and $f'(x;y)$ is a linear function of y, we say that f is *differentiable* at x. It is seen that f is differentiable at x if and only if the gradient $\nabla f(x)$ exists and satisfies $y'\nabla f(x) = f'(x;y)$ for every $y \in \Re^n$. The function f is called differentiable if it is differentiable at every $x \in \Re^n$. In particular, f is differentiable if $\nabla f(x)$ exists for every x and is a continuous function of x, in which case f is said to be *continuously differentiable.* It is seen that a continuously differentiable function is always continuous. Finally, continuously differentiable functions have the property

$$\lim_{y \to 0} \frac{f(x+y) - f(x) - y'\nabla f(x)}{\|y\|} = 0, \qquad \forall x,$$

where $\|\cdot\|$ is an arbitrary vector norm.

Notice that the definitions concerning differentiability of f at a point x only involve the values of f in a neighborhood of x, that is, in an open set containing x. Thus, f does not have to be defined on all of $\Re^n$, as long as it is defined in a neighborhood of the point at which the derivative is computed.

If $f : \Re^n \mapsto \Re^m$ is a vector valued function, it is called differentiable (respectively, continuously differentiable) if each component f_i of f is differentiable (respectively, continuously differentiable), and we let $\nabla f(x)$ be the matrix of dimensions $n \times m$ whose ith column is the gradient $\nabla f_i(x)$ of f_i. Thus,

$$\nabla f(x) = \Big[\nabla f_1(x) \cdots \nabla f_m(x)\Big].$$

The transpose of ∇f is called the *Jacobian* of f and is a matrix whose ijth entry is equal to the partial derivative $\partial f_i/\partial x_j$.

Now suppose that each one of the partial derivatives of a function $f : \Re^n \mapsto \Re$ is a continuously differentiable function of x. We use the notation $(\partial^2 f/\partial x_i \partial x_j)(x)$ to indicate the ith partial derivative of $\partial f/\partial x_j$ at a point $x \in \Re^n$. We define $\nabla^2 f(x)$, called the *Hessian* of f, as the matrix whose ijth entry, denoted by $\nabla^2_{ij} f(x)$, is equal to $(\partial^2 f/\partial x_i \partial x_j)(x)$. We have $(\partial^2 f/\partial x_i \partial x_j)(x) = (\partial^2 f/\partial x_j \partial x_i)(x)$ for every x, which implies that $\nabla^2 f(x)$ is symmetric.

Let $f : \Re^k \mapsto \Re^m$ and $g : \Re^m \mapsto \Re^n$ be continuously differentiable functions and let $h = g \circ f$ be their composition, that is, $h(x) = g\big(f(x)\big)$. Then, the *chain rule* for differentiation states that

$$\nabla h(x) = \nabla f(x) \nabla g\big(f(x)\big), \qquad \forall x \in \Re^k.$$

Proposition A.30. (*Mean Value Theorem*) If $f : \Re \mapsto \Re$ is continuously differentiable, then for every $x, y \in \Re$, there exists some $z \in [x, y]$ such that

$$f(y) - f(x) = f'(z)(y - x),$$

where f' is the derivative of f.

Proposition A.31. (*Second Order Taylor Series*) Let $f : \Re^n \mapsto \Re$ be twice continuously differentiable.

(a) For any $x \in \Re^n$, there exists a function $h : \Re^n \mapsto \Re$ satisfying

$$\lim_{\|y\| \to 0} \frac{h(y)}{\|y\|^2} = 0$$

and such that

$$f(x + y) = f(x) + y'\nabla f(x) + \tfrac{1}{2} y' \nabla^2 f(x) y + h(y), \qquad \forall y \in \Re^n. \tag{A.10}$$

(b) For any $x, y \in \Re^n$, there exists some $\alpha \in [0, 1]$ such that

$$f(x + y) = f(x) + y'\nabla f(x) + \tfrac{1}{2} y' \nabla^2 f(x + \alpha y) y.$$

Proposition A.32. (*Descent Lemma*) If $f : \Re^n \mapsto \Re$ is continuously differentiable and has the property $\|\nabla f(x) - \nabla f(y)\|_2 \le K \|x - y\|_2$ for every $x, y \in \Re^n$, then

$$f(x + y) \le f(x) + y'\nabla f(x) + \frac{K}{2} \|y\|_2^2.$$

Proof. Let t be a scalar parameter and let $g(t) = f(x + ty)$. The chain rule yields $(dg/dt)(t) = y'\nabla f(x + ty)$. Now

$$\begin{aligned}
f(x+y) - f(x) = g(1) - g(0) &= \int_0^1 \frac{dg}{dt}(t)\,dt = \int_0^1 y'\nabla f(x+ty)\,dt \\
&\le \int_0^1 y'\nabla f(x)\,dt + \left| \int_0^1 y'\big(\nabla f(x+ty) - \nabla f(x)\big)\,dt \right| \\
&\le \int_0^1 y'\nabla f(x)\,dt + \int_0^1 \|y\|_2 \cdot \|\nabla f(x+ty) - \nabla f(x)\|_2 dt \\
&\le y'\nabla f(x) + \|y\|_2 \int_0^1 Kt\|y\|_2\,dt \\
&= y'\nabla f(x) + \tfrac{1}{2}K\|y\|_2^2.
\end{aligned}$$

Q.E.D.

Proposition A.33. Let $f : \Re \mapsto \Re$ be twice continuously differentiable and fix some $x \in \Re$. Then

$$\frac{d^2 f}{dx^2}(x) = \frac{f(x+\Delta) + f(x-\Delta) - 2f(x)}{\Delta^2} + h(\Delta),$$

where h is some function satisfying $\lim_{\Delta \to 0} h(\Delta) = 0$.

Proof. We apply Prop. A.31(a) twice: once with $y = \Delta$ and once with $y = -\Delta$. By adding the two equalities obtained, the desired result follows. **Q.E.D.**

Definition A.12. Let $X \subset \Re^n$ and let $f : X \mapsto \Re$ be a given function. A vector $x \in X$ is called a *local minimum* of f (over the set X) if there exists some $\epsilon > 0$ such that $f(y) \ge f(x)$ for every $y \in X$ satisfying $\|x - y\| \le \epsilon$, where $\|\cdot\|$ is some vector norm. A vector $x \in X$ is called a *global minimum* of f (over the set X) if $f(y) \ge f(x)$ for every $y \in X$. A local or global maximum is defined similarly.

Proposition A.34. Let $f : \Re^n \mapsto \Re$ be a continuously differentiable function. If some $x \in \Re^n$ is a local minimum of f over $\Re^n$, then $\nabla f(x) = 0$.

Proof. Fix some $y \in \Re^n$. Then

$$y'\nabla f(x) = \lim_{t \downarrow 0} \frac{f(x+ty) - f(x)}{t} \ge 0,$$

where the last inequality follows from the assumption that x is a local minimum. Since y is arbitrary, the same inequality holds with y replaced by $-y$. Therefore, $y'\nabla f(x) = 0$ for all $y \in \Re^n$, which shows that $\nabla f(x) = 0$. **Q.E.D.**

Convexity

Definition A.13. Let C be a subset of $\Re^n$. We say that C is *convex* if

$$\alpha x + (1-\alpha)y \in C, \qquad \forall x, y \in C,\ \forall \alpha \in [0,1].$$

Let C be a convex subset of $\Re^n$. A function $f : C \mapsto \Re$ is called *convex* if

$$f\big(\alpha x + (1-\alpha)y\big) \le \alpha f(x) + (1-\alpha) f(y), \qquad \forall x, y \in C,\ \forall \alpha \in [0,1]. \qquad \text{(A.12)}$$

The function f is called *concave* if $-f$ is convex. The function f is *strictly convex* if for every $x, y \in C$, $x \ne y$, we have

$$f\big(\alpha x + (1-\alpha)y\big) < \alpha f(x) + (1-\alpha) f(y), \qquad \forall \alpha \in (0,1).$$

We occasionally deal with convex functions that can take the value of infinity. An extended real valued function $f : C \mapsto \Re \cup \{\infty\}$ is also called convex if condition (A.12) holds.

Proposition A.35. Let $C \subset \Re^n$ be a convex set.

(a) If $f : C \mapsto \Re$ is convex, $x^1, \ldots, x^m \in C$, $a_1, \ldots, a_m \ge 0$, and $\sum_{i=1}^m a_i = 1$, then

$$f\left(\sum_{i=1}^m a_i x^i\right) \le \sum_{i=1}^m a_i f(x^i).$$

(b) A linear function is convex.

(c) The weighted sum of convex functions, with positive weights, is convex.

(d) If I is an index set and $f_i : C \mapsto \Re$ is convex for each $i \in I$, then the extended real valued function $h : C \mapsto \Re \cup \{\infty\}$ defined by $h(x) = \sup_{i \in I} f_i(x)$ is also convex.

(e) Any vector norm is convex.

(f) If x is a local minimum of a convex function $f : C \mapsto \Re$, then it is also a global minimum.

(g) If $f : C \mapsto \Re$ is strictly convex, then there exists at most one global minimum of f.

Proof. Part (a) is obtained by repeated application of inequality (A.12). Parts (b) and (c) are immediate consequences of the definition of convexity.

For part (d), let us fix some $x, y \in C$, $\alpha \in [0,1]$, and let $z = \alpha x + (1-\alpha)y$. For every $i \in I$, we have

$$f_i(z) \le \alpha f_i(x) + (1-\alpha) f_i(y) \le \alpha h(x) + (1-\alpha) h(y).$$

Taking the supremum over all $i \in I$, we conclude that $h(z) \leq \alpha h(x) + (1-\alpha)h(y)$, and h is convex.

Let $\|\cdot\|$ be a vector norm. For any $x, y \in \Re^n$ and any $\alpha \in [0,1]$, we have

$$\|\alpha x + (1-\alpha)y\| \leq \|\alpha x\| + \|(1-\alpha)y\| = \alpha\|x\| + (1-\alpha)\|y\|,$$

which proves part (e).

Suppose that x is a local minimum of f but not a global minimum. Then there exists some $y \neq x$ such that $f(y) < f(x)$. Using inequality (A.12), we conclude that $f\bigl(\alpha x + (1-\alpha)y\bigr) < f(x)$ for every $\alpha \in [0,1)$. This contradicts the assumption that x is a local minimum and proves part (f). To prove part (g), suppose that two global minima x and y existed. Then their average $(x+y)/2$ would belong to C, since C is convex, and the value of f would be smaller at that point by the strict convexity of f. **Q.E.D.**

Proposition A.36. If $f : \Re^n \mapsto \Re$ is convex, then it is continuous. More generally, if $C \subset \Re^n$ is convex and $f : C \mapsto \Re$ is convex, then f is continuous in the interior of C.

Proof. Without any loss of generality, it is sufficient to prove continuity at the origin, assuming that $0 \in C$ and that the unit cube $S = \{z \mid \|x\|_\infty \leq 1\}$ is contained in C. Let e^i, $i = 1, \ldots, 2^n$, be the corners of S, that is, each e^i is a vector whose entries belong to $\{-1, 1\}$. It is not difficult to see that any $x \in S$ can be expressed in the form $x = \sum_{i=1}^{2^n} a_i e^i$, where each a_i is a nonnegative scalar and $\sum_{i=1}^{2^n} a_i = 1$. Let $A = \max_i f(e^i)$. From Prop. A.35(a), it follows that $f(x) \leq A$ for every $x \in S$.

Let $\{x^k\}$ be a sequence in $\Re^n$ that converges to zero. For the purpose of proving continuity at zero, we can assume that $x^k \in S$ for all k. Using (A.12), we have

$$f(x^k) \leq (1 - \|x^k\|_\infty) f(0) + \|x^k\|_\infty f\left(\frac{x^k}{\|x^k\|_\infty}\right).$$

Letting k tend to infinity, $\|x^k\|_\infty$ goes to zero and we obtain

$$\limsup_{k\to\infty} f(x^k) \leq f(0) + A \limsup_{k\to\infty} \|x^k\|_\infty = f(0).$$

Inequality (A.12) also implies that

$$f(0) \leq \frac{\|x^k\|_\infty}{\|x^k\|_\infty + 1} f\left(\frac{-x^k}{\|x^k\|_\infty}\right) + \frac{1}{\|x^k\|_\infty + 1} f(x^k)$$

and letting k tend to infinity, we obtain $f(0) \leq \liminf_{k\to\infty} f(x^k)$. Thus, $\lim_{k\to\infty} f(x^k) = f(0)$ and f is continuous at zero. **Q.E.D.**

Proposition A.37. Let $I \subset \Re$ be convex. (Thus, I is an interval.) Let $f : I \mapsto \Re$ be convex. If $x, y, z \in I$ and $x < y < z$, then

$$\frac{f(y) - f(x)}{y - x} \leq \frac{f(z) - f(x)}{z - x} \leq \frac{f(z) - f(y)}{z - y}.$$

Proof. Using inequality (A.12), we obtain

$$f(y) \leq \left(\frac{y - x}{z - x}\right) f(z) + \left(\frac{z - y}{z - x}\right) f(x)$$

and either of the desired inequalities follows by appropriately rearranging terms. **Q.E.D.**

Let $I \subset \Re$ be an interval and let $f : I \mapsto \Re$ be convex. Let a and b be the infimum and the supremum, respectively, of I. For any $x \in I$, $x \neq b$, and for any $\alpha > 0$ such that $x + \alpha \in I$, we define

$$s^+(x, \alpha) = \frac{f(x + \alpha) - f(x)}{\alpha}.$$

Let $0 < \alpha \leq \alpha'$. We use the first inequality in Prop. A.37 with $y = x + \alpha$ and $z = x + \alpha'$ to obtain $s^+(x, \alpha) \leq s^+(x, \alpha')$. Therefore, $s^+(x, \alpha)$ is a nondecreasing function of α and, as α decreases to zero, it converges either to a finite number or to $-\infty$. Let $f^+(x)$ be the value of the limit, which we call the *right derivative* of f at the point x. Similarly, if $x \in I$, $x \neq a$, $\alpha > 0$, and $x - \alpha \in I$, we define

$$s^-(x, \alpha) = \frac{f(x) - f(x - \alpha)}{\alpha},$$

which is, by a symmetrical argument, a nonincreasing function of α. Its limit as α decreases to zero, denoted by $f^-(x)$, is called the *left derivative* of f at the point x, and is either finite or equal to ∞.

In case the end points a and b belong to the domain I of f, we define for completeness $f^-(a) = -\infty$ and $f^+(b) = \infty$.

Proposition A.38. Let $I \subset \Re$ be convex and let $f : I \mapsto \Re$ be a convex function. Let a and b be the end points of I as above.

(a) We have $f^-(y) \leq f^+(y)$ for every $y \in I$.

(b) If x belongs to the interior of I, then $f^+(x)$ and $f^-(x)$ are finite.

(c) If $x, z \in I$ and $x < z$, then $f^+(x) \leq f^-(z)$.

(d) The functions $f^-, f^+ : I \mapsto [-\infty, +\infty]$ are nondecreasing.

(e) The function f^+ (respectively, f^-) is right– (respectively, left–) continuous at every interior point of I. Also, if $a \in I$ (respectively, $b \in I$) and f is continuous at a (respectively, b), then f^+ (respectively, f^-) is right– (respectively, left–) continuous at a (respectively, b).

(f) If f is differentiable at a point x belonging to the interior of I, then $f^+(x) = f^-(x) = (df/dx)(x)$.

(g) For any $x, z \in I$ and any d satisfying $f^-(x) \leq d \leq f^+(x)$, we have

$$f(z) \geq f(x) + d(z - x).$$

(h) The function $f^+ : I \mapsto (-\infty, \infty]$ [respectively, $f^- : I \mapsto [-\infty, \infty)$] is upper (respectively, lower) semicontinuous at every $x \in I$.

Proof.

(a) If y is an end point of I, the result is trivial because $f^-(a) = -\infty$ and $f^+(b) = \infty$. We assume that y is an interior point, we let $\alpha > 0$, and use Prop. A.37, with $x = y - \alpha$ and $z = y + \alpha$, to obtain $s^-(y, \alpha) \leq s^+(y, \alpha)$. Taking the limit as α decreases to zero, we obtain $f^-(y) \leq f^+(y)$.

(b) Let x belong to the interior of I and let $\alpha > 0$ be such that $x - \alpha \in I$. Then $f^-(x) \geq s^-(x, \alpha) > -\infty$. For similar reasons, we obtain $f^+(x) < \infty$. Part (a) then implies that $f^-(x) < \infty$ and $f^+(x) > -\infty$.

(c) We use Prop. A.37, with $y = (z + x)/2$, to obtain $s^+\big(x, (z - x)/2\big) \leq s^-\big(z, (z - x)/2\big)$. The result then follows because $f^+(x) \leq s^+\big(x, (z - x)/2\big)$ and $s^-\big(z, (z - x)/2\big) \leq f^-(z)$.

(d) This follows by combining parts (a) and (c).

(e) Fix some $x \in I$, $x \neq b$, and some positive δ and α such that $x + \delta + \alpha < b$. We allow x to be equal to a, in which case f is assumed to be continuous at a. We have $f^+(x + \delta) \leq s^+(x + \delta, \alpha)$. We take the limit, as δ decreases to zero, to obtain $\lim_{\delta \downarrow 0} f^+(x + \delta) \leq s^+(x, \alpha)$. We have used here the fact that $s^+(x, \alpha)$ is a continuous function of x, which is a consequence of the continuity of f (Prop. A.36). We now let α decrease to zero to obtain $\lim_{\delta \downarrow 0} f^+(x + \delta) \leq f^+(x)$. The reverse inequality is also true because f^+ is nondecreasing and this proves the right–continuity of f^+. The proof for f^- is similar.

(f) This is immediate from the definition of f^+ and f^-.

(g) Fix some $x, z \in I$. The result is trivially true for $x = z$. We only consider the case $x < z$; the proof for the case $x > z$ is similar. Since $s^+(x, \alpha)$ is nondecreasing in α, we have $\big(f(z) - f(x)\big)/(z - x) \geq s^+(x, \alpha)$ for α belonging to $(0, z - x)$. Letting α decrease to zero, we obtain $\big(f(z) - f(x)\big)/(z - x) \geq f^+(x) \geq d$ and the result follows.

(h) This follows from parts (d), (e), and the definition of semicontinuity (Definition A.4). **Q.E.D.**

Any $d \in \Re$ satisfying $f^-(x) \leq d \leq f^+(x)$ is called a *subgradient* of f at x. The following result generalizes Prop. A.38(g) to the case of a multivariable and differentiable convex function.

Proposition A.39. Let $C \subset \Re^n$ be a convex set and let $f : \Re^n \mapsto \Re$ be differentiable.

(a) The function f is convex on the set C if and only if

$$f(z) \geq f(x) + (z - x)'\nabla f(x), \qquad \forall x, z \in C. \tag{A.13}$$

(b) If the inequality (A.13) is strict whenever $x \neq z$, then f is strictly convex on C.

(c) Suppose that $C = \Re^n$ and that f is convex. We have $\nabla f(x) = 0$ if and only if x is a global minimum of f.

Proof.

(a) Suppose that f is convex on C. Let $x \in C$ and $z \in C$. By the convexity of C, we obtain $x + \alpha(z - x) \in C$ for every $\alpha \in [0, 1]$. Furthermore,

$$\lim_{\alpha \downarrow 0} \frac{f\big(x + \alpha(z - x)\big) - f(x)}{\alpha} = (z - x)'\nabla f(x).$$

Using the convexity of f, we have

$$f\big(x + \alpha(z - x)\big) \leq \alpha f(z) + (1 - \alpha)f(x), \qquad \forall \alpha \in [0, 1].$$

Using this inequality to replace the $f\big(x+\alpha(z-x)\big)$ term in the previous inequality, we obtain (A.13).

For the proof of the converse, suppose that inequality (A.13) is true. We fix some $x, y \in C$ and some $\alpha \in [0, 1]$. Let $z = \alpha x + (1 - \alpha)y$. Using inequality (A.13) twice, we obtain

$$\begin{aligned} f(x) &\geq f(z) + (x - z)'\nabla f(z), \\ f(y) &\geq f(z) + (y - z)'\nabla f(z). \end{aligned}$$

We multiply the first inequality by α, the second by $(1 - \alpha)$, and add them to obtain

$$\alpha f(x) + (1 - \alpha)f(y) \geq f(z) + \big(\alpha x + (1 - \alpha)y - z\big)'\nabla f(z) = f(z),$$

which proves that f is convex.

(b) The proof for the strictly convex case is almost identical to the proof for part (a) and is omitted.

(c) If $\nabla f(x) = 0$, then inequality (A.13) shows that $f(z) \geq f(x)$ for all $z \in \Re^n$, and x is a global minimum. Conversely, if x is a global minimum, then it is also a local minimum, and Prop. A.34 shows that $\nabla f(x) = 0$. **Q.E.D.**

Proposition A.40. Let $f : \Re^n \mapsto \Re$ be twice continuously differentiable, and let A be a real symmetric $n \times n$ matrix.

(a) The function f is convex if and only if $\nabla^2 f(x)$ is nonnegative definite for all x.

(b) If $\nabla^2 f(x)$ is positive definite for every x, then f is strictly convex.

(c) The function $f(x) = x'Ax$ is convex if and only if A is nonnegative definite.

(d) The function $f(x) = x'Ax$ is strictly convex if and only if A is positive definite. In particular, $\|x\|_2^2 = x'Ix$ is strictly convex.

Proof.

(a) If $\nabla^2 f(x)$ is nonnegative definite for all x, then Prop. A.31(b) shows that $f(x + y) \geq f(x) + y'\nabla f(x)$ for all $x, y \in \Re^n$. Using Prop. A.39(a), we conclude that f is convex. Conversely, suppose that f is convex and suppose, to derive a contradiction, that there exist some x, y such that $y'\nabla^2 f(x)y < 0$. Using the continuity of $\nabla^2 f$, we see that we can choose the magnitude of y to be small enough so that $y'\nabla^2 f(x + \alpha y)y < 0$ for every $\alpha \in [0, 1]$. Then Prop. A.31(b) yields $f(x + y) < f(x) + y'\nabla f(x)$, which, in view of Prop. A.39(a), contradicts the convexity of f.

(b) The proof is similar to the proof of the corresponding statement in part (a).

(c) An easy calculation shows that $\nabla^2 f(x) = 2A$ for all $x \in \Re^n$, and the result follows from part (a).

(d) If A is positive definite, then strict convexity of f follows from part (b). For the converse, suppose that f is strictly convex. Then part (c) implies that A is nonnegative definite and it remains to show that A is actually positive definite. In view of Prop. A.26(b), it suffices to show that zero is not an eigenvalue of A. Suppose the contrary. Then there exists some $x \neq 0$ such that $Ax = -Ax = 0$. It follows that $\frac{1}{2}\big(f(x) + f(-x)\big) = 0 = f(0)$, which contradicts the strict convexity of f. **Q.E.D.**

Proposition A.41. (*Strong Convexity*) Let $f : \Re^n \mapsto \Re^n$ be continuously differentiable and let α be a positive constant. If f satisfies the condition

$$\big(\nabla f(x) - \nabla f(y)\big)'(x - y) \geq \alpha\|x - y\|_2^2, \qquad \forall x, y \in \Re^n, \qquad (A.14)$$

then f is strictly convex. Furthermore, if f is twice continuously differentiable, then the condition (A.14) is equivalent to the nonnegative definiteness of $\nabla^2 f(x) - \alpha I$ for every $x \in \Re^n$, where I is the identity matrix.

Proof. Fix some $x, y \in \Re^n$ such that $x \neq y$, and define the function $h : \Re \mapsto \Re$ by $h(t) = f\big(x + t(y - x)\big)$. Consider some $t, t' \in \Re$ such that $t < t'$. Using the chain rule and Eq. (A.14), we have

$$(t'-t)\Big(\frac{dh}{dt}(t')-\frac{dh}{dt}(t)\Big)=\Big(\nabla f\big(x+t'(y-x)\big)-\nabla f\big(x+t(y-x)\big)\Big)'(y-x)(t'-t)$$
$$\geq \alpha(t'-t)^2\|x-y\|_2^2>0.$$

Thus, dh/dt is strictly increasing and for any $t\in(0,1)$, we have

$$\frac{h(t)-h(0)}{t}=\frac{1}{t}\int_0^t\frac{dh}{d\tau}(\tau)\,d\tau<\frac{1}{1-t}\int_t^1\frac{dh}{d\tau}(\tau)\,d\tau=\frac{h(1)-h(t)}{1-t}.$$

Equivalently, $th(1)+(1-t)h(0)>h(t)$. The definition of h yields $tf(y)+(1-t)f(x)>f\big(ty+(1-t)x\big)$. Since this inequality has been proved for arbitrary $t\in(0,1)$ and $x\neq y$, we conclude that f is strictly convex.

Suppose now that f is twice continuously differentiable and Eq. (A.14) holds. Let c be a scalar variable. We use Proposition A.31(b) twice to obtain

$$f(x+cy)=f(x)+cy'\nabla f(x)+\frac{c^2}{2}y'\nabla^2f(x+tcy)y,$$

and

$$f(x)=f(x+cy)-cy'\nabla f(x+cy)+\frac{c^2}{2}y'\nabla^2f(x+scy)y,$$

for some t and s belonging to $[0,1]$. Adding these two inequalities and using (i) we obtain

$$\frac{c^2}{2}y'\Big(\nabla^2f(x+scy)+\nabla^2f(x+tcy)\Big)y=\big(\nabla f(x+cy)-\nabla f(x)\big)'(cy)\geq \alpha c^2\|y\|_2^2.$$

We divide both sides by c^2 and then take the limit as c tends to zero to conclude that $y'\nabla^2f(x)y\geq\alpha\|y\|_2^2$. Since this inequality is valid for every $y\in\Re^n$, it follows that $\nabla^2f(x)-\alpha I$ is nonnegative definite.

For the converse, assume that $\nabla^2f(x)-\alpha I$ is nonnegative definite for all $x\in\Re^n$. Consider the function $g:\Re\mapsto\Re$ defined by

$$g(t)=\nabla f\big(tx+(1-t)y\big)'(x-y).$$

Using the Mean Value Theorem (Prop. A.30), we have $\big(\nabla f(x)-\nabla f(y)\big)'(x-y)=g(1)-g(0)=(dg/dt)(t)$ for some $t\in[0,1]$. The result follows because

$$\frac{dg}{dt}(t)=(x-y)'\nabla^2f\big(tx+(1-t)y\big)(x-y)\geq\alpha\|x-y\|_2^2,$$

where the last inequality is a consequence of the nonnegative definiteness of $\nabla^2 f\big(tx + (1-t)y\big) - \alpha I$. **Q.E.D.**

Notice that the directional derivative $f'(x;y)$ of a convex function $f : \Re^n \mapsto \Re$ at a vector $x \in \Re^n$ in the direction $y \in \Re^n$ is equal to the right derivative $F_y^+(0)$ of the convex scalar function $F_y(\alpha) = f(x+\alpha y)$ at $\alpha = 0$, that is,

$$f'(x;y) = \lim_{\alpha \downarrow 0} \frac{f(x+\alpha y) - f(x)}{\alpha}, \tag{A.15}$$

and, in particular, the limit in Eq. (A.15) is guaranteed to exist. Similarly, the left derivative $F_y^-(0)$ of F_y is equal to $-f'(x;-y)$ and, by using Prop. A.38(a), we obtain

$$-f'(x;-y) \le f'(x;y), \qquad \forall\, y \in \Re^n. \tag{A.16}$$

The following proposition generalizes the upper semicontinuity property of right derivatives of scalar convex functions [Prop. A.38(h)], and shows that if f is differentiable, then its gradient is continuous.

Proposition A.42. Let $f : \Re^n \mapsto \Re$ be convex, and let $\{f_k\}$ be a sequence of convex functions $f_k : \Re^n \mapsto \Re$ with the property that $\lim_{k\to\infty} f_k(x_k) = f(x)$ for every $x \in \Re^n$ and every sequence $\{x_k\}$ that converges to x. Then for any $x \in \Re^n$ and $y \in \Re^n$, and any sequence $\{x_k\}$ converging to x, we have

$$\limsup_{k\to\infty} f_k'(x_k;y) \le f'(x;y). \tag{A.17}$$

Furthermore, if f is differentiable at all $x \in \Re^n$, then its gradient $\nabla f(x)$ is a continuous function of x.

Proof. For any $\mu > f'(x;y)$, there exists an $\alpha > 0$ such that

$$\frac{f(x+\alpha y) - f(x)}{\alpha} < \mu.$$

Hence, for all sufficiently large k, we have

$$\frac{f_k(x_k+\alpha y) - f_k(x_k)}{\alpha} < \mu,$$

and since $f_k'(x_k;y)$ does not exceed the left–hand side of the last inequality, we obtain that

$$\limsup_{k\to\infty} f_k'(x_k;y) < \mu.$$

Since this is true for all $\mu > f'(x;y)$, inequality (A.17) follows.

If f is differentiable at all $x \in \Re^n$, then using the continuity of f and the part of the proposition just proved, we have for every $\{x_k\}$ converging to x and every $y \in \Re^n$,

$$\limsup_{k\to\infty} \nabla f(x_k)'y = \limsup_{k\to\infty} f'(x_k; y) \le f'(x; y) = \nabla f(x)'y.$$

By replacing y by $-y$ in the preceding argument, we obtain

$$-\liminf_{k\to\infty} \nabla f(x_k)'y = \limsup_{k\to\infty} \big(-\nabla f(x_k)'y\big) \le -\nabla f(x)'y.$$

Therefore, we have $\nabla f(x_k)'y \to \nabla f(x)'y$ for every y, which implies that $\nabla f(x_k) \to \nabla f(x)$. Hence, the gradient is continuous. **Q.E.D.**

We will encounter functions of the form $f(x) = \max_{z\in Z} \phi(x, z)$ when dealing with dual optimization problems (see Appendix C). The following result characterizes the directional derivatives of f.

Proposition A.43. (*Danskin's Theorem* [Dan67]) Let $Z \subset \Re^m$ be a compact set, and let $\phi : \Re^n \times Z \mapsto \Re$ be a continuous function such that $\phi(\cdot, z) : \Re^n \mapsto \Re$, viewed as a function of its first argument, is convex for each $z \in Z$.

(a) The function $f : \Re^n \mapsto \Re$ given by

$$f(x) = \max_{z\in Z} \phi(x, z)$$

is convex and has directional derivative given by

$$f'(x; y) = \max_{z\in Z(x)} \phi'(x, z; y), \tag{A.18}$$

where $\phi'(x, z; y)$ is the directional derivative of the function $\phi(\cdot, z)$ at x in the direction y, and

$$Z(x) = \big\{\bar{z} \mid \phi(x, \bar{z}) = \max_{z\in Z} \phi(x, z)\big\}. \tag{A.19}$$

In particular, if $Z(x)$ consists of a unique point $\bar{z}$ and $\phi(\cdot, \bar{z})$ is differentiable at x, then f is differentiable at x, and $\nabla f(x) = \nabla_x \phi(x, \bar{z})$, where $\nabla_x \phi(x, \bar{z})$ is the vector with coordinates $(\partial\phi/\partial x_i)(x, \bar{z})$, $i = 1, \ldots, n$.

(b) The conclusion of (a) also holds if, instead of assuming that Z is compact, we assume that $Z(x)$ is nonempty for all $x \in \Re^n$, and that ϕ and Z are such that for every sequence $\{x_k\}$ converging to some x, there exists a bounded sequence $\{z_k\}$ with $z_k \in Z(x_k)$ for all k.

Proof. We only prove part (a). The proof of part (b) is almost identical. The convexity of f has been established in Prop. A.35(d). We note that since ϕ is continuous and Z is compact, the set $Z(x)$ is nonempty by Weierstrass' theorem (Prop. A.8) and f is finite. For any $z \in Z(x)$, $y \in \Re^n$, and $\alpha > 0$, we use the definition of f to obtain

$$\frac{f(x+\alpha y) - f(x)}{\alpha} \geq \frac{\phi(x+\alpha y, z) - \phi(x,z)}{\alpha}.$$

Taking the limit as α decreases to zero, we obtain $f'(x;y) \geq \phi'(x,z;y)$. Since this is true for every $z \in Z(x)$, we conclude that

$$f'(x;y) \geq \sup_{z \in Z(x)} \phi'(x,z;y), \qquad \forall\, y \in \Re^n. \tag{A.20}$$

To prove the reverse inequality and that the supremum in the right–hand side of inequality (A.20) is attained, consider a sequence $\{\alpha_k\}$ of positive scalars that converges to zero and let $x_k = x + \alpha_k y$. For each k, let z_k be a vector in $Z(x_k)$. Since $\{z_k\}$ belongs to the compact set Z, it has a subsequence converging to some $\bar{z} \in Z$. Without loss of generality, we assume that the entire sequence $\{z_k\}$ converges to $\bar{z}$. We have

$$\phi(x_k, z_k) \geq \phi(x_k, z), \qquad \forall\, z \in Z,$$

so by taking the limit as $k \to \infty$ and by using the continuity of ϕ, we obtain

$$\phi(x, \bar{z}) \geq \phi(x, z), \qquad \forall\, z \in Z.$$

Therefore, $\bar{z} \in Z(x)$. We now have

$$\begin{aligned} f'(x;y) &\leq \frac{f(x+\alpha_k y) - f(x)}{\alpha_k} = \frac{\phi(x+\alpha_k y, z_k) - \phi(x,\bar{z})}{\alpha_k} \leq \frac{\phi(x+\alpha_k y, z_k) - \phi(x, z_k)}{\alpha_k} \\ &\leq -\phi'(x+\alpha_k y, z_k; -y) \leq \phi'(x+\alpha_k y, z_k; y), \end{aligned} \tag{A.21}$$

where the last inequality follows from inequality (A.16). We apply Prop. A.42 to the functions f_k defined by $f_k(\cdot) = \phi(\cdot, z_k)$, and with $x_k = x + \alpha_k y$, to obtain

$$\limsup_{k \to \infty} \phi'(x+\alpha_k y, z_k; y) \leq \phi'(x, \bar{z}; y). \tag{A.22}$$

We take the limit in inequality (A.21) as $k \to \infty$, and use inequality (A.22) to conclude that

$$f'(x;y) \leq \phi'(x, \bar{z}; y).$$

This relation together with inequality (A.20) proves Eq. (A.18).

For the last statement of part (a), if $Z(x)$ consists of the unique point $\bar{z}$, Eq. (A.18) and the differentiability assumption on ϕ yield

$$f'(x;y) = \phi'(x,\bar{z};y) = y'\nabla_x\phi(x,\bar{z}), \qquad \forall y \in \Re^n,$$

which implies that $\nabla f(x) = \nabla_x \phi(x,\bar{z})$. **Q.E.D.**

Linear Differential Equations

Consider the differential equation

$$\frac{dx}{dt}(t) = A(t)x(t) + B(t)u(t). \tag{A.23}$$

Here $x(t) \in \Re^n$, $u(t) \in \Re^m$, and $A(t)$, $B(t)$ are matrices of dimensions $n \times n$ and $n \times m$, respectively. We assume that $u(t)$, $A(t)$, and $B(t)$ are continuous functions of t. Then it can be shown that for any given initial condition $x(0) = x_0 \in \Re^n$, the differential equation (A.23) has a unique solution over the interval $[0,\infty)$ [CoL55].

Proposition A.44.

(a) Under the above continuity assumptions, the solution of Eq. (A.23) admits the representation

$$x(t) = \int_0^t \Phi(t,\tau)B(\tau)u(\tau)\,d\tau + \Phi(t,0)x_0, \tag{A.24}$$

where $\Phi(t,\tau)$ is an $n \times n$ matrix satisfying $\Phi(\tau,\tau) = I$ for every $\tau \geq 0$, and

$$\frac{d\Phi}{dt}(t,\tau) = A(t)\Phi(t,\tau), \qquad \forall \tau,\ \forall t \geq \tau. \tag{A.25}$$

(b) Suppose that $A(t)$ is a continuous and bounded function of time. Let $\|\cdot\|$ be some induced matrix norm. Then there exist constants C and c such that

$$\|\Phi(t,\tau)\| \leq Ce^{c(t-\tau)}, \qquad \forall \tau \geq 0,\ \forall t \geq \tau. \tag{A.26}$$

Proof.

(a) We first notice that Eq. (A.25) defines $\Phi(t,\tau)$ uniquely, because of the existence and uniqueness result quoted earlier. Let us define $x(t)$, $t \geq 0$, by Eq. (A.24). We notice that $x(t)$ satisfies the initial condition $x(0) = x_0$ and, by differentiating the right–hand side of Eq. (A.24), we see that $x(t)$ also satisfies Eq. (A.23). The result follows because Eq. (A.23) has a unique solution.

(b) Let D_1 be a bound on $\|A(t)\|$. Let $D_2 = \|I\|$. Let $c = D_1$ and $C = 2D_2$. Notice that inequality (A.26) is true if $t = \tau$. Fix some $\tau \geq 0$. Suppose, to derive a contradiction, that there exists some $t > \tau$ such that inequality (A.26) fails to hold. Let t^* be the infimum of the set of all such t. Since $\Phi(t, \tau)$ is a continuous function of t, it follows that inequality (A.26) holds with equality at time t^*. We integrate Eq. (A.25) and take the norm of both sides to obtain

$$\begin{aligned}
2D_2 e^{D_1(t^*-\tau)} = \|\Phi(t^*, \tau)\| &\leq \left\| \int_\tau^{t^*} A(t)\Phi(t,\tau)\, dt \right\| + \|\Phi(\tau,\tau)\| \\
&\leq \int_\tau^{t^*} \|A(t)\| \cdot \|\Phi(t,\tau)\|\, dt + D_2 \\
&\leq \int_\tau^{t^*} D_1 2D_2 e^{D_1(t-\tau)}\, dt + D_2 \\
&= 2D_2 e^{D_1(t^*-\tau)} - 2D_2 + D_2 < 2D_2 e^{D_1(t^*-\tau)},
\end{aligned}$$

which is a contradiction. **Q.E.D.**

B

Graph Theory

In this appendix, we collect the definitions and notational conventions relating to graph theory that we will use throughout the book. For further material, consult [Chr75], [Har69], [Law76], [PaS82], and [Roc84].

Undirected Graphs

We define a *graph*, $G = (N, A)$, to be a finite nonempty set N of *nodes* and a collection A of pairs of distinct nodes from N. Each pair of nodes in A is called an *arc* (or *link* in some contexts). An arc (i, j) is viewed as an unordered pair, and is indistinguishable from the pair (j, i); thus $(i, j) = (j, i)$. If (i, j) is an arc, we say that (i, j) is *incident* to i and to j, and we say that i and j are *adjacent* nodes (or *neighboring* nodes or *neighbors*). The *degree* of a node i is the number of arcs that are incident to i. The numbers of nodes and arcs of G are denoted by $|N|$ and $|A|$, respectively.

Note that we have disallowed self–arcs, that is, arcs connecting a node with itself. We have also disallowed multiple arcs between the same pair of nodes, and thus we can refer unambiguously to the arc between nodes i and j as arc (i, j). This was done for notational convenience; all of our analysis can be simply extended to the case of graphs that can have multiple arcs between any pair of distinct nodes. The standard method for doing this is to replace each arc between nodes i and j by an additional node, call it n, together with the two arcs (i, n) and (n, j).

A *walk* in a graph G is a finite sequence of nodes $(n_1, n_2, \ldots, n_k)$ such that $(n_1, n_2), (n_2, n_3), \ldots, (n_{k-1}, n_k)$ are arcs of G. A walk $(n_1, \ldots, n_k)$ with $n_1 = n_k$, $k \geq 3$, is called a *cycle*. A walk is said to be *simple* if it contains no repeated nodes, except possibly for the start and end nodes, in which case, it is a (simple) cycle. We say that a graph is *connected* if for each node i, there is a walk $(i = n_1, n_2, \ldots, n_k = j)$ to each other node j.

We say that $G' = (N', A')$ is a *subgraph* of $G = (N, A)$ if G' is a graph, $N' \subset N$, and $A' \subset A$. A *tree* is a connected graph that contains no cycles. A *spanning tree* of a graph G is a subgraph of G that is a tree and that includes all the nodes of G. The following proposition gives a basic result on trees and spanning trees:

Proposition B.1. Let $G = (N, A)$ be a connected graph with $|N|$ nodes and $|A|$ arcs. Then:

(a) G contains a spanning tree.
(b) $|A| \geq |N| - 1$.
(c) G is a tree if and only if $|A| = |N| - 1$.

Proof.

(a) We show that the following algorithm constructs a spanning tree:
 1. Let n be an arbitrary node in N. Let $N' = \{n\}$, $A' =$ empty.
 2. If $N' = N$, then stop [$G' = (N', A')$ is a spanning tree]; else go to Step 3.
 3. Let $(i, j) \in A$ be an arc with $i \in N'$ and $j \notin N'$. Update N' and A' by

$$N' := N' \cup \{j\},$$
$$A' := A' \cup \{(i, j)\}.$$

Go to Step 2.

To see why the algorithm works, note that Step 3 is only entered when N' is a proper subset of N, so that the existence of the arc (i, j) in Step 3 follows from the connectedness of G. We use induction on successive executions of Step 3 to show that $G' = (N', A')$ is always a tree. Initially, $G' = (\{n\}, \text{empty})$ is trivially a tree, so assume that $G' = (N', A')$ is a tree before the update of Step 3. This ensures that there is a walk between each pair of nodes in N' using arcs of A'. After node j and arc (i, j) are added, each node has a walk to j simply by adding j to the walk to i, and, similarly, j has a walk to each other node. Finally, node j cannot be in any cycles since (i, j) is the only arc of G' incident to j. Furthermore, there are no cycles not including j by the induction hypothesis.

(b) Observe that the algorithm starts with a subgraph having one node and zero arcs, and adds one node and one arc on each execution of Step 3. This means that the spanning tree G' resulting from the algorithm always has $|N|$ nodes and $|N| - 1$ arcs. Since G' is a subgraph of G, the number of arcs $|A|$ in G must satisfy $|A| \geq |N| - 1$.

(c) Assume that $|A| = |N| - 1$. This means that the algorithm uses all arcs of G in the spanning tree G', so that $G = G'$ and G must be a tree itself. Conversely, if $|A| \geq |N|$, then G contains at least one arc (i, j) not in the spanning tree G' generated by the algorithm. Letting $(j, \ldots, i)$ be the walk from j to i in G', it is seen that $(i, j, \ldots, i)$ is a cycle in G and G cannot be a tree. **Q.E.D.**

Directed Graphs

A *directed graph* or *digraph* $G = (N, A)$ is a finite nonempty set N of nodes and a collection A of *ordered* pairs of distinct nodes from N; each ordered pair of nodes in A is called a *directed arc* (or simply arc). Thus, a directed graph can be viewed as a graph where each arc has a direction associated with it. We do not allow more than one arc between a pair of nodes in the same direction, but we do not exclude the possibility that there is a separate arc connecting a pair of nodes in each of the two directions. If (i, j) is a directed arc, we say that (i, j) is an *outgoing* arc from node i, and an *incoming* arc to node j; we also say that (i, j) is *incident* to i and to j, and that i and j are *adjacent* nodes (or *neighboring* nodes or *neighbors*).

A *path* P in a directed graph is a sequence of nodes $(n_1, n_2, \ldots, n_k)$ with $k \geq 2$ and a corresponding sequence of $k - 1$ arcs such that the ith arc in the sequence is either (n_i, n_{i+1}) (in which case it is called a *forward* arc of the path) or (n_{i+1}, n_i) (in which case, it is called a *backward* arc of the path). We denote by P^+ and P^- the sets of forward and backward arcs of P, respectively. The arcs in P^+ and P^- are said to *belong* to P. Nodes n_1 and n_k are called the *start node* (or *origin*) and *end node* (or *destination*) of P, respectively.

A *directed cycle* (or simply *cycle* when confusion cannot arise) is a path for which the start and end nodes are the same. A path is said to be *simple* if it contains no repeated nodes except possibly for the start and end nodes, in which case, it is a (simple) cycle. A path is said to be *positive* (or *negative*) if all of its arcs are forward (respectively, backward) arcs. We refer similarly to a positive and a negative cycle. A digraph that does not contain any positive cycles is said to be *acyclic*.

A digraph is said to be *connected* if for each pair of nodes i and j, there is a path starting at i and ending at j; it is said to be *strongly connected* if for each pair of nodes i and j, there is a positive path starting at i and ending at j.

Flows

A *flow vector* f in a directed graph (N, A) is a set of real numbers $\{f_{ij} \mid (i, j) \in A\}$. We refer to f_{ij} as the flow of the arc (i, j). The *divergence vector* y associated with a flow vector is the $|N|$–dimensional vector with coordinates

$$y_i = \sum_{\{j|(i,j)\in A\}} f_{ij} - \sum_{\{j|(i,j)\in A\}} f_{ji}, \qquad \forall\, i \in N. \tag{B.1}$$

Thus, y_i is the total flow departing from node i less the total flow arriving at i. We say that node i is a *source* (respectively, *sink*) for the flow vector f if $y_i > 0$ (respectively, $y_i < 0$). If $y_i = 0$ for all $i \in N$, then f is called a *circulation.*

We say that a path P *conforms* to a flow vector f if $f_{ij} > 0$ for all $(i,j) \in P^+$ and $f_{ij} < 0$ for all $(i,j) \in P^-$, and either P is a cycle or else the start and end nodes of P are a source and a sink of f, respectively. A *simple path flow* is a flow vector f of the form

$$f_{ij} = \begin{cases} a & \text{if } (i,j) \in P^+, \\ -a & \text{if } (i,j) \in P^-, \\ 0 & \text{otherwise,} \end{cases} \tag{B.2}$$

where a is a positive scalar, and P^+ and P^- are the sets of forward and backward arcs, respectively, of a simple path P. We say that a simple path flow f^s *conforms* to a flow vector f if the path P corresponding to f^s via Eq. (B.2) conforms to f. The following is an important result proved in several sources, (e.g., [Roc84], p. 103).

Conformal Realization Theorem. A nonzero flow vector f can be decomposed into the sum of finitely many simple path flow vectors $f^1, f^2, \ldots, f^k$ that conform to f. If f is integer, then $f^1, f^2, \ldots, f^k$ can also be chosen to be integer. If f is a circulation, then $f^1, f^2, \ldots, f^k$ can be chosen to be circulations.

Proof. We first assume that f is a circulation. Our proof consists of showing how to obtain from f a simple circulation f' such that

$$0 \le f_{ij} \quad \Rightarrow \quad 0 \le f'_{ij} \le f_{ij}, \tag{B.3a}$$

$$f_{ij} \le 0 \quad \Rightarrow \quad f_{ij} \le f'_{ij} \le 0, \tag{B.3b}$$

$$f_{ij} = f'_{ij} \neq 0 \qquad \text{for at least one arc } (i,j). \tag{B.3c}$$

Once this is done, we have $f_{ij} - f'_{ij} > 0$ (< 0) only if $f_{ij} > 0$ $(f_{ij} < 0)$, and $f_{ij} - f'_{ij} = 0$ for at least one arc (i,j) with $f_{ij} \neq 0$. If f is integer, then f' and $f - f'$ will also be integer. We then repeat the process with the circulation f replaced by the circulation $f - f'$ and so on until the zero flow is obtained. This is guaranteed to happen eventually because $f - f'$ has at least one more arc with zero flow than f.

We now describe the procedure by which f' with the properties (B.3) is obtained. Choose an arc (i,j) with $f_{ij} \neq 0$. Assume that $f_{ij} > 0$. (A similar procedure can be used when $f_{ij} < 0$.) Take $T_0 = \{j\}$. Given T_k, let

$$T_{k+1} = \big\{ n \notin \cup_{p=0}^{k} T_p \,|\, \text{there is a node } m \in T_k, \text{ and either an arc } (m,n)$$
$$\text{such that } f_{mn} > 0, \text{ or an arc } (n,m) \text{ such that } f_{nm} < 0 \big\},$$

and mark each node $n \in T_{k+1}$ with the label "m", where m is a node of T_k such that $f_{mn} > 0$ or $f_{nm} < 0$. We claim that one of the sets T_k contains node i. To see this note that there is no outgoing arc from $\cup_k T_k$ with positive flow and no incoming arc into $\cup_k T_k$ with negative flow. If i did not belong to $\cup_k T_k$, there would exist at least one incoming arc into $\cup_k T_k$ with positive flow, namely, the arc (i,j). Thus, the total incoming flow into $\cup_k T_k$ would not be equal to the total outgoing flow from $\cup_k T_k$, and this contradicts the fact that f is a circulation. Therefore, one of the sets T_k contains node i.

We now trace labels backwards from i until node j is reached. (This will happen eventually because if "m" is the label of node n and $n \in T_{k+1}$, then $m \in T_k$, so a "cycle" of labels cannot be formed before reaching j.) In particular, let "i_1" be the label of i, let "i_2" be the label of i_1, etc., until a node i_k with label "j" is found. The cycle $C = (j, i_k, i_{k-1}, \ldots, i_1, i, j)$ is simple, it contains (i,j) as a forward arc, and is such that all its forward arcs have positive flow and all its backward arcs have negative flow (see Fig. B.1). Let $a = \min_{(m,n)\in C} |f_{mn}| > 0$. Then the circulation f', where

$$f'_{ij} = \begin{cases} a & \text{if } (i,j) \in C^+, \\ -a & \text{if } (i,j) \in C^-, \\ 0 & \text{otherwise,} \end{cases} \tag{B.4}$$

has the required properties (B.3).

If f is not a circulation, we introduce a new node s and for each node $i \in N$, an arc (s,i) with flow f_{si} equal to the divergence y_i of Eq. (B.1). Then the resulting flow vector is a circulation, and application of the decomposition result just shown for circulations proves the proposition. **Q.E.D.**

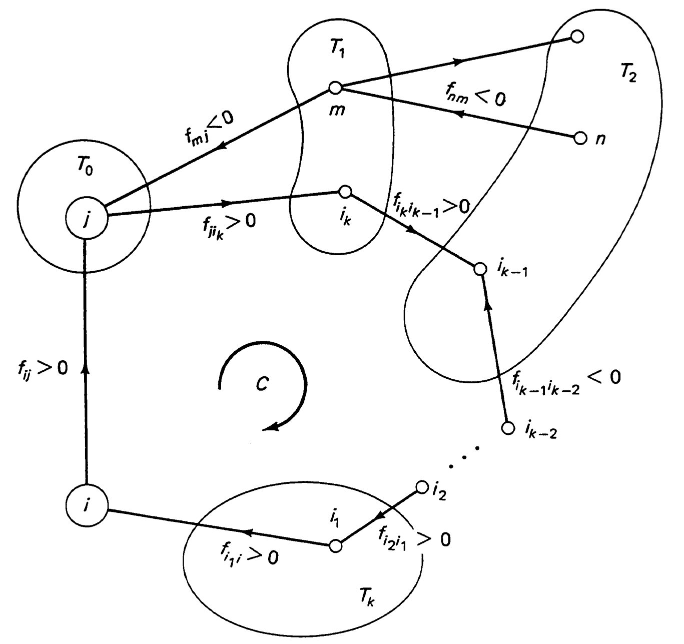


Figure B.1 Construction of a cycle of nonzero flow arcs used in the proof of the Conformal Realization Theorem.

We close this appendix by using the Conformal Realization Theorem to prove a useful fact. We first introduce some definitions. For any positive path $P = (n_1, n_2, \dots, n_k)$, the *multiplicity* of an arc belonging to P is the number of times that it appears in the sequence $(n_1, n_2), (n_2, n_3), \dots, (n_{k-1}, n_k)$. We say that the positive path P is *decomposed* into a set of positive paths $P_1, P_2, \dots, P_d$ if:

(a) an arc belongs to P if and only if it belongs to at least one of the paths $P_1, P_2, \dots, P_d$;

(b) the multiplicity of an arc belonging to P is equal to the sum of the multiplicities of the arc in those paths $P_1, P_2, \dots, P_d$ to which it belongs.

We have the following result:

Path Decomposition Theorem. A positive path P can be decomposed into a (possibly empty) collection of simple positive cycles, together with a simple positive path $\bar{P}$ that has the same start node and end node as P.

Proof. For every arc (i, j) that belongs to P, let f_{ij} be equal to its multiplicity, and for every other arc (i, j) let $f_{ij} = 0$. Assume first that P is a cycle. Then, for each node i, the number of arcs that are outgoing from i and belong to P is equal to the number of arcs that are incoming to i and belong to P. Hence, the flow vector f is a circulation, and the result follows by applying the Conformal Realization Theorem. If P is not a cycle, introduce a new node s and the arcs (s, i_1) and (i_k, s) where i_1 and i_k are the start node and end node of P, respectively. Let $f_{si_1} = f_{i_k s} = 1$. Then the flow vector f is a circulation in the expanded graph. By applying the Conformal Realization Theorem, we see that f is decomposed into a collection of simple positive cycles, of which only one contains the arcs (s, i_1) and (i_k, s). This latter cycle is used to determine the simple positive path $\bar{P}$ that has the same start and end node as P. **Q.E.D.**

C

Duality Theory

In this appendix, we develop a Lagrange multiplier theorem and an associated duality theorem for convex optimization problems with linear constraints. For additional material on duality, consult [Roc70], [Roc84], and [StW70]. Our line of development is based on a simple result known as Farkas' Lemma. To understand this lemma, we introduce some notions related to cones in $\Re^n$.

Definition C.1. A set $C \subset \Re^n$ is said to be a *cone* if $ax \in C$ for all $a \geq 0$ and $x \in C$.

Definition C.2. The *polar cone* of a cone C is the cone given by

$$C^{\perp} = \{y \mid y'z \leq 0,\ \forall\, x \in C\}. \tag{C.1}$$

Figure C.1 illustrates these definitions.

To develop an example of a cone that is particularly interesting to us, let $e_1, e_2, \ldots, e_m$ and $a_1, a_2, \ldots, a_r$ be given vectors in $\Re^n$. Then it can be seen that the set

$$C = \left\{ x \,\middle|\, x = \sum_{i=1}^{m} p_i e_i + \sum_{j=1}^{r} u_j a_j,\ p_i \in \Re,\ u_j \geq 0 \right\} \tag{C.2}$$

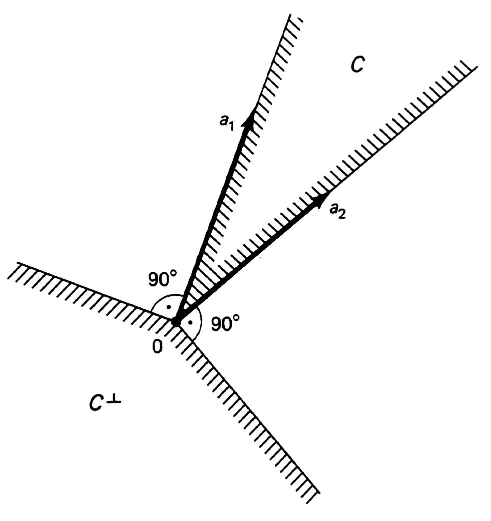


Figure C.1 Illustration of a cone and its polar in $\Re^2$. Here $C = \{x \mid x = u_1 a_1 + u_2 a_2,\ u_1 \geq 0,\ u_2 \geq 0\}$ and $C^{\perp} = \{y \mid y'a_1 \leq 0,\ y'a_2 \leq 0\}$, where a_1 and a_2 are the vectors shown; compare with Eqs. (C.2) and (C.3).

is a closed cone with a polar cone given by

$$C^{\perp} = \{y \mid y'e_i = 0,\ y'a_j \leq 0,\ \forall\, i = 1,\dots,m,\ j = 1,\dots,r\}. \tag{C.3}$$

The following result, when specialized to the cones C and $C^{\perp}$ of Eqs. (C.2) and (C.3), respectively, yields what is usually referred to as Farkas' lemma [PaS82].

Polar Cone Theorem. For any closed convex cone C, we have $(C^{\perp})^{\perp} = C$.

Proof. See Fig. C.2.

Consider now the optimization problem

$$\begin{aligned} \text{minimize} \quad & F(x) \\ \text{subject to} \quad & e_i'x = s_i, \qquad i = 1,\dots,m, \\ & a_j'x \leq t_j, \qquad j = 1,\dots,r, \\ & x \in P, \end{aligned} \tag{P}$$

where $F : \Re^n \mapsto \Re$ is a convex function, e_i and a_j are given vectors in $\Re^n$, s_i and t_j are given scalars, and P is a nonempty polyhedral subset of $\Re^n$. (A polyhedral set P is one that is specified by a finite collection of linear inequalities; we admit the possibility that $P = \Re^n$.) A vector x satisfying the constraints of problem (P) will be referred to as *primal feasible* (or simply *feasible*). Define the *Lagrangian* function

$$L(x, p, u) = F(x) + \sum_{i=1}^{m} p_i(e_i'x - s_i) + \sum_{j=1}^{r} u_j(a_j'x - t_j), \tag{C.4}$$

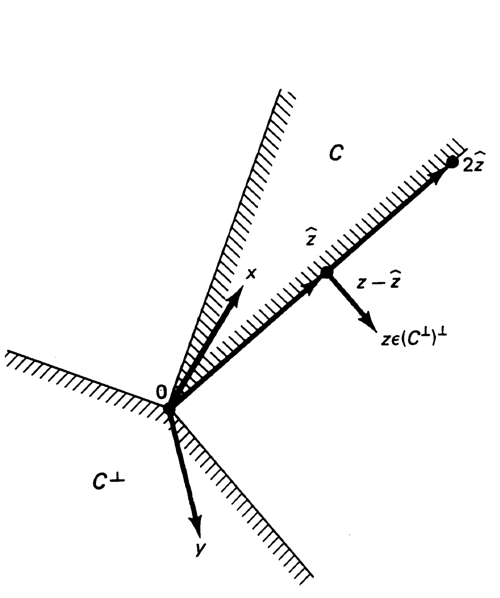


Figure C.2 Proof of the Polar Cone Theorem. If $x \in C$, then for all $y \in C^{\perp}$, we have $x'y \leq 0$, which implies that $x \in (C^{\perp})^{\perp}$. Hence, $C \subset (C^{\perp})^{\perp}$. To prove the reverse inclusion, take $z \in (C^{\perp})^{\perp}$, and let $\hat{z}$ be the unique projection of z on C, as shown in the figure. The projection exists because C is closed; see Prop. 3.2 in Section 3.3, which also implies that

$$(z - \hat{z})'(x - \hat{z}) \leq 0, \qquad \forall\, x \in C.$$

By taking $x = 0$ and $x = 2\hat{z}$ in the preceding relation, it is seen that

$$(z - \hat{z})'\hat{z} = 0.$$

Combining the last two relations, we obtain $(z - \hat{z})'x \leq 0$ for all $x \in C$. Therefore, $(z - \hat{z}) \in C^{\perp}$, and since $z \in (C^{\perp})^{\perp}$, we obtain $(z - \hat{z})'z \leq 0$, which when added to $(z - \hat{z})'\hat{z} = 0$ yields $\|z - \hat{z}\|_2^2 \leq 0$. Therefore, $z = \hat{z}$ and $z \in C$. It follows that $(C^{\perp})^{\perp} \subset C$.

where p and u are the vectors $(p_1, \ldots, p_m)$ and $(u_1, \ldots, u_r)$, respectively. Consider also the *dual functional* $q : \Re^{m+r} \mapsto [-\infty, \infty)$ defined by

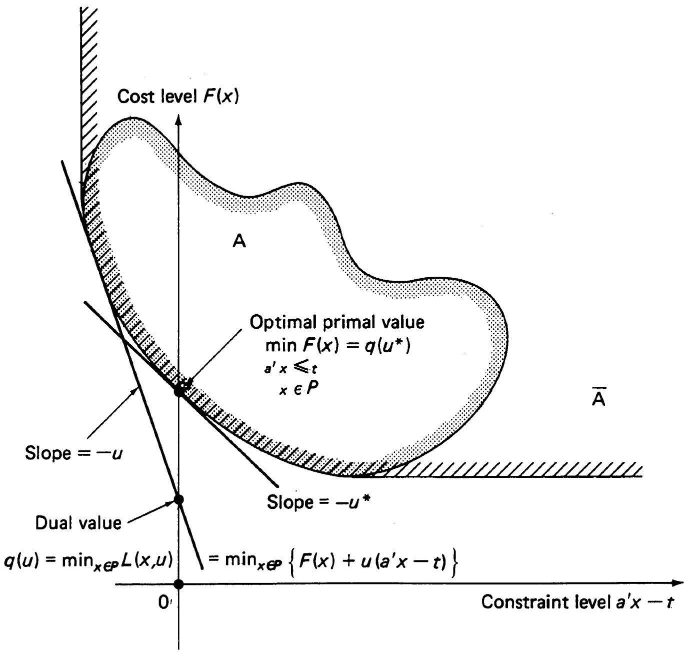
$$q(p, u) = \inf_{x \in P} L(x, p, u). \tag{C.5}$$

The *dual problem* is

$$\begin{aligned} &\text{maximize} \quad q(p, u) \\ &\text{subject to} \quad p \in \Re^m, \quad u \in \Re^r, \quad u \geq 0. \end{aligned} \tag{D}$$

Note that if the polyhedron P is bounded, then the dual functional takes real values, but in general, $q(p, u)$ can take the value $-\infty$. Thus we generally view q as an extended real valued function. A pair (p, u) satisfying the constraints of problem (D) will be referred to as *dual feasible* (or simply *feasible*). From Eqs. (C.4) and (C.5) we see that q is obtained as the infimum of a collection of linear functions of p and u (one for each $x \in P$). It follows that q is *concave* [Prop. A.35(d) in Appendix A]. Figure C.3 provides a geometric interpretation of the dual functional, and illustrates why the optimal value of the original problem (P) is "normally" equal to the optimal dual value.

The following theorems hold for general convex functions $F : \Re^n \mapsto \Re$, but will be shown under the additional assumption that F is differentiable. The proof without this assumption requires a more refined version of Farkas' lemma (see [Roc70], pp. 187 and 277). These theorems will be proved in Section 5.5 without assuming differentiability



Cost level $F(x)$
A
Optimal primal value
$\min_{\substack{a'x\le t\\x\in P}} F(x) = q(u^*)$
$\bar{A}$
Slope $= -u$
Slope $= -u^*$
Dual value
$q(u) = \min_{x\in P} L(x,u)$
$= \min_{x\in P}\{F(x) + u(a'x - t)\}$
0
Constraint level $a'x - t$

Figure C.3 Geometrical interpretation of the dual functional for the case of the single inequality constraint problem

$$\text{minimize} \quad F(x)$$

$$\text{subject to} \quad a'x \le t, \qquad x \in P.$$

We construct the set of constraint–cost pairs $A = \{(a'x - t, F(x)) \mid x \in P\}$. Given $u \ge 0$, the dual value $q(u)$ is obtained by "supporting" A from below with a line of slope $-u$. The point where this line intercepts the vertical axis is $q(u)$. As u varies, $q(u)$ is always below the optimal primal value and for a particular value u^*, $q(u^*)$ equals the optimal primal value. This happens because the convexity of F can be used to show that the set $\bar{A} = \{(z,y) \mid a'x - t \le z,\ F(x) \le y \text{ for some } x \in P\}$ shown in the figure is convex.

of F in the special case of a network flow problem with a separable cost function. They also have extensions to more general convex optimization problems, where the inequality constraints are specified by convex nonlinear functions and the constraint set P is a convex set which is not necessarily polyhedral (see [Roc70] and [StW70]).

Lagrange Multiplier Theorem. A vector x^* is an optimal solution of problem (P) if and only if x^* is feasible and there exist vectors $p^* = (p_1^*, \dots, p_m^*)$ and $u^* = (u_1^*, \dots, u_r^*)$ with $u^* \ge 0$ such that

$$F(x^*) = L(x^*, p^*, u^*) = \min_{x \in P} L(x, p^*, u^*), \tag{C.6}$$

$$u_j^* = 0, \qquad \forall\, j \text{ such that } a_j' x^* < t_j. \tag{C.7}$$

Proof. (Assuming that F is differentiable.) Let x^* be feasible. If there exist p^* and u^* with the given property, we have for all x that are feasible for (P),

$$F(x^*) \le L(x, p^*, u^*) = F(x) + \sum_{i=1}^{m} p_i^*(e_i'x - s_i) + \sum_{j=1}^{r} u_j^*(a_j'x - t_j) \le F(x), \tag{C.8}$$

where the last inequality follows from the condition $u^* \ge 0$ and the feasibility of x. Hence, x^* is optimal for (P).

Conversely, assume that x^* is optimal for (P). We will first prove the version of the result where $P = \Re^n$ or, equivalently, where the linear inequalities defining P are lumped together with the inequality constraints $a_j'x \le t_j$, $j = 1, \dots, r$. We will then use the result for this special case together with the just shown forward part of the theorem to prove the result for the general case.

The optimality of x^* implies that for every feasible z, the rate of change of F starting from x^* and going toward z is nonnegative, that is,

$$\nabla F(x^*)'(z - x^*) \ge 0, \qquad \forall \text{ feasible } z, \tag{C.9}$$

(cf. Prop. 3.1 in Section 3.3). Consider a representation of the polyhedral set P as

$$P = \{x \mid a_j'x \le t_j,\ j = r+1, \dots, \bar{r}\},$$

where a_j and t_j, $j = r+1, \dots, \bar{r}$, are some vectors and scalars, respectively. Let J be the set of indices for which the corresponding inequalities $a_j'x \le t_j$ hold as equalities at x^*, that is,

$$J = \{j \mid a_j'x^* = t_j,\ j = 1, \dots, \bar{r}\}.$$

Consider the cone

$$C = \left\{ y \,\middle|\, y = \sum_{i=1}^{m} p_i e_i + \sum_{j \in J} u_j a_j,\ p_i \in \Re,\ u_j \ge 0 \right\}, \tag{C.10}$$

and its polar given by [cf. Eqs. (C.2) and (C.3)]

$$C^\perp = \{y \mid y'e_i = 0,\ y'a_j \le 0,\ \forall\, i = 1, \dots, m,\ j \in J\}.$$

It is seen that we have $y \in C^\perp$ if and only if $y = \gamma(z - x^*)$ for some $\gamma > 0$ and some feasible z. Thus, from the condition $\nabla F(x^*)'(z - x^*) \ge 0$ [cf. Eq. (C.9)] we obtain

$$\nabla F(x^*)'y \ge 0, \qquad \forall\, y \in C^\perp.$$

Hence, $-\nabla F(x^*)$ belongs to $(C^{\perp})^{\perp}$, which, by the Polar Cone Theorem, is equal to C. It follows from the representation of C [cf. Eq. (C.10)] that there exist p^* and $w^* \geq 0$ such that $w_j^* = 0$ for all $j \notin J$ and

$$\nabla F(x^*) + \sum_{i=1}^{m} p_i^* e_i + \sum_{j=1}^{r} w_j^* a_j + \sum_{j=r+1}^{\bar{r}} w_j^* a_j = 0. \tag{C.11}$$

Define $u^* = (w_1^*, \ldots, w_r^*)$ and consider the problem

$$\begin{aligned} &\text{minimize} \quad L(x, p^*, u^*) \\ &\text{subject to} \quad a_j' x \leq t_j, \qquad j = r+1, \ldots, \bar{r}. \end{aligned}$$

The function $L(x, p^*, u^*) + \sum_{j=r+1}^{\bar{r}} w_j^*(a_j'x - t_j)$ is the Lagrangian function for this problem and is convex with respect to x. From Eq. (C.11), it follows that x^* minimizes this function over $\Re^n$. By using the earlier shown forward part of the theorem, we obtain that x^* solves the above problem, that is, minimizes $L(x, p^*, u^*)$ over all $x \in P$. Using the properties of u^* and the feasibility of x^* we see that (C.7) holds and that $F(x^*) = L(x^*, p^*, u^*)$. **Q.E.D.**

Duality Theorem.

(a) If the primal problem (P) has an optimal solution, the dual problem (D) also has an optimal solution and the two optimal values are equal.

(b) In order for x^* to be an optimal primal solution and (p^*, u^*) to be an optimal dual solution it is necessary and sufficient that x^* be primal feasible, (p^*, u^*) be dual feasible, and

$$F(x^*) = L(x^*, p^*, u^*) = \min_{x \in P} L(x, p^*, u^*).$$

Proof.

(a) We have, using the definitions of the Lagrangian and the dual functions [cf. Eqs. (C.4) and (C.5)]

$$q(p, u) \leq L(x, p, u) \leq F(x), \qquad \forall \text{ primal feasible } x, \text{ and dual feasible } (p, u). \tag{C.12}$$

Furthermore, the Lagrange Multiplier Theorem implies that if x^* is a primal optimal solution, then there exist dual feasible p^* and u^* such that $q(p^*, u^*) = F(x^*)$. From Eq. (C.12), it follows that (p^*, u^*) is dual optimal and that the primal and dual optimal values are equal.

(b) If x^* is primal optimal and (p^*, u^*) is dual optimal, then using the equality of the optimal primal and dual values [part (a)], Eq. (C.12), and the definition of the dual functional, we obtain

$$F(x^*) = L(x^*, p^*, u^*) = q(p^*, u^*) = \min_{x \in P} L(x, p^*, u^*).$$

Conversely, the relation $F(x^*) = \min_{x \in P} L(x, p^*, u^*)$ can be written as $F(x^*) = q(p^*, u^*)$, and, since x^* is primal feasible and (p^*, u^*) is dual feasible, Eq. (C.12) implies that x^* is primal optimal and (p^*, u^*) is dual optimal. **Q.E.D.**

Saddle Point Theorem. In order for x^* to be an optimal primal solution and (p^*, u^*) to be an optimal dual solution, it is necessary and sufficient that $x^* \in P$, $u^* \geq 0$, and

$$L(x^*, p, u) \leq L(x^*, p^*, u^*) \leq L(x, p^*, u^*), \qquad \forall\, x \in P,\ p \in \Re^m,\ u \geq 0. \tag{C.13}$$

Proof. If x^* is primal optimal and (p^*, u^*) is dual optimal, then $x^* \in P$ and $u^* \geq 0$. Furthermore, from part (b) of the Duality Theorem, we obtain

$$L(x^*, p^*, u^*) = \min_{x \in P} L(x, p^*, u^*),$$

thereby proving the right–hand side of Eq. (C.13). We also have for all p and $u \geq 0$, using Eq. (C.12) and part (b) of the Duality Theorem,

$$L(x^*, p, u) \leq F(x^*) = L(x^*, p^*, u^*),$$

which shows the left–hand side of Eq. (C.13).

Conversely, assume that $x^* \in P$, $u^* \geq 0$, and Eq. (C.13) holds. It is seen from the definition of the Lagrangian function (C.4) that

$$\sup_{u \geq 0, p} L(x^*, p, u) = \begin{cases} F(x^*), & \text{if } e_i' x^* = s_i,\ \forall\, i,\ a_j' x^* \leq t_j,\ \forall\, j, \\ +\infty, & \text{otherwise.} \end{cases}$$

Therefore, from Eq. (C.13), we obtain that x^* is primal feasible and

$$F(x^*) = L(x^*, p^*, u^*) = \min_{x \in P} L(x, p^*, u^*).$$

The primal optimality of x^* and the dual optimality of (p^*, u^*) follow from part (b) of the Duality Theorem. **Q.E.D.**

Examples of Dual Problems

As an example of application of the preceding theorems, consider the linear program

$$\begin{aligned} &\text{minimize} && a'x \\ &\text{subject to} && e_i' x = s_i, \qquad i = 1, \ldots, m, \\ & && b_j \leq x_j \leq c_j, \qquad j = 1, \ldots, n, \end{aligned} \tag{LP}$$

where a and e_i are given vectors in $\Re^n$, and s_i are given scalars. The jth component of the vector e_i is denoted by e_{ij}. Using the definition of the dual functional q [cf. Eqs. (C.4) and (C.5)], we obtain

$$q(p) = \min_{b_j \le x_j \le c_j,\ j=1,\ldots,n} \left\{ \sum_{j=1}^{n} \left(a_j + \sum_{i=1}^{m} p_i e_{ij} \right) x_j - \sum_{i=1}^{m} p_i s_i \right\}.$$

This minimization can be carried out separately for each x_j, leading to the form

$$q(p) = \sum_{j=1}^{n} q_j(p) - \sum_{i=1}^{m} p_i s_i, \tag{C.14}$$

where

$$q_j(p) = \begin{cases} \left(a_j + \sum_{i=1}^{m} p_i e_{ij}\right) b_j, & \text{if } a_j + \sum_{i=1}^{m} p_i e_{ij} \ge 0, \\ \left(a_j + \sum_{i=1}^{m} p_i e_{ij}\right) c_j, & \text{if } a_j + \sum_{i=1}^{m} p_i e_{ij} < 0. \end{cases} \tag{C.15}$$

The dual problem is

$$\begin{aligned} &\text{maximize} && q(p) \\ &\text{subject to} && p \in \Re^m. \end{aligned} \tag{DLP}$$

From part (b) of the Duality Theorem, we also obtain that x^* and p^* are primal and dual optimal, respectively, if and only if

$$e_i' x^* = s_i, \qquad \forall\, i = 1, \ldots, m, \tag{C.16}$$

and x_j^* minimizes $\left(a_j + \sum_{i=1}^{m} p_i^* e_{ij}\right) x_j$ subject to $b_j \le x_j \le c_j$ for each j. The latter minimizing property of x_j^* is equivalent to

$$x_j^* = b_j, \qquad \text{if } a_j + \sum_{i=1}^{m} p_i^* e_{ij} > 0, \tag{C.17a}$$

$$x_j^* = c_j, \qquad \text{if } a_j + \sum_{i=1}^{m} p_i^* e_{ij} < 0, \tag{C.17b}$$

$$b_j \le x_j^* \le c_j, \qquad \text{if } a_j + \sum_{i=1}^{m} p_i^* e_{ij} = 0. \tag{C.17c}$$

The relations (C.17) are known as the *complementary slackness conditions.*

Consider also the linear program

$$\begin{aligned} &\text{minimize} \quad a'x \\ &\text{subject to} \quad e_i'x = s_i, \qquad i = 1, \ldots, m, \\ &\qquad\qquad\quad 0 \leq x_j, \qquad j = 1, \ldots, n, \end{aligned} \tag{LP$'$}$$

where a, e_i, and s_i are as in the previous linear program (LP). Using the definition of the dual functional q of Eqs. (C.4) and (C.5), we obtain

$$q(p) = \inf_{0 \leq x_j,\, j=1,\ldots,n} \left\{ \sum_{j=1}^{n} \left(a_j + \sum_{i=1}^{m} p_i e_{ij} \right) x_j - \sum_{i=1}^{m} p_i s_i \right\},$$

so $q(p) = -\infty$ if $a_j + \sum_{i=1}^{m} p_i e_{ij} < 0$ for some j, and $q(p) = -\sum_{i=1}^{m} p_i s_i$ otherwise. Thus, the dual problem is

$$\begin{aligned} &\text{maximize} \quad -\sum_{i=1}^{m} p_i s_i \\ &\text{subject to} \quad a_j + \sum_{i=1}^{m} p_i e_{ij} \geq 0, \qquad \forall\, j = 1, \ldots, n. \end{aligned}$$

By making the change of variables $\pi_i = -p_i$, this problem is written in the following form, which is the one usually encountered in linear programming textbooks,

$$\begin{aligned} &\text{maximize} \quad \sum_{i=1}^{m} \pi_i s_i \\ &\text{subject to} \quad \sum_{i=1}^{m} \pi_i e_{ij} \leq a_j, \qquad \forall\, j = 1, \ldots, n. \end{aligned} \tag{DLP$'$}$$

As an example of a special case of the linear program (LP$'$), consider the following version of the assignment problem [see Section 5.3, Eq. (3.6)]

$$\begin{aligned} &\text{maximize} \quad \sum_{i=1}^{n} \sum_{j \in A(i)} a_{ij} f_{ij} \\ &\text{subject to} \quad \sum_{j \in A(i)} f_{ij} = 1, \qquad \forall\, i = 1, \ldots, n, \\ &\qquad\qquad\quad \sum_{\{i \mid j \in A(i)\}} f_{ij} = 1, \qquad \forall\, j = 1, \ldots, n, \\ &\qquad\qquad\quad 0 \leq f_{ij}, \qquad \forall\, i = 1, \ldots, n, \; j \in A(i), \end{aligned} \tag{AP}$$

where each $A(i)$ is a subset of $\{1,\dots,n\}$. This problem becomes a special case of (LP$'$) once the sign of a_{ij} is reversed and maximization is replaced by minimization. By carrying out this transformation, writing the corresponding dual problem (DLP$'$), and changing sign of the dual variables while replacing minimization by maximization, we obtain the dual problem

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{n} r_i + \sum_{j=1}^{n} p_j \\ &\text{subject to} \quad r_i + p_j \geq a_{ij}, \qquad \forall\ i,\ j \in A(i). \end{aligned} \tag{DAP}$$

Here the dual variables r_i (respectively, p_j) correspond to the first (respectively, second) set of equality constraints of the assignment problem (AP).

As another example of a duality relation, consider the quadratic programming problem

$$\begin{aligned} &\text{minimize} \quad \tfrac{1}{2}x'Qx - b'x \\ &\text{subject to} \quad Ax \leq c, \end{aligned} \tag{QP}$$

where Q is a given $n \times n$ positive definite symmetric matrix, A is a given $m \times n$ matrix, and $b \in \Re^n$ and $c \in \Re^m$ are given vectors. The dual functional is

$$q(u) = \inf_x \left\{ \tfrac{1}{2}x'Qx - b'x + u'(Ax - c) \right\}.$$

The infimum is attained for $x = Q^{-1}(b - A'u)$, and, after substitution of this expression in the preceding relation for q, a straightforward calculation yields

$$q(u) = -\tfrac{1}{2}u'AQ^{-1}A'u - u'(c - AQ^{-1}b) - \tfrac{1}{2}b'Q^{-1}b.$$

The dual problem, after a sign change that converts it to a minimization problem, is equivalent to

$$\begin{aligned} &\text{minimize} \quad \tfrac{1}{2}u'Pu + r'u \\ &\text{subject to} \quad u \geq 0, \end{aligned} \tag{DQP}$$

where

$$P = AQ^{-1}A', \qquad r = c - AQ^{-1}b.$$

If u^* is any optimal solution of the dual problem (DQP), then the optimal solution of the primal quadratic programming problem (QP) is given by $x^* = Q^{-1}(b - A'u^*)$.

Differentiability Properties of the Dual Functional

We finally consider the question of differentiability of the dual function q of Eq. (C.5) when P is a bounded polyhedron. Since $L(\cdot, p, u)$ is convex, it is continuous (Prop. A.36 in Appendix A) and it follows that the infimum of $L(x, p, u)$ over $x \in P$ is attained for all (p, u) by the Weierstrass theorem (Prop. A.8 in Appendix A). Thus, the concave function q is everywhere finite and is therefore continuous (Prop. A.36 in Appendix A). Situations where q is differentiable can be characterized by using Danskin's theorem (Prop. A.43 in Appendix A). This theorem applies to functions of the form $\max_{z \in Z} \phi(y, z)$ where ϕ is convex with respect to y for every fixed z. Equivalently, by changing the sign of ϕ, the theorem applies to functions of the form $\min_{z \in Z} \psi(y, z)$, where ψ is concave with respect to y for every fixed z. The dual functional is of the latter form, provided that we identify ψ, y, z, and Z with L, (p, u), x, and P respectively, and we notice that L is linear (and hence concave) with respect to (p, u) for each fixed x. Since P is compact and L is continuous, it follows from Danskin's theorem that if the infimum of $L(x, p, u)$ over $x \in P$ is attained at a unique point, then q is differentiable at (p, u). Thus, by using also the fact that a convex differentiable function has a continuous gradient (Prop. A.42 in Appendix A), we obtain the following result:

Dual Function Differentiability Theorem. Assume that F is a strictly convex function and that the polyhedron P is bounded. Then the dual functional q is continuously differentiable and for all (p, u), we have

$$\frac{\partial q(p, u)}{\partial p_i} = e_i' \bar{x} - s_i, \qquad i = 1, \ldots, m,$$

$$\frac{\partial q(p, u)}{\partial u_j} = a_j' \bar{x} - t_j, \qquad j = 1, \ldots, r,$$

where $\bar{x}$ is the unique vector that minimizes $L(x, p, u)$ over $x \in P$ for the given pair (p, u).

D

Probability Theory and Markov Chains

We now present some background material on probability theory. We start with two results on exponentially distributed random variables needed for Exercise 4.1 in Section 1.4. We continue with the basic definitions and properties of finite state Markov chains. Finally, we present some more advanced results that are needed for Section 7.8. It is assumed that the reader is already familiar with the basic concepts of probability theory, such as random variables, independence, expectations, and conditional expectations. A rigorous proof of some of the results to be presented requires measure theory ([Ash72] and [Rud74]). Still, knowledge of measure theory is not needed for understanding the contents of this appendix. For more information, the reader could consult [Ash70], [Fel68], and [WoH85] for probability theory, and [Ash70] and [Ros83a] for Markov chains.

We use the notation Pr(A) to denote the probability of an event A. A piecewise continuous function $p : \Re \mapsto [0, \infty)$ is called the *density* of a real valued random variable X if

$$\Pr\big(X \in [a, b]\big) = \int_a^b p(x)\, dx,$$

for every $a, b \in \Re$ such that $a < b$. The *expectation* of a real valued random variable X with density p is given by

$$E[X] = \int_{-\infty}^{\infty} xp(x)\, dx,$$

and its *variance* by

$$\text{Var}(X) = E\Big[\big(X - E[X]\big)^2\Big] = \int_{-\infty}^{\infty} \big(x - E[X]\big)^2 p(x)\, dx,$$

provided that the integrals exist. If X is a random variable that takes only nonnegative values, then its expectation satisfies the *Markov inequality*

$$E[X] \geq a\text{Pr}(X \geq a), \tag{D.1}$$

for every scalar a. Suppose now that X is nonnegative and that $\lim_{x\to\infty} x\text{Pr}(X \geq x) = 0$. We notice that the function $x\text{Pr}(X \geq x)$ also vanishes at zero and infinity and its derivative is equal to $\text{Pr}(X \geq x) - xp(x)$. It follows that

$$E[X] = \int_0^{\infty} xp(x)\, dx = \int_0^{\infty} \text{Pr}(X \geq x)\, dx. \tag{D.2}$$

Properties of Exponential Random Variables

A random variable X is said to have an *exponential* distribution, with mean λ, if it has the density

$$p(x) = \begin{cases} 0, & x < 0, \\ \lambda e^{-\lambda x}, & x \geq 0. \end{cases}$$

Notice that if X has an exponential distribution, then

$$\text{Pr}(X \geq x) = \int_x^{\infty} \lambda e^{-\lambda y}\, dy = e^{-\lambda x}.$$

Proposition D.1. Let $\{X_i\}$ be a sequence of independent exponentially distributed random variables, with mean one. Then

$$\ln n \leq E\left[\max_{1\leq i\leq n} X_i\right] \leq 1 + \ln n, \qquad \forall n \geq 1.$$

(Here ln is the natural logarithm.)

Proof. Let $a_n = E\big[\max_{1\le i\le n} X_i\big]$. Then $a_1 = 1$. For $n \ge 2$, we use Eq. (D.2) to obtain

$$\begin{aligned} a_n &= \int_0^\infty \Pr\Big(\max_{1\le i\le n} X_i \ge x\Big)\,dx \\ &= \int_0^\infty \Big(1-(1-e^{-x})^n\Big)\,dx \\ &= \int_0^\infty \Big(1-(1-e^{-x})^{n-1}\Big)\,dx + \int_0^\infty e^{-x}(1-e^{-x})^{n-1}\,dx \\ &= a_{n-1} + \frac{1}{n}(1-e^{-x})^n\Big|_0^\infty = a_{n-1} + \frac{1}{n}. \end{aligned}$$

Thus, $a_n = 1 + (1/2) + \cdots + (1/n)$. We then notice that

$$\ln n = \int_1^n \frac{1}{x}\,dx \le \sum_{i=1}^n \frac{1}{i} \le 1 + \int_1^n \frac{1}{x}\,dx = 1 + \ln n,$$

which proves the desired result. **Q.E.D.**

Proposition D.2. Let $\{X_i\}$ be a sequence of independent exponentially distributed random variables, with mean one. Then there exist positive constants α and K, such that

$$\Pr\left(\sum_{i=1}^n X_i \ge nk\right) \le e^{-\alpha kn}, \tag{D.3}$$

for every positive integer n and any k larger than K.

Proof. Fix some $\beta \in (0,1)$, and let γ be a positive scalar. A direct calculation yields

$$E\left[e^{\beta(X_i-\gamma)}\right] = \int_0^\infty e^{\beta(x-\gamma)}e^{-x}\,dx = e^{-\beta\gamma}\frac{1}{1-\beta}.$$

In particular, we can choose γ sufficiently large so that

$$E\left[e^{\beta(X_i-\gamma)}\right] < 1.$$

Using the independence of the random variables X_i, we obtain

$$E\left[e^{\beta\sum_{i=1}^n (X_i-\gamma)}\right] = \prod_{i=1}^n E\left[e^{\beta(X_i-\gamma)}\right] < 1.$$

Using the Markov inequality (D.1), we obtain

$$e^{\beta mn}\Pr\left(e^{\beta\sum_{i=1}^{n}(X_i-\gamma)} > e^{\beta mn}\right) < 1.$$

This in turn implies that

$$\Pr\left(\sum_{i=1}^{n} X_i \geq n(m+\gamma)\right) = \Pr\left(e^{\beta\sum_{i=1}^{n}(X_i-\gamma)} > e^{\beta mn}\right) < e^{-\beta mn}.$$

Let $k = 2m$ and $K = 2\gamma$. Then if $k \geq K$, we have $m \geq \gamma$ and

$$\begin{aligned}\Pr\left(\sum_{i=1}^{n} X_i \geq nk\right) = \Pr\left(\sum_{i=1}^{n} X_i \geq n2m\right) &\leq \Pr\left(\sum_{i=1}^{n} X_i \geq n(m+\gamma)\right)\\ &< e^{-\beta mn} = e^{-\alpha kn},\end{aligned}$$

where $\alpha = \beta/2$. **Q.E.D.**

Markov Chains

A discrete time, finite state, homogeneous *Markov chain* is a sequence $\{X_k \mid k = 0,1,2,\ldots\}$ of random variables that take values in a finite set (state space) $\{1,\ldots,n\}$ and such that

$$\Pr\Big(X_{k+1} = j \mid X_1,\ldots,X_{k-1},X_k = i\Big) = p_{ij}, \qquad \forall k \geq 0, \tag{D.4}$$

where each p_{ij} is a given nonnegative scalar. In particular, the probability distribution of the next state X_{k+1} depends on the past only through the current state X_k. Furthermore, since the coefficients p_{ij} do not depend on the time index k, the *transition probabilities* $\Pr(X_{k+1} = j \mid X_k = i)$ are independent of k.

The sum over all j of the transition probabilities $\Pr(X_{k+1} = j \mid X_k = i)$ has to be equal to 1. Thus,

$$\sum_{j=1}^{n} p_{ij} = 1, \qquad \forall i.$$

We form an $n \times n$ matrix P with entries p_{ij}. The sum of the entries in any row of P is equal to 1; furthermore, all entries of P are nonnegative. Any square matrix possessing these two properties is called *stochastic.*

Proposition D.3. The product of two stochastic matrices of the same dimensions is a stochastic matrix.

Proof. Let P and Q be $n \times n$ stochastic matrices. Since $P \geq 0$ and $Q \geq 0$, it follows that $PQ \geq 0$. Also, for any i, we have

$$\sum_{j=1}^{n}[PQ]_{ij} = \sum_{j=1}^{n}\sum_{k=1}^{n}[P]_{ik}[Q]_{kj} = \sum_{k=1}^{n}\sum_{j=1}^{n}[P]_{ik}[Q]_{kj}$$
$$= \sum_{k=1}^{n}[P]_{ik}\sum_{j=1}^{n}[Q]_{kj} = \sum_{k=1}^{n}[P]_{ik} = 1,$$

where the last two equalities followed from the assumption that Q and P are stochastic. **Q.E.D.**

As a special case of Prop. D.3, if P is stochastic and t is a positive integer, then P^t is also stochastic and its entries have the following probabilistic interpretation:

$$[P^t]_{ij} = \Pr\bigl(X_t = j \mid X_0 = i\bigr). \tag{D.5}$$

To see this, assume that Eq. (D.5) holds for $t-1$ and use Eq. (D.4) to obtain

$$\Pr\bigl(X_t = j \mid X_0 = i\bigr) = \sum_{k=1}^{n}\Pr\bigl(X_t = j \mid X_{t-1} = k, X_0 = i\bigr)\cdot\Pr\bigl(X_{t-1} = k \mid X_0 = i\bigr)$$
$$= \sum_{k=1}^{n}\Pr\bigl(X_t = j \mid X_{t-1} = k\bigr)[P^{t-1}]_{ik} = \sum_{k=1}^{n}p_{kj}[P^{t-1}]_{ik} = [P^t]_{ij}.$$

Let π be an n–dimensional nonnegative row vector whose entries sum to 1. Such a vector defines a probability distribution for the initial state X_0 by means of the formula $\Pr(X_0 = i) = \pi_i$. Using Eq. (D.5), we obtain

$$[\pi P^t]_i = \sum_{j=1}^{n}\pi_j[P^t]_{ji} = \sum_{j=1}^{n}\Pr\bigl(X_t = i \mid X_0 = j\bigr)\cdot\Pr\bigl(X_0 = j\bigr) = \Pr\bigl(X_t = i\bigr).$$

In particular, if π has the property $\pi P = \pi$, then $\Pr(X_t = i) = \pi_i = \Pr(X_0 = i)$ for all t and all i, and the distribution of X_t does not change with time. Such a π is called an *invariant* or *steady–state distribution* of the Markov chain associated with P.

A useful classification of the states of a Markov chain is obtained by forming a directed graph $G = (N, A)$, with $N = \{1, \ldots, n\}$ and $A = \{(i,j) \mid i \neq j \text{ and } p_{ij} \neq 0\}$. For any state $i \in N$, let R_i be the set consisting of state i and all states $j \neq i$ such that there exists a positive path from i to j (that is, all arcs in the path are forward arcs). A state i is called *transient* if there exists some $j \in N$ such that $j \in R_i$ but $i \notin R_j$. Nontransient states are called *recurrent.*

If $j \in R_i$, we say that i communicates to j. It can be seen that the set of recurrent states can be partitioned into a collection of disjoint sets $C_1, \ldots, C_\ell$ such that any two states in the same set C_i communicate to each other and no two states in distinct sets C_i and C_j, $i \neq j$, communicate. Each one of the sets C_i is called an *ergodic* class (see Fig. D.1 for an illustration). We say that the matrix P is *irreducible* if $j \in R_i$ for every

$i, j \in N$. An equivalent condition is that there are no transient states and there exists a single ergodic class.

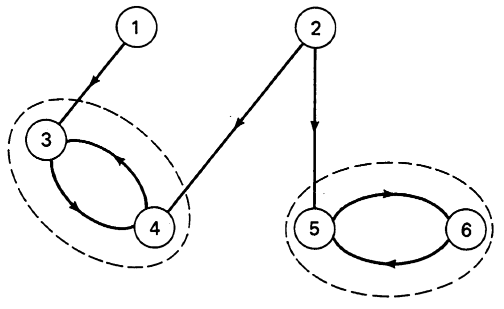


Figure D.1 The transition diagram of a Markov chain. States 1 and 2 are transient. There are two ergodic classes, namely, $\{3,4\}$ and $\{5,6\}$.

Suppose that the Markov chain associated with a stochastic matrix P has m_0 transient states and ℓ ergodic classes, with m_i states in the ith ergodic class C_i. Let us assume that the states have been renumbered so that the transient states have the smallest indices and so that the states in C_i have smaller indices than the states in C_j if $i < j$. Since states in an ergodic class C_i communicate only to other states in C_i, the matrix P has the structure

$$P = \begin{bmatrix} A_{00} & \cdots & \cdots & A_{0\ell} \\ 0 & A_{11} & 0 & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & A_{\ell\ell} \end{bmatrix},$$

where each A_{0i} and A_{ii} is a matrix of size $m_0 \times m_i$ and $m_i \times m_i$, respectively.

A stochastic matrix P is called *periodic* if there exists some $k > 1$ and disjoint nonempty subsets $N_0, \ldots, N_{k-1}$ of the state space N such that if $i \in N_\ell$ and $p_{ij} > 0$ then $j \in N_{\ell+1 (\text{mod } k)}$. We say that P is *aperiodic* if it is not periodic.

Convergence of Random Variables and their Expectations

A sequence $\{X_i\}$ of random variables is said to converge to a random variable X, *with probability one*, if

$$\Pr\Big(\lim_{i\to\infty} X_i = X\Big) = 1.$$

More generally, we say that an event A occurs with probability one ("w.p.1," for short) if $\Pr(A) = 1$. The sequence $\{X_i\}$ converges to X in the *mean square* sense if

$$\lim_{i\to\infty} E\left[(X_i - X)^2\right] = 0.$$

Finally, it converges to X *in probability* if

$$\lim_{i\to\infty} \Pr\big(|X_i - X| \geq \delta\big) = 0,$$

for every $\delta > 0$. Convergence with probability one (respectively, in the mean square sense, in probability) of vector valued random variables is defined by requiring convergence with probability one (respectively, in the mean square sense, in probability) of each component. It is a basic fact that convergence with probability one implies convergence in probability (see, e.g., [Ash72], p.94).

The following result allows us to interchange summation and expectation, under certain assumptions ([Ash72], p.44).

Proposition D.4. (*Lebesgue's Monotone Convergence Theorem*) Let $\{X_i\}$ be a sequence of nonnegative random variables and suppose that $\sum_{i=1}^{\infty} E[X_i] < \infty$. Let $Y_k = \sum_{i=1}^{k} X_i$. Then:

(a) The sequence $\{Y_k\}$ converges, w.p.1, to a finite valued random variable Y.

(b) There holds $E[Y] = \sum_{i=1}^{\infty} E[X_i]$.

Given a finite set $\mathcal{F} = \{X_1, \ldots, X_m\}$ of random variables, we use the notation $E[X \mid \mathcal{F}]$ to denote the conditional expectation of a random variable X, given the random variables $X_1, \ldots, X_m$.[†] The conditional expectation $E[X \mid \mathcal{F}] = E[X \mid X_1, \ldots, X_m]$ is a function of $X_1, \ldots, X_m$ and is therefore itself a random variable. We define $M(\mathcal{F})$ as the set of all random variables X of the form $X = f(X_1, \ldots, X_m)$, where f is an arbitrary function from $\Re^m$ into $\Re$. In particular, $E[X \mid \mathcal{F}] \in M(\mathcal{F})$.[††]

The following result provides some properties of conditional expectations (see, e.g., [Ash72], Section 6.5).

Proposition D.5. Let X, Y, $X_1, \ldots, X_m$, Y_i, $i \geq 0$, be random variables with finite expectations, and let $\mathcal{F} = \{X_1, \ldots, X_m\}$, $\mathcal{G} = \{X_1, \ldots, X_n\}$. Then:

(a) There holds

$$E\big[E[X \mid \mathcal{F}]\big] = E[X]. \tag{D.6}$$

(b) If $n \leq m$ (equivalently, $\mathcal{G} \subset \mathcal{F}$), then

$$E\big[E[X \mid \mathcal{F}] \mid \mathcal{G}\big] = E[X \mid \mathcal{G}], \quad \text{w.p.1.} \tag{D.7}$$

(c) If $Y \in M(\mathcal{F})$, then

$$E[X \cdot Y \mid \mathcal{F}] = Y \cdot E[X \mid \mathcal{F}], \quad \text{w.p.1.} \tag{D.8}$$

[†] The reader familiar with measure theory should interpret $\mathcal{F}$ as the σ–field generated by $X_1, \ldots, X_m$, and $E[X \mid \mathcal{F}]$ becomes the usual notion of conditional expectation given a σ–field [Ash72]. We also point out that conditional expectations can be uniquely defined only on a set of probability 1. (That is, there can exist different versions of $E[X \mid \mathcal{F}]$ which are different with probability zero but which are not always equal.) For this reason, any statements involving conditional expectations are only valid w.p.1.

[††] Actually, only "measurable" functions [Ash72] should be allowed in the definition of $M(\mathcal{F})$, which excludes certain pathological functions. This is not a concern for our purposes.

(d) If $E[|X|] < \infty$, then $E[X \mid \mathcal{F}]$ is well defined and finite, w.p.1.

(e) If each Y_i is nonnegative and $\sum_{i=1}^{\infty} E[Y_i] < \infty$, then

$$E\left[\sum_{i=1}^{\infty} Y_i \;\Big|\; \mathcal{F}\right] = \sum_{i=1}^{\infty} E[Y_i \mid \mathcal{F}], \tag{D.9}$$

where the infinite sums are interpreted as limits with probability 1. Furthermore, both sides of Eq. (D.9) are finite, with probability 1.

The following result can be viewed as a generalization, to a probabilistic context, of the fact that a bounded monotonic sequence converges ([Ash72], Section 7.4).

Proposition D.6. (*Supermartingale Convergence Theorem*) Let $\{Y_i\}$ be a sequence of random variables and let $\{\mathcal{F}_i\}$ be a sequence of finite sets of random variables such that $\mathcal{F}_i \subset \mathcal{F}_{i+1}$ for each i. Suppose that:

(a) Each Y_i is nonnegative and belongs to $M(\mathcal{F}_i)$.

(b) For each i, we have $E[Y_i] < \infty$.

(c) For each i, we have $E[Y_{i+1} \mid \mathcal{F}_i] \leq Y_i$, w.p.1.

Then there exists a nonnegative random variable Y such that the sequence $\{Y_i\}$ converges to Y, w.p.1.

In Section 7.8, we use the following extension of the Supermartingale Convergence Theorem.

Proposition D.7. Let $\{Y_i\}$ and $\{Z_i\}$ be two sequences of random variables. Let $\{\mathcal{F}_i\}$ be a sequence of finite sets of random variables such that $\mathcal{F}_i \subset \mathcal{F}_{i+1}$ for each i. Suppose that:

(a) The random variables Y_i and Z_i are nonnegative, have finite expectations, and belong to $M(\mathcal{F}_i)$ for each i.

(b) There holds

$$E[Y_{i+1} \mid \mathcal{F}_i] \leq Y_i + Z_i, \qquad \forall i, \text{ w.p.1}. \tag{D.10}$$

(c) There holds

$$\sum_{i=1}^{\infty} E[Z_i] < \infty. \tag{D.11}$$

Then there exists a nonnegative random variable Y such that the sequence $\{Y_i\}$ converges to Y, w.p.1.

Proof. Proposition D.5(e) states that $V_k = E\big[\sum_{i=k}^{\infty} Z_i \mid \mathcal{F}_k\big]$ is finite, w.p.1, for every k. We define random variables W_k by

$$W_k = Y_k + V_k = Y_k + E\left[\sum_{i=k}^{\infty} Z_i \;\Big|\; \mathcal{F}_k\right].$$

Then, from Prop. D.5(b) and inequality (D.10),

$$\begin{aligned} E[W_{k+1} \mid \mathcal{F}_k] &= E[Y_{k+1} \mid \mathcal{F}_k] + E\left[E\Big[\sum_{i=k+1}^{\infty} Z_i \;\Big|\; \mathcal{F}_{k+1}\Big] \;\Big|\; \mathcal{F}_k\right] \\ &\le Y_k + Z_k + E\left[\sum_{i=k+1}^{\infty} Z_i \;\Big|\; \mathcal{F}_k\right] = W_k, \qquad \text{w.p.1.} \end{aligned}$$

Notice that each W_k is nonnegative, belongs to $M(\mathcal{F}_k)$, and has finite expectation. Thus, Prop. D.6 applies to the sequence $\{W_k\}$ and shows that it converges, with probability one, to a nonnegative random variable W.

We will now prove that V_k converges to zero, with probability one. This will imply that $Y_k = W_k - V_k$ converges to W, with probability one, and will complete the proof. Using Prop. D.5(b), we have

$$\begin{aligned} E[V_{k+1} \mid \mathcal{F}_k] &= E\left[E\Big[\sum_{i=k+1}^{\infty} Z_i \;\Big|\; \mathcal{F}_{k+1}\Big] \;\Big|\; \mathcal{F}_k\right] = E\left[\sum_{i=k+1}^{\infty} Z_i \;\Big|\; \mathcal{F}_k\right] \\ &= V_k - E[Z_k \mid \mathcal{F}_k] \le V_k, \qquad \text{w.p.1,} \end{aligned}$$

where the last inequality follows from the nonnegativity of Z_k. We apply Prop. D.6 to the sequence $\{V_k\}$ to see that it converges to a nonnegative random variable V, with probability one. We now use the Monotone Convergence Theorem to obtain

$$E[V_k] = E\left[\sum_{i=k}^{\infty} Z_i\right] = \sum_{i=k}^{\infty} E[Z_i],$$

which converges to zero as k tends to infinity, because of Eq. (D.11). Suppose that V is nonzero with positive probability. Then there exist some $\delta > 0$ and $\epsilon > 0$ such that $\Pr(V \ge \delta) \ge \epsilon$. On the other hand, V_k converges to V in probability and this implies that we can find some k_0 such that $\Pr\big(|V_k - V| \ge \delta/2\big) \le \epsilon/2$ for every $k \ge k_0$. Therefore, for $k \ge k_0$,

$$\Pr\Big(V_k \geq \delta/2\Big) \geq \Pr\Big(V \geq \delta \text{ and } |V_k - V| \leq \delta/2\Big)$$
$$\geq \Pr\Big(V \geq \delta\Big) - \Pr\Big(|V_k - V| \geq \delta/2\Big) \geq \epsilon/2.$$

This implies that $E[V_k] \geq (\delta\epsilon)/4$ for every $k \geq k_0$, contradicts the convergence of $E[V_k]$ to zero, and shows that $V = 0$. **Q.E.D.**

References

[AaM76] Aashtiani, H. A., and T. L. Magnanti. 1976. Implementing primal-dual network flow algorithms. Operations Research Center, Working Paper OR-055-76, Massachusetts Institute of Technology, Cambridge.

[AaM81] Aashtiani, H. A., and T. L. Magnanti. 1981. Equilibria on a congested transportation network. *SiAM J. Algeb. & Disc. Math.* 2:213–26.

[ADM82] Ahmed, H. M., J.-M. Delosme, and M. Morf. 1982. Highly concurrent computing structures for matrix arithmetic and signal processing. *Computer* 15:65–82.

[Agm54] Agmon, S. 1954. The relaxation method for linear inequalities. *Can. J. Math.* 6:382–92.

[Ahn79] Ahn, B. H. 1979. *Computation of Market Equilibria for Policy Analysis: The Project Independence Evaluation Study (PIES) Approach.* New York:Garland.

[AhO86] Ahuja, R. K., and J. B. Orlin. 1986. A fast and simple algorithm for the maximum flow problem. Working paper. Sloan School of Management, Massachusetts Institute of Technology, Cambridge.

[AhO87] Ahuja, R. K., and J. B. Orlin. 1987. Private communication.

[AHU74] Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1974. *The Design and Analysis of Computer Algorithms.* Reading, MA: Addison-Wesley.

[AHV85] Andre, F., D. Herman, and J.-P. Verjus. 1985. *Synchronization of Parallel Programs.* Cambridge, MA: The MIT Press.

[Akl85] Akl, S. G. 1985. *Parallel Sorting Algorithms.* Orlando, FL: Academic Press.

[Ame77] Ames, W. F. 1977. *Numerical Methods for Partial Differential Equations.* New York: Academic Press.

[Apt86] Apt, K. R. 1986. Correctness proofs of distributed termination algorithms. *ACM Trans. Prog. Lang. & Syst.* 8:388–405.

[Ash70] Ash, R. B. 1970. *Basic Probability Theory.* New York: Wiley.

[Ash72] Ash, R. B. 1972. *Real Analysis and Probability.* New York: Academic Press.

[AtK84] Atallah, M. J., and S. R. Kosaraju. 1984. Graph problems on a mesh-connected processor array. *J. ACM.* 31:649–67.

[Aus76] Auslender, A. 1976. *Optimization: Methodes Numeriques.* Paris: Mason.

[Avr76] Avriel, M. 1976. *Nonlinear Programming: Analysis and Methods.* Englewood Cliffs, NJ: Prentice-Hall.

[Awe85] Awerbuch, B. 1985. Complexity of network synchronization. *J. ACM.* 32:804–23.

[AwG87] Awerbuch, B., and R. G. Gallager. 1987. A new distributed algorithm to find breadth first search trees. *IEEE Trans. Inf. Theory.* IT-33:315–22.

[BaG87] Barbosa, V. C., and E. M. Gafni. 1987. Concurrency in heavily loaded neighborhood-constrained systems. *Proc. 7th Int. Conf. Distr. Comput. Sys.*

[BaJ77] Bazaraa, M. S., and J. J. Jarvis. 1977. *Linear Programming and Network Flows.* New York: Wiley.

[BaK78] Bachem, A., and B. Korte. 1978. An algorithm for quadratic optimization over transportation polytopes. *Z. Angew. Math. & Mech.* 58:T459–61.

[BaK80] Bachem, A., and B. Korte. 1980. Minimum norm problems over transportation polytopes. *Lin. Algeb. & Appl.* 31:102–18.

[Bar86] Barbosa, V. C. 1986. Concurrency in systems with neighborhood constraints. Doctoral dissertation. Computer Science Dept., UCLA.

[Bau78] Baudet, G. M. 1978. Asynchronous iterative methods for multiprocessors. *J. ACM.* 15:226–44.

[BaW75] Balinski, M., and P. Wolfe (eds.) 1975. *Nondifferentiable Optimization, Math. Prog. Study 3.* Amsterdam: North-Holland.

[BBG77] Bradley, G. H., G. G. Brown, and G. W. Graves. 1977. Design and implementation of large-scale primal transshipment problems. *Manag. Sci.* 24:1–34.

[BBK84] Bojanczyk, A., R. P. Brent, and H. T. Kung. 1984. Numerically stable solution of linear equations using mesh-connected processors. *SiAM J. Sci. & Stat. Comput.* 5:95–104.

[BeC87] Bertsekas, D. P., and D. A. Castanon. 1987. The auction algorithm for transportation problems. Unpublished report.

[BeE87a] Bertsekas, D. P., and J. Eckstein. 1987. Distributed asynchronous relaxation methods for linear network flow problems. *Proc. IFAC '87.*

[BeE87b] Bertsekas, D. P., and D. El Baz. 1987. Distributed asynchronous relaxation methods for convex network flow problems. *SiAM J. Contr. & Optim.* 25:74–85.

[BeE88] Bertsekas, D. P., and J. Eckstein. 1988. Dual coordinate step methods for linear network flow problems. Laboratory for Information and Decision Systems Report LIDS-P-1768, Massachusetts Institute of Technology, Cambridge. *Math. Prog.* Series B. (in press).

[BeG82] Bertsekas, D. P., and E. M. Gafni. 1982. Projection methods for variational inequalities with application to the traffic assignment problem. In D. C. Sorensen and R. J.-B Wets (eds.), *Mathematical Programming Studies*, Volume 17, 139–59. Amsterdam: North-Holland.

[BeG83] Bertsekas, D. P., and E. M. Gafni. 1983. Projected Newton methods and optimization of multicommodity flows. *IEEE Trans. Auto. Contr.* AC-28:1090–6.

[BeG87] Bertsekas, D. P., and R. G. Gallager. 1987. *Data Communication Networks.* Englewood Cliffs, NJ: Prentice-Hall.

[Bel57] Bellman, R. 1957. *Dynamic Programming.* Princeton, NJ: Princeton University Press.

[BeM73] Bertsekas, D. P., and S. K. Mitter. 1973. Descent numerical methods for optimization problems with nondifferentiable cost functions. *SiAM J. Contr.* 11:637–52.

[Ber74] Bertsekas, D. P. 1974. Partial conjugate gradient methods for a class of optimal control problems. *IEEE Trans. Auto. Contr.* 19:209–17.

[Ber75] Bertsekas, D. P. 1975. Necessary and sufficient conditions for a penalty method to be exact. *Math. Prog.* 9:87–99.

[Ber76a] Bertsekas, D. P. 1976. On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Trans. Auto. Contr.* AC-21:174–84.

[Ber76b] Bertsekas, D. P. 1976. Multiplier methods: A survey. *Automatica.* 12:133–45.

[Ber76c] Bertsekas, D. P. 1976. Newton's method for linear optimal control problems. *Proc. IFAC Symp. Large Scale Sys.*, 353–9.

[Ber77] Bertsekas, D. P. 1977. Monotone mappings with application in dynamic programming. *SiAM J. Contr. & Optim.* 15:438–64.

[Ber79a] Bertsekas, D. P. 1979. Convexification procedures and decomposition methods for nonconvex optimization problems. *JOTA.* 29:169–97.

[Ber79b] Bertsekas, D. P. 1979. A distributed algorithm for the assignment problem. Laboratory for Information and Decision Systems, Working Paper, Massachusetts Institute of Technology, Cambridge.

[Ber79c] Bertsekas, D. P. 1979. Algorithms for nonlinear multicommodity network flow problems. In A. Bensoussan and J. L. Lions (eds.), *International Symposium on Systems Optimization and Analysis*, 210–24. New York: Springer-Verlag.

[Ber80] Bertsekas, D. P. 1980. A class of optimal routing algorithms for communication networks. *Proc. 5th Intl. Conf. Comput. Commun.*, 71–6.

[Ber81] Bertsekas, D. P. 1981. A new algorithm for the assignment problem. *Math. Prog.* 21:152–71.

[Ber82a] Bertsekas, D. P. 1982. *Constrained Optimization and Lagrange Multiplier Methods.* New York: Academic Press.

[Ber82b] Bertsekas, D. P. 1982. Projected Newton methods for optimization problems with simple constraints. *SiAM J. Contr. & Optim.* 20:221–46.

[Ber82c] Bertsekas, D. P. 1982. A unified framework for minimum cost network flow problems. Laboratory for Information and Decisions Systems Report LIDS-P-1245-A, Massachusetts Institute of Technology, Cambridge. Also in *Math. Prog.* (1985), 125–45.

[Ber82d] Bertsekas, D. P. 1982. Distributed dynamic programming, *IEEE Trans. Auto. Contr.* AC-27:610–16.

[Ber83] Bertsekas, D. P. 1983. Distributed asynchronous computation of fixed points. *Math. Prog.* 27:107–20.

[Ber85] Bertsekas, D. P. 1985. A distributed asynchronous relaxation algorithm for the assignment problem. *Proc. 24th IEEE Conf. Dec. & Contr.*, 1703–4.

[Ber86a] Bertsekas, D. P. 1986. Distributed asynchronous relaxation methods for linear network flow problems. Laboratory for Information and Decisions Systems Report LIDS-P-1606, revision of Nov. 1986, Massachusetts Institute of Technology, Cambridge.

[Ber86b] Bertsekas, D. P. 1986. Distributed relaxation methods for linear network flow problems. *Proc. 25th IEEE Conf. Dec. & Contr.*, 2101–6.

[Ber87] Bertsekas, D. P. 1987. *Dynamic Programming: Deterministic and Stochastic Models.* Englewood Cliffs, NJ: Prentice-Hall.

[Ber88] Bertsekas, D. P. 1988. The auction algorithm: A distributed relaxation method for the assignment problem. *Ann. Oper. Res.* (in press).

[BeS78] Bertsekas, D. P., and S. E. Shreve. 1978. *Stochastic Optimal Control: The Discrete Time Case.* New York: Academic Press.

[BeT85] Bertsekas, D. P., and P. Tseng. 1985. Relaxation methods for minimum cost ordinary and generalized network flow problems. Laboratory for Information and Decision Systems Report LIDS-P-1462, Massachusetts Institute of Technology, Cambridge. Also in *Oper. Res. J.* (1988) 36:93–114.

[BeT88] Bertsekas, D. P., and P. Tseng. 1988. RELAX: A computer code for minimum cost network flow problems. *Ann. Oper. Res.* 13:125–90.

[BGG84] Bertsekas, D. P., E. M. Gafni, and R. G. Gallager. 1984. Second derivative algorithms for minimum delay distributed routing in networks. *IEEE Trans. Commun.* COM-32:911–19.

[BGV79] Bertsekas, D. P., E. M. Gafni, and K. S. Vastola. 1979. Validation of algorithms for optimal routing of flow in networks. *Proc. 18th IEEE Conf. Dec. & Contr*, 220–227.

[BhI85] Bhatt, S. N., and I. C. F. Ipsen. 1985. How to embed trees in hypercubes. Dept. of Computer Science, Research Report YALEU/DCS/RR-443, Yale University, New Haven, CT.

[BHT87] Bertsekas, D. P., P. Hossein, and P. Tseng. 1987. Relaxation methods for network flow problems with convex arc costs. *SiAM J. Contr. & Optim.* 25:1219–43.

[Bin84] Bini, D. 1984. Parallel solution of certain Toeplitz linear systems. *SiAM J. Comput.* 13:268–76.

[BlJ85] Bland, R. G., and D. L. Jensen. 1985. On the computational behavior of a polynomial-time network flow algorithm. School of Operations Research and Industrial Engineering, Technical Report 661, Cornell University, Ithaca, NY.

[BLS83] Bertsekas, D. P., G. S. Lauer, N. R. Sandell, Jr., and T. A. Posbergh. 1983. Optimal short term scheduling of large-scale power systems. *IEEE Trans. Auto. Contr.* AC-28:1–11.

[BLT76] Bensoussan, A., J. L. Lions, and R. Temam. 1976. Sur les methodes de decomposition, de decentralisation et de coordination et applications. In J. L. Lions and G. I. Marchouk (eds.), *Methodes Numeriques pour les Sciences Physiques at Economiques.* Paris:Dunod.

[Boj84a] Bojanczyk, A. 1984. Complexity of solving linear systems in different models of computation. *SiAM J. Numer. Analy.* 21:591–603.

[Boj84b] Bojanczyk, A. 1984. Optimal asynchronous Newton method for the solution of nonlinear equations. *J. ACM.* 32:792–803.

[Bok81] Bokhari, S. H. 1981. On the mapping problem. *IEEE Trans. Comput.* C-30:207–14.

[BoM75] Borodin, A., and I. Munro. 1975. *The Computational Complexity of Algebraic and Numeric Problems.* New York: American Elsevier.

[Bon83] Bonnans, J. F. 1983. A variant of a projected variable metric method for bound constrained optimization problems. Research Report 242. INRIA, France.

[BoV82] Borkar, V., and P. Varaiya. 1982. Asymptotic agreement in distributed estimation. *IEEE Trans. Auto. Contr.* AC-27:650–5.

[BrH83] Broomell, G., and J. R. Heath. 1983. Classification categories and historical development of circuit switching topologies. *Computing Surveys.* 15:95–134.

[BrL85] Brent, R. P., and F. T. Luk. 1985. The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays. *SiAM J. Sci. Stat. Comput.* 6:69–84.

[BrP67] Browder, F. E., and W. V. Petryshyn. 1967. Construction of fixed points of nonlinear mappings in Hilbert space. *J. Math. Anal. & Appl.* 20:197–228.

[Bra81] Brandt, A. 1981. Muligrid solvers on parallel computers. In M. H. Schultz (ed.), *Elliptic Problem Solvers.* New York: Academic Press.

[Bre67] Bregman, L. M. 1967. The relaxation method for finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Comput. Math. & Math. Phys.* 7:200–17.

[Bre74] Brent, R. P. 1974. The parallel evaluation of general arithmetic expressions. *J. ACM.* 21:201–6.

[Bro70] Brockett, R. W. 1970. *Finite Dimensional Linear Systems.* New York: Wiley.

[BSS88] Byrd, R. H., R. B. Schnabel, and G. A. Shultz. 1988. Parallel quasi-Newton methods for unconstrained optimization. Dept. of Computer Science, Technical Report CU-CS-396-88, University of Colorado-Boulder.

[CaG74] Cantor, D. G., and M. Gerla. 1974. Optimal routing in a packet switched computer network. *IEEE Trans. Comput.* C-23:1062–9.

[CaM87] Calamai, P. H., and J. J. More. 1987. Projected gradient methods for linearly constrained problems. *Math. Prog.* 38:93–116.

[Cap87] Cappello, P. C. 1987. Gaussian elimination on a hypercube automaton. *J. Parallel & Dist. Comput.* 4:288–308.

[CDZ86] Cottle, R. W., S. G. Duvall, and K. Zikan. 1986. A Lagrangean relaxation algorithm for the constrained matrix problem. *Naval Res. Logist. Quart.* 33:55–76.

[CeH87] Censor, Y., and G. T. Herman. 1987. On some optimization techniques in image reconstruction from projections. *Appl. Numer. Math.* 3:365–91.

[Cen81] Censor, Y. 1981. Row-action methods for huge and sparse systems and their applications. *SiAM Rev.* 23:444–91.

[ChC58] Charnes, A., and W. W. Cooper. 1958. Nonlinear network flows and convex programming over incidence matrices. *Naval Res. Logist. Quart.* 5:321–40.

[ChL85] Chandy, K. M., and L. Lamport. 1985. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.* 3:63–75.

[ChM69] Chazan, D., and W. L. Miranker. 1969. Chaotic relaxation. *Lin. Algeb. & Appl.* 2:199–222.

[ChM70] Chazan, D., and W. L. Miranker. 1970. A nongradient and parallel algorithm for unconstrained minimization. *SiAM J. Contr.* 8:207–17.

[ChM81] Chandy, K. M., and J. Misra. 1981. Asynchronous distributed simulation via a sequence of parallel computations. *Commun. ACM.* 24:198–206.

[ChM82] Chandy, K. M., and J. Misra. 1982. Distributed computation on graphs: Shortest path algorithms. *Commun. ACM.* 25:833–7.

[ChM84] Chandy, K. M., and J. Misra. 1984. The drinking philosophers problem. *ACM Trans. Prog. Lang. & Sys.* 6:632–46.

[Chr75] Christofides, N. 1975. *Graph Theory: An Algorithmic Approach.* New York: Academic Press.

[ChS85] Chan, T. F., and R. Schreiber. 1985. Parallel networks for multi-grid algorithms: Architecture and complexity. *SiAM J. Sci. & Stat. Comput.* 6:698–711.

[ChS86] Chan, T. F., and Y. Saad. 1986. Multigrid algorithms on the hypercube multiprocessor. *IEEE Trans. Comput.* C-35:969–77.

[Chv83] Chvatal, V. 1983. *Linear Programming.* New York: W. H. Freeman.

[Cim38] Cimmino, G. 1938. Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari. *Ricerca Sci.* XVI Ser. II, Anno IX, 1:326–33.

[Cof76] Coffman, E. G., Jr. (ed.). 1976. *Computer and Job Shop Scheduling Theory.* Englewood Cliffs, NJ: Prentice-Hall.

[Coh78] Cohen, G. 1978. Optimization by decomposition and coordination: A unified approach. *IEEE Trans. Auto. Contr.* AC-23:222–32.

[CoL55] Coddington, E. A., and N. Levinson. 1955. *Theory of Ordinary Differential Equations.* New York: McGraw-Hill.

[Coo81] Cook, S. A. 1981. Towards a complexity theory of synchronous parallel computation. *Enseigne. Math.* 27:99–124.

[CoP82] Cottle, R. W., and J. S. Pang. 1982. On the convergence of a block successive overrelaxation method for a class of linear complementarity problems. In D.C. Sorensen and R. J.-B. Wets (eds.), *Mathematical Programming Studies,* Volume 17, 126–38. Amsterdam: North-Holland.

[CoR86] Cosnard, M., and Y. Robert. 1986. Complexity of parallel QR factorization. *J. ACM.* 33:712–23.

[Cot84] Cottle, R. W. 1984. Application of a block successive overrelaxation method to a class of constrained matrix problems. In R. W. Cottle, M. L. Kelmanson, and B. Korte (eds.), *Mathematical Programming,* 89–103. Amsterdam: North-Holland.

[CoZ83] Cohen, G., and D. L. Zhu. 1983. Decomposition and coordination methods in large scale optimization problems. The nondifferentiable case and the use of augmented Lagrangians. In J. B. Cruz, Jr. (ed.), *Advances in Large Scale Systems, Theory and Applications,* Vol. 1. Greenwich, CT: JAI Press.

[Cry71] Cryer, C. W. 1971. The solution of a quadratic programming problem using systematic overrelaxation. *SiAM J. Contr.* 9:385–92.

[Csa76] Csanky, L. 1976. Fast parallel matrix inversion algorithms. *SiAM J. Comput.* 5:618–23.

[Cyb87] Cybenko, G. 1987. Dynamic load balancing for distributed memory multiprocessors. Dept. of Computer Science, Technical Report 87-1, Tufts University, Medford, MA.

[Daf71] Dafermos, S. C. 1971. An extended traffic assignment model with applications to two-way traffic. *Trans. Sci.* 5:366–89.

[Daf80] Dafermos, S.C 1980. Traffic equilibrium and variational inequalities. *Trans. Sci.* 14:42–54.

[Daf83] Dafermos, S. C. 1983. An iterative scheme for variational inequalities. *Math. Prog.* 26:40–7.

[Dan63] Dantzig, G. B. 1963. *Linear Programming and Extensions.* Princeton, NJ: Princeton University Press.

[Dan67] Danskin, J. M. 1967. *Theory of Max-Min and Its Application in Weapons Allocation Problems.* New York: Springer-Verlag.

[Dat85] Datta, K. 1985. Parallel complexities and computations of Cholesky's decomposition and QR factorization. *Intl. J. Comput. Math.* 18:67–82.

[Den67] Denardo, E. V. 1967. Contraction mappings in the theory underlying dynamic programming. *SiAM Rev.* 9:165–77.

[DeP84] Deo, N., and C. Pang. 1984. Shortest path algorithms: Taxonomy and annotation. *Networks.* 14:275–323.

[DES82] Dembo, R. S., S. C. Eisenstadt, and T. Steihaug. 1982. Inexact Newton methods. *SiAM J. Numer. Anal.* 19:400–8.

[DeS83] Dennis, J. E., Jr., and R. B. Schnabel. 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Englewood Cliffs, NJ: Prentice-Hall.

[DFV83] Dijkstra, E. W., W. H. J. Feijen, and A. J. M. Van Gasteren. 1983. Derivation of a termination algorithm for distributed computations. *Inf. Proc. Lett.* 15:217–19.

[DGK79] Dial, R., F. Glover, D. Karney, and D. Klingman. 1979. A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. *Networks.* 9:215–48.

[Dij71] Dijkstra, E. W. 1971. Hierarchical ordering of sequential processes. *Acta Informatica.* 1:115–38.

[DiS80] Dijkstra, E. W., and C. S. Sholten. 1980. Termination detection for diffusing computations. *Inf. Proc. Lett.* 11:1–4.

[DNS81] Dekel, E., D. Nassimi, and S. Sahni. 1981. Parallel matrix and graph algorithms. *SiAM J. Comput.* 10:657–73.

[Don71] Donnelly, J. D. P. 1971. Periodic chaotic relaxation. *Lin. Algeb. & Appl.* 4:117–28.

[DoS87] Dongarra, J. J., and D. C. Sorensen. 1987. A fully parallel algorithm for the symmetric eigenvalue problem. *SiAM J. Sci. Stat. Comput.* 8:s139–54.

[DuB82] Dubois, M., and F. A. Briggs. 1982. Performance of synchronized iterative processes in multiprocessor systems. *IEEE Trans. Software Eng.* SE-8:419–31.

[Dun81] Dunn, J. C. 1981. Global and asymptotic convergence rate estimates for a class of projected gradient processes. *SiAM J. Contr. & Optim.* 19:368–400.

[DuS63] Dunford, N., and J. T. Schwartz. 1963. *Linear Operators.* New York: Wiley.

[Eck87] Eckstein, J. 1987. Private communication.

[Eck88] Eckstein, J. 1988. The Lions-Mercier splitting algorithm and the alternating direction method are instances of the proximal point algorithm. Laboratory for Information and Decision Systems Report LIDS-P-1769, Massachusetts Institute of Technology, Cambridge.

[EdK72] Edmonds, J., and R. M. Karp. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM.* 19:248–64.

[Elf80] Elfving, T. 1980. Block-iterative methods for consistent and inconsistent linear equations. *Numer. Math.* 35:1–12.

[ElT82] El Tarazi, M. N. 1982. Some convergence results for asynchronous algorithms. *Numer. Math.* 39:325–40.

[Eph86] Ephremides, A. 1986. The routing problem in computer networks. In I. F. Blake and H. V. Poor (eds.), *Communication and Networks*, 299–325. New York: Springer-Verlag.

[Eri88] Eriksen, O. 1988. A termination detection protocol and its formal verification. *J. Parallel & Distr. Comput.* 5:82–91.

[Eve63] Everett, H. 1963. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Oper. Res.* 11:399–417.

[FaF63] Fadeev, D. K., and V. N. Fadeeva. 1963. *Computational Methods of Linear Algebra*. San Francisco: W. H. Freeman.

[FBB80] Findeisen, W., F. N. Bailey, M. Brdys, K. Malinowski, P. Tatjewski, and A. Wozniak. 1980. *Control and Coordination in Hierarchical Systems*. New York: Wiley.

[Fel68] Feller, W. 1968. *An Introduction to Probability Theory and Its Applications*. New York: Wiley.

[Fin79] Finn, S. G. 1979. Resynch procedures and a failsafe network protocol. *IEEE Trans. Commun.* COM-27:840–6.

[FJL88] Fox, G., M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker. 1988. *Solving Problems on Concurrent Processors*, Vol. 1. Englewood Cliffs, NJ: Prentice-Hall.

[FlN74] Florian, M., and S. Nguyen. 1974. A method for computing network equilibrium with elastic demand. *Trans. Sci.* 8:321–32.

[Fly66] Flynn, M. J. 1966. Very high-speed computing systems. *Proc. IEEE.* 54:1901–9.

[FNP81] Florian, M. S., S. Nguyen, and S. Pallottino. 1981. A dual simplex algorithm for finding all shortest paths. *Networks.* 11:367–78.

[FoF62] Ford, L. R., Jr., and D. R. Fulkerson. 1962. *Flow in Networks*. Princeton, NJ: Princeton University Press.

[FoG83] Fortin, M., and R. Glowinski. 1983. On decomposition-coordination methods using an augmented Lagrangian. In M. Fortin and R. Glowinski (eds.), *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems*. 97–146. Amsterdam: North-Holland.

[FoM67] Forsythe, G. E., and C. B. Moler. 1967. *Computer Solution of Linear Algebraic Systems*. Englewood Cliffs, NJ: Prentice-Hall.

[For56] Ford, L. R., Jr. 1956. Network flow theory. Report P-923, The Rand Corp., Santa Monica, CA.

[FoW78] Fortune, S., and J. Wyllie. 1978. Parallelism in random access machines. *Proc. 10th ACM STOC.*, 114–18.

[Fra80] Francez, N. 1980. Distributed termination. *ACM Trans. Prog. Lang. & Sys.* 2:42–5.

[FrT84] Fredman, M. L., and R. E. Tarjan. 1984. Fibonacci heaps and their uses in improved network optimization algorithms. *Proc. 25th Annual Symp. Found. Comput. Sci.*, 338–46.

[Gab79] Gabay, D. 1979. Methodes numeriques pour l'optimisation non-lineaire. These de Doctorat d'Etat et Sciences Mathematiques, Universite Pierre et Marie Curie (Paris VI).

[GaB81] Gafni, E. M., and D. P. Bertsekas. 1981. Distributed algorithms for generating loopfree routes in networks with frequently changing topology. *IEEE Trans. Commun.* COM-29:11–18.

[Gab83] Gabay, D. 1983. Applications of the method of multipliers to variational inequalities. In M. Fortin and R. Glowinski (eds.), *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems.* 299–331. Amsterdam: North-Holland.

[GaB84] Gafni, E. M., and D. P. Bertsekas. 1984. Two-metric projection methods for constrained optimization. *SiAM J. Contr. & Optim.* 22:936–64.

[GaB86] Gafni, E. M., and V. C. Barbosa. 1986. Optimal snapshots and the maximum flow in precedence graphs. *Proc. 24th Allerton Conf.* 1089–97.

[Gaf79] Gafni, E. M. 1979. Convergence of a routing algorithm. MS thesis. Dept. of Electrical Engineering, University of Illinois-Urbana.

[Gaf86] Gafni, E. M. 1986. Perspectives on distributed network protocols: A case for building blocks. Computer Science Dept., UCLA; MILCOM '86, Monterey, CA.

[GaJ79] Garey, M. R., and D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco: W. H. Freeman.

[Gal68] Gallager, R. G. 1968. *Information Theory and Reliable Communications.* New York: Wiley.

[Gal76] Gallager, R. G. 1976. A shortest path routing algorithm with automatic resynch. Unpublished note.

[Gal77] Gallager, R. G. 1977. A minimum delay routing algorithm using distributed computation. *IEEE Trans. Commun.* COM-23:73–85.

[Gal82] Gallager, R. G. 1982. Distributed minimum hop algorithm. Laboratory for Information and Decision Systems Report LIDS-P-1175, Massachusetts Institute of Technology, Cambridge.

[Gal85] Gallopoulos, E. 1985. Processor arrays for problems in computational physics. PhD thesis. Dept. of Computer Science, University of Illinois-Urbana.

[GaM76] Gabay, D., and B. Mercier. 1976. A dual algorithm for the solution of nonlinear variational problems via finite-element approximations. *Comput. and Math. Appl.* 2:17–40.

[Gan59] Gantmacher, F. R. 1959. *The Theory of Matrices.* New York: Chelsea.

[GaT87] Gabow, H. N., and R. E. Tarjan. 1987. Faster scaling algorithms for graph matching. Unpublished manuscript.

[GaV84] Gannon, D. B., and J. Van Rosendale. 1984. On the impact of communication complexity on the design of parallel numerical algorithms. *IEEE Trans. Comput.* C-32:1180–94.

[Gea86] Gear, C. W. 1986. The potential for parallelism in ordinary differential equations. Dept. of Computer Science, Report No. UIUCDCS-R-86-1246, University of Illinois at Urbana-Champaign; also presented at the Second International Conference of Computational Mathematics, University of Benin, Benin City, Nigeria.

[Gea87] Gear, C. W. 1987. Parallel methods for ordinary differential equations. Dept. of Computer Science, Report No. UIUCDCS-R-87-1369, University of Illinois at Urbana-Champaign.

[GeK81] Gentleman, W. M., and H. T. Kung. 1981. Matrix triangularization by systolic arrays. *Proc. SPIE 298, Real Time Signal Processing IV*, 19–26.

[Gen78] Gentleman, W. M. 1978. Some complexity results for matrix computations on parallel processors. *J. ACM*. 25:112–15.

[Geo72] Geoffrion, A. M. 1972. Generalized Benders decomposition. *JOTA*. 10:237–60.

[GHN87] Geist, G. A., M. T. Heath, and E. Ng. 1987. Parallel algorithms for matrix computations. In L. H. Jamieson, D. B. Gannon, and R. J. Douglass (eds.), *The Characteristics of Parallel Algorithms*. 233–51. Cambridge, MA: The MIT Press.

[GKK74] Glover, F., D. Karney, D. Klingman, and A. Napier. 1974. A computation study on start procedures, basis change criteria, and solution algorithms for transportation problems. *Manage. Sci.* 20:793–819.

[GKK87] Gohberg. I., T. Kailath, I. Koltracht, and P. Lancaster. 1987. Linear complexity algorithms for linear systems of equations with recursive structure. *Lin. Algeb. & Appl.* 88/89:271–315.

[GlL87] Glowinski, R., and P. Le Tallec. 1987. Augmented Lagrangian methods for the solution of variational problems. Mathematics Research Center, Technical Summary Report 2965, University of Wisconsin-Madison.

[GlM75] Glowinski, R., and A. Marrocco. 1975. Sur l'approximation par elements finis d'ordre un et la resolution par penalisation-dualite d'une classe de problemes de Dirichlet non lineaires. *Revue Francaise d'Automatique Informatique Recherche Operationnelle, Analyse numerique*. R-2:41–76.

[GLT81] Glowinski, R., J. L. Lions, and R. Tremolieres. 1981. *Numerical Analysis of Variational Inequalities*. Amsterdam: North-Holland.

[GMW81] Gill, P. E., W. Murray, and M. H. Wright. 1981. *Practical Optimization*. New York: Academic Press.

[Gof80] Goffin, J. L. 1980. The relaxation method for solving problems of linear inequalities. *Math. Oper. Res.* 5:388–414.

[Gol64] Goldstein, A. A. 1964. Convex programming in Hilbert Space. *Bull. Am. Math. Soc.* 70:709–10.

[Gol78] Goldschlager, L. M. 1978. A unified approach to models of synchronous parallel machines. *Proc. 10th ACM STOC.*, 89–94.

[Gol85a] Goldberg, A. V. 1985. A new max-flow algorithm. Laboratory for Computer Science, Tech. Mem. MIT/LCS/TM-291, Massachusetts Institute of Technology, Cambridge.

[Gol85b] Golshtein, E. G. 1985. A decomposition method for linear and convex programming problems. *Ekon. i Mat. Metody* [translated as *Matecon*]. 21:1077–91.

[Gol86a] Golshtein, E. G. 1986. The block method of convex programming. *Sov. Math. Doklady.* 33:584–7.

[Gol86b] Goldberg, A. V. 1986. Solving minimum-cost flow problems by successive approximations. Extended abstract. Submitted to *19th ACM STOC.*

[Gol87a] Goldberg, A. V. 1987. Efficient graph algorithms for sequential and parallel computers. Laboratory for Computer Science, Technical Report TR-374, Massachusetts Institute of Technology, Cambridge.

[Gol87b] Golshtein, E. G. 1987. A general approach to decomposition of optimization systems. *Sov. J. Comput. & Sys. Sci.* 25:105–14 [translated from *Tekhnicheskaya Kibernetika* (1987). 1:59–69].

[GoT86] Goldberg, A. V., and R. E. Tarjan. 1986. A new approach to the maximum flow problem. *Proc. 18th ACM STOC.* 136–46.

[GoT87] Goldberg, A. V., and R. R. Tarjan. 1987. Solving minimum cost flow problems by successive approximation. *Proc. 19th ACM STOC.* 7–18.

[GoV83] Golub, G. H., and C. F. Van Loan. 1983. *Matrix Computations.* Baltimore: The Johns Hopkins University Press.

[Gra71] Grad, J. 1971. Matrix balancing. *Comput. J.* 14:280–4.

[GrH80] Grigoriadis, M. D., and T. Hsu. 1980. The Rutgers minimum cost network flow subroutines. RNET Documentation. Dept. of Computer Science Report, Rutgers University, New Brunswick, NJ.

[GrS81] Grcar, J., and A. Sameh. 1981. On certain parallel Toeplitz linear system solvers. *SiAM J. Sci. & Stat. Comput.* 2:238–56.

[GuP74] Guillemin, V., and A. Pollack. 1974. *Differential Topology.* Englewood Cliffs, NJ: Prentice-Hall.

[HaB70] Haarhoff, P. C., and J. D. Buys. 1970. A new method for the optimization of a nonlinear function subject to nonlinear constraints. *The Comput. J.* 13:178–84.

[Hac85] Hackbush, W. 1985. *Multi-Grid Methods and Applications.* New York: Springer-Verlag.

[HaC87] Hajek, B., and R. L. Cruz. 1987. Delay and routing in interconnection networks. In A. R. Odoni, L. Bianco, and G. Szego (eds.), *Flow Control of Congested Networks.* 235–42. New York: Springer-Verlag.

[HaL88] Han, S. P., and G. Lou. 1988. A parallel algorithm for a class of convex programs. *SiAM J. Contr. & Optim.* 26:345–55.

[Ham86] Hamming, R. W. 1986. *Coding and Information Theory.* Englewood Cliffs, NJ: Prentice-Hall.

[Han86] Han, S. P. 1986. Optimization by updated conjugate subspaces. In D. F. Griffiths and G. A. Watson (eds.), *Numerical Analysis: Pitman Research Notes in Mathematics Series 140*, 82–97. Burnt Mill, England: Longman Scientific and Technical.

[Han88] Han, S. P. 1988. A successive projection method. *Math. Prog.* 40:1–14.

[Har69] Harary, F. 1969. *Graph Theory*. Reading, MA: Addison-Wesley.

[HaY81] Hageman, L. A., and D. M. Young. 1981. *Applied Iterative Methods*. New York: Academic Press.

[Hay84] Hayes, J. F. 1984. *Modeling and Analysis of Computer Communications Networks*. New York: Plenum.

[HaZ87] Haxari, C., and H. Zedan. 1987. A distributed algorithm for distributed termination. *Inf. Proc. Lett.* 24:293–7.

[HeL78] Herman, G. T., and A. Lent. 1978. A family of iterative quadratic optimization algorithms for pairs of inequalities, with application in diagnostic radiology. *Math. Prog. Studies.* 9:15–29.

[Hel74] Heller, D. 1974. On the efficient computation of recurrence relations. Dept. of Computer Science, Technical Report, Carnegie-Mellon University, Pittsburgh, PA.

[Hel76] Heller, D. 1976. Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. *SiAM J. Numer. Anal.* 13:484–96.

[Hel78] Heller, D. 1978. A survey of parallel algorithms in numerical linear algebra. *SiAM Rev.* 20:740–77.

[Hen64] Henrici, P. 1964. *Elements of Numerical Analysis*. New York: Wiley.

[HeS52] Hestenes, M. R., and E. L. Stiefel. 1952. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards Sect. 5.* 49:409–36.

[Hes69] Hestenes, M. R. 1969. Multiplier and gradient methods. *JOTA*. 4:303–20.

[HeS82] Heyman, D. P., and M. J. Sobel. 1982. *Stochastic Models in Operations Research*. New York: McGraw-Hill.

[Hil57] Hildreth, C. 1957. A quadratic programming procedure. *Naval Res. Logist. Quart.* 4:79–85. See also "Erratum," *Naval Res. Logist. Quart.* 4:361.

[Hil74] Hildebrand, F. B. 1974. *Introduction to Numerical Analysis*. New York: McGraw-Hill.

[Hil85] Hillis, W. D. 1985. *The Connection Machine*. Cambridge, MA: The MIT Press.

[HLL78] Herman, G. T., A. Lent, and P. H. Lutz. 1978. Relaxation methods for image reconstruction. *Commun. ACM.* 21:152–8.

[HLN84] Hearn, D. W., S. Lawphongpanich, and S. Nguyen. 1984. Convex programming formulation of the asymmetric traffic assignment problem. *Trans. Res.* 18B:357–65.

[HLV87] Hearn, D. W., S. Lawphongpanich, and J. A. Ventura. 1987. Restricted simplicial decomposition: Computation and extensions. *Math. Programming Studies.* 31:99–118.

[Hoc65] Hockney, R. W. 1965. A fast direct solution of Poisson's equation using Fourier analysis. *J. ACM.* 12:95–113.

[Hoc85] Hockney, R. W. 1985. MIMD computing in the USA—1984. *Parallel Comput.* 2:119–36.

[HoJ81] Hockney, R. W., and C. R. Jesshope. 1981. *Parallel Computers*. Bristol, England: Adam Hilger.

[HoK71] Hoffman, K., and R. Kunze. 1971. *Linear Algebra.* Englewood Cliffs, NJ: Prentice-Hall.

[HoM76] Ho, Y. C., and S. K. Mitter (eds.). 1976. *Directions in Large Scale Systems.* New York: Plenum.

[Hou64] Householder, A. S. 1964. *The Theory of Matrices in Numerical Analysis.* New York: Dover.

[HuS87a] Humblet, P. A., and S. R. Soloway. 1987. A fail-safe layer for distributed network algorithms and changing topologies. Laboratory for Information and Decision Systems Report LIDS-P-1702, Massachusetts Institute of Technology, Cambridge.

[HuS87b] Humblet, P. A., and S. R. Soloway. 1987. Topology broadcast algorithms. Laboratory for Information and Decisions Systems Report LIDS-P-1692, Massachusetts Institute of Technology, Cambridge.

[Hwa84] Hwang, K., (ed.). 1984. *Supercomputers: Design and Applications.* Silver Springs, MD: IEEE Computer Society Press.

[Hwa87] Hwang, K. 1987. Advanced parallel processing with supercomputer architectures. *Proc. IEEE.* 75:1348–79.

[HwB84] Hwang, K., and F. A. Briggs. 1984. *Computer Architecture and Parallel Processing.* New York: McGraw-Hill.

[HyK77] Hyafil, L., and H. T. Kung. 1977. The complexity of parallel evaluation of linear recurrences. *J. ACM.* 24:513–21.

[IEE78] *IEEE Trans. Auto. Contr.* 1978. Special Issue on Large-Scale Systems and Decentralized Control, Vol. AC-23.

[IEE87] *IEEE Computer.* 1987. Special Issue on Interconnection Networks, Vol. 20.

[IpS85] Ipsen, I. C. F., and Y. Saad. 1985. The impact of parallel architectures on the solution of eigenvalue problems. Dept. of Computer Science, Research Report YALEU/DCS/RR-444, Yale University, New Haven, CT.

[IsK66] Isacson, E., and H. B. Keller. 1966. *Analysis of Numerical Methods.* New York: Wiley.

[ISS86] Ipsen, I. C. F., Y. Saad, and M. H. Schultz. 1986. Complexity of dense-linear-system solution on a multiprocessor ring. *Lin. Algeb. & Appl.* 77:205–39.

[IsS86] Israeli, A., and Y. Schiloach. 1986. An improved parallel algorithm for maximal matching. *Inf. Proc. Lett.* 22:57–60.

[JeB80] Jensen, P. A., and J. W. Barnes. 1980. *Network Flow Programming.* New York: Wiley.

[Jef85] Jefferson, D. R. 1985. Virtual time. *ACM Trans. Prog. Lang. & Sys.* 7:404–25.

[Jer79] Jeroslow, R. G. 1979. Some relaxation methods for linear inequalities. *Cahiers du C.E.R.O.* 21:43–53.

[Joh85a] Johnsson, S. L. 1985. Cyclic reduction on a binary tree. *Comput. Phys. Commun.* 37:195–203.

[Joh85b] Johnsson, S. L. 1985. Solving narrow banded systems on ensemble architectures. *ACM Trans. Math. Software.* 11:271–88.

[Joh87a] Johnsson, S. L. 1987. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel & Distr. Comput.* 4:133–72.

[Joh87b] Johnsson, S. L. 1987. Solving tridiagonal systems on ensemble architectures. *SiAM J. Sci. & Stat. Comput.* 8:354–92.

[Kar78] Karp, R. M. 1978. A characterization of the minimum cycle mean in a digraph. *Disc. Math.* 23:309–11.

[KeH80] Kennington, J., and R. Helgason. 1980. *Algorithms for Network Programming.* New York: Wiley.

[KiC87] Kim, T., and K.-T. Chwa. 1987. An O(n log n log log n) parallel maximum matching algorithm for bipartite graphs. *Inf. Proc. Lett.* 24:15–17.

[KiL86] Kindervater, G. A. P., and J. K. Lenstra. 1986. An introduction to parallelism in combinatorial optimization. *Disc. Appl. Math.* 14:135–56.

[KiS80] Kinderlehrer, D., and G. Stampacchia. 1980. *An Introduction to Variational Inequalities and their Applications.* New York: Academic Press.

[KMW67] Karp, R. M., R. E. Miller, and S. Winograd. 1967. The organization of computations for uniform recurrence equations. *J. ACM.* 14:563–90.

[KoB76] Kort, B. W., and D. P. Bertsekas. 1976. Combined primal-dual and penalty methods for convex programming. *SiAM J. Contr.* 14:268–94.

[KoO68] Kowalik, J., and M. R. Osborne. 1968. *Methods for Unconstrained Optimization Problems.* New York: Elsevier.

[Kor76] Korpelevich, G. M. 1976. The extragradient method for finding saddle points and other problems. *Ekon. i Mat. Metody* [translated as *Matecon*]. 12:747–56.

[Kru37] Kruithof, J. 1937. Calculation of telephone traffic. *De Ingenieur (E. Electrotechnik 3).* 52:E15–25.

[Kru87] Kruse, R. L. 1987. *Data Structures and Program Design.* Englewood Cliffs, NJ: Prentice-Hall.

[KSS81] Kung, H. T., B. Sproul, and G. Steele (eds.). 1981. *VLSI Systems and Computations.* Rockville, MD: Computer Science Press.

[Kuc82] Kucera, L. 1982. Parallel computation and conflicts in memory access. *Inf. Proc. Lett.* 14:93–96.

[Kun76a] Kung, H. T. 1976. New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences. *J. ACM.* 23:252–61.

[Kun76b] Kung, H. T. 1976. Synchronized and asynchronous parallel algorithms for multiprocessors. In J. F. Traub (ed.), *Algorithms and Complexity*, 153–200. New York: Academic Press.

[Kun82] Kung, H. T. 1982. Why systolic architectures? *Computer.* 15:37–45.

[Kun88] Kung, S. Y. 1988. *VLSI Array Processors.* Englewood Cliffs, NJ: Prentice-Hall.

[KuP81] Kuhn, R. M., and D. A. Padua (eds.). 1981. *Tutorial on Parallel Processing.* Silver Springs, MD: IEEE Computer Society Press.

[Kus84] Kushner, H. J. 1984. *Approximation and Weak Convergence Methods for Random Processes.* Cambridge, MA: The MIT Press.

[KuV86] Kumar, P. R., and P. P. Varaiya. 1986. *Stochastic Systems: Estimation, Identification, and Adaptive Control.* Englewood Cliffs, NJ: Prentice-Hall.

[KuY87a] Kushner, H. J., and G. Yin. 1987. Stochastic approximation algorithms for parallel and distributed processing. *Stochastics.* 22:219–50.

[KuY87b] Kushner, H. J., and G. Yin. 1987. Asymptotic properties of distributed and communicating stochastic approximation algorithms. *SiAM J. Contr. & Optim.* 25:1266–90.

[KVC88] Krumme, D. W., K. N. Venkataraman, and G. Cybenko. 1988. The token exchange problem. Dept. of Applied Math. Technical Report 88-2. Tufts University, Medford, MA.

[LaF80] R. E. Ladner, and M. J. Fischer. 1980. Parallel prefix computation. *J. ACM.* 27:831–8.

[LaH84] Lawphongpanich, S., and D. W. Hearn. 1984. Simplicial decomposition of the asymmetric traffic assignment problems. *Trans. Res.* 18B:123–33.

[Lam78] Lamport, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM.* 21:558–65.

[LaS81] Lamond, B., and N. F. Stewart. 1981. Bregman's balancing method. *Trans. Res.* 15B:239–48.

[Las70] Lasdon, L. S. 1970. *Optimization Theory for Large Systems.* New York: Macmillan.

[Las73] Lasdon, L. S. 1973. Decomposition in resource allocation. In D. M. Himmelblau (ed.), *Decomposition of Large-Scale Problems,* 207–31. Amsterdam: North-Holland.

[LaT85] Lancaster, P., and M. Tismenetsky. 1985. *The Theory of Matrices.* New York: Academic Press.

[Law67] Lawler, E. L. 1967. Optimal cycles in doubly weighted linear graphs. In P. Rosenstiehl (ed.), *Theory of Graphs,* 209–14. Paris: Dunod; New York: Gordon & Breach.

[Law76] Lawler, E. 1976. *Combinatorial Optimization: Networks and Matroids.* New York: Holt, Rinehart & Winston.

[Lem74] Lemarechal, C. 1974. An algorithm for minimizing convex functions. In J. L. Rosenfeld (ed.), *Information Processing '74,* 552–6. Amsterdam: North-Holland.

[Lem75] Lemarechal, C. 1975. An extension of Davidon methods to nondifferentiable problems. *Math. Prog. Studies.* 3:95–109.

[LeP65] Levitin, E. S., and B. T. Poljak. 1965. Constrained minimization methods. *Z. Vycisl. Mat. i Mat. Fiz.* 6:787–823. English translation in *USSR Comput. Math. Phys.* 6:1–50.

[LeR81] Lehmann, D., and M. Rabin. 1981. On the advantages of free choice: A symmetric solution of the dining philosophers problem. *Proc. 8th ACM Symp. Princ. Prog. Lang.,* 133–8.

[LeV40] Le Verrier, U. J. J. 1840. Sur les variations seculaires des elements elliptiques des sept planets principales. *J. Math. Pures Appl.* 5:220–54.

[LiB87] Li, S. and T. Basar. 1987. Asymptotic agreement and convergence of asynchronous stochastic algorithms. *IEEE Trans. Auto. Contr.* AC-32:612–18.

[LiM79] Lions, P. L., and B. Mercier. 1979. Splitting algorithms for the sum of two nonlinear operators. *SiAM J. Numer. Analy.* 16:964–79.

[LjS83] Ljung, L., and T. Soderstrom. 1983. *Theory and Practice of Recursive Identification.* Cambridge, MA: The MIT Press.

[Lju77] Ljung, L. 1977. Analysis of recursive stochastic algorithms. *IEEE Trans. Auto. Contr.* AC-22:551–75.

[LKK83] Lord, R. E., J. S. Kowalik, and S. P. Kumar. 1983. Solving linear algebraic expressions on an MIMD computer. *J. ACM.* 30:103–17.

[LMS83] Lavenberg, S., R. Muntz, and B. Samadi. 1983. Performance analysis of a rollback method for distributed simulation. In A. K. Agrawala and S. K. Tripathi (eds.), *Performance '83*, 117–32. Amsterdam: North-Holland.

[LMS86] Lang, B., J. C. Miellou, and P. Spiteri. 1986. Asynchronous relaxation algorithms for optimal control problems. *Math. & Comput. Simul.* 28:227–42.

[LoR88] Lootsma, F. A., and K. M. Ragsdell. 1988. State of the art in parallel nonlinear optimization. *Parallel Comput.* 6:131–55.

[LPS87] Lo, S.-S., B. Philippe, and A. Sameh. 1987. A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem. *SiAM J. Sci. & Stat. Comput.* 8:s155–65.

[LRS82] Lelarasmee, E., A. E. Ruehli, and A. L. Sangiovanni-Vincentelli. 1982. The waveform relaxation method for the time-domain analysis of large scale integrated circuits. *IEEE Trans. Comput.-Aided Des. Integ. Circ.* CAD-1:131–45.

[Lue69] Luenberger, D. G. 1969. *Optimization by Vector Space Methods.* New York: Wiley.

[Lue84] Luenberger, D. G. 1984. *Linear and Nonlinear Programming.* Reading, MA: Addison-Wesley.

[LuM86] Lubachevsky, B., and D. Mitra. 1986. A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit spectral radius. *J. ACM.* 33:130–50.

[Luo87] Luo, Z.-Q. 1987. Private communication.

[Luq84] Luque, F. J. 1984. Asymptotic convergence analysis of the proximal point algorithm. *SiAM J. Contr. & Optim.* 22:277–93.

[Luq86a] Luque, F. J. 1986. The nonlinear proximal point algorithm and multiplier methods. Laboratory for Information and Decision Systems Report LIDS-P-1596, Massachusetts Institute of Technology, Cambridge.

[Luq86b] Luque, F. J. 1986. The nonlinear proximal point algorithm. Laboratory for Information and Decision Systems Report LIDS-P-1598, Massachusetts Institute of Technology, Cambridge.

[Mac79] Macgill, S. H. 1979. Convergence and related properties for a modified biproportional problem. *Envir. Plan A.* 11:499–506.

[MaD86] Mangasarian, O. L., and R. De Leone. 1986. Parallel gradient projection successive overrelaxation for symmetric linear complementarity problems and linear programs. Computer Sciences Dept. Technical Report 659, University of Wisconsin-Madison.

[MaD87] Mangasarian, O. L., and R. De Leone. 1987. Parallel successive overrelaxation methods for symmetric linear complementarity problems and linear programs. *JOTA*. 54:437–46.

[Man77] Mangasarian, O. L. 1977. Solution of symmetric linear complementarity problems by iterative methods. *JOTA*. 22:465–85.

[Man84] Mangasarian, O. L. 1984. Sparsity-preserving SOR algorithms for separable quadratic and linear programming. *Comput. & Oper. Res.* 11:105–12.

[MaP88] Maggs, B. M., and S. A. Plotkin. 1988. Minimum-cost spanning tree as a path-finding problem. *Inf. Proc. Lett.* 26:291–3.

[Mar70] Martinet, B. 1970. Regularisation d'inequations variationnelles par approximations successives. *Rev. Francaise Inf. Rech. Oper.* 4:154–59.

[Mar72] Martinet, B. 1972. Determination approchee d'un point fixe d'une application pseudocontractante. *C. R. Acad. Sci. Paris.* 274A:163–5.

[Mar73] Maruyama, K. 1973. On the parallel evaluation of polynomials. *IEEE Trans. Comput.* C-22:2–5.

[McV87] McBryan, O. A., and E. F. Van de Velde. 1987. Hypercube algorithms and implementations. *SiAM J. Sci. Stat. Comput.* 8:s227–87.

[McW77] McQuillan, J. M., and D. C. Walden. 1977. The ARPANET design decisions. *Computer Networks.* 1:243–89.

[MeC80] Mead, C., and L. Conway. 1980. *Introduction to VLSI Systems.* Reading, MA: Addison-Wesley.

[MeZ88] Meyer, R. R., and S. A. Zenios (eds.). 1988. *Parallel Optimization on Novel Computer Architectures, Annals of Operations Research.* Bazel, Switzerland: A. C. Baltzer.

[MiC82] Misra, J., and K. M. Chandy. 1982. Termination detection of diffusing computations in communicating sequential processes. *ACM Trans. Prog. Lang. & Sys.* 4:37.

[MiC87] Mitra, D., and R. A. Cieslak. 1987. Randomized parallel communications on an extension of the Omega network. *J. ACM.* 34:802–24.

[Mie75] Miellou, J. C. 1975. Algorithmes de relaxation chaotique a retards. *R.A.I.R.O.* 9:55–82.

[MiM84] Mitra, D., and I. Mitrani. 1984. Analysis and optimum performance of two message-passing parallel processors synchronized by rollback. In E. Gelenbe (ed.), *Performance '84*, 35–50. New York: Elsevier.

[Min78] Minieka, E. 1978. *Optimization Algorithms for Networks and Graphs.* New York: Marcel Dekker.

[MiR85] Miller, G. L., and J. Reif. 1985. Parallel tree contraction and its applications. *Proc. 26th Annual Symp. Found. Comput. Sci.*, 478–89.

[MiS85] Miellou, J. C., and P. Spiteri. 1985. Un critere de convergence pour des methodes generales de point fixe. *Math. Modelling & Numer. Analy.* 19:645–69.

[Mis86] Misra, J. 1986. Distributed discrete-event simulation. *Comput. Surv.* 18:39–65.

[Mit87] Mitra, D. 1987. Asynchronous relaxations for the numerical solution of differential equations by parallel processors. *SiAM J. Sci. Stat. Comput.* 8:s43–58.

[MMT70] Mesarovic, M. D., D. Macko, and Y. Takahara. 1970. *Theory of Hierarchical Multilevel Systems.* New York: Academic Press.

[Mor65] Moreau, J. J. 1965. Proximite et dualite dans un espace Hilbertien. *Bull. Soc. Math. France.* 93:273–99.

[MoS54] Motzkin, T. S., and I. J. Schoenberg. 1954. The relaxation method for linear inequalities. *Can. J. Math.* 6:393–404.

[MRR80] McQuillan, J. M., I. Richer, and E. C. Rosen. 1980. The new routing algorithm for the ARPANET. *IEEE Trans. Commun.* COM-28:711–19.

[Mul78] Mulvey, J. M. 1978. Testing of a large-scale network optimization program. *Math. Prog.* 15:291–314.

[MuP73] Munro, I., and M. Paterson. 1973. Optimal algorithms for parallel polynomial evaluation. *J. Comput. & Sys. Sci.* 7:189–98.

[MuP76] Muller, D. E., and F. P. Preparata. 1976. Restructuring of arithmetic expressions for parallel evaluation. *J. ACM.* 23:534–43.

[Nag87] Nagurney, A. 1987. Competitive equilibrium problems, variational inequalities and regional science. *J. Reg. Sci.* 27:503–17.

[NaS80] Nassimi, D., and S. Sahni. 1980. An optimal routing algorithm for mesh-connected parallel computers. *J. ACM.* 27:6–29.

[Nem66] Nemhauser, G. L. 1966. *Introduction to Dynamic Programming.* New York: Wiley.

[NeS83] Newton, A. R., and A. L. Sangiovanni-Vincentelli. 1983. Relaxation-based electrical simulation. *IEEE Trans. Electron. Devices.* ED-30:1184–1207.

[OhK84] Ohuchi, A., and I. Kaji. 1984. Lagrangian dual coordinatewise maximization algorithm for network transportation problems with quadratic costs. *Networks.* 14:515–30.

[OLR85] O'hEigeartaigh, M., S. K. Lenstra, and A. H. G. Rinnoy Kan (eds.). 1985. *Combinatorial Optimization: Annotated Bibliographies.* New York: Wiley.

[Orc74] Orcutt, S. E., Jr. 1974. Computer organization and algorithms for very high-speed computations. PhD Thesis, Dept. of Electrical Engineering, Stanford University, Stanford, CA.

[OrR70] Ortega, J. M., and W. C. Rheinboldt. 1970. *Iterative Solution of Nonlinear Equations in Several Variables.* New York: Academic Press.

[OrV85] Ortega, J. M., and R. G. Voigt. 1985. Solution of partial differential equations on vector and parallel computers. *SiAM Rev.* 27:149–240.

[Ozv87] Ozveren, C. 1987. Communication aspects of parallel processing. Laboratory for Information and Decision systems Report LIDS-P-1721, Massachusetts Institute of Technology, Cambridge.

[PaC82a] Pang, J. S., and D. Chan. 1982. Gauss-Seidel methods for variational inequality problems over product sets. Unpublished manuscript. School of Management, University of Texas-Dallas.

[PaC82b] Pang, J. S., and D. Chan. 1982. Iterative methods for variational and complementarity problems. *Math. Prog.* 24:284–313.

[Pan85] Pang, J. S. 1985. Asymmetric variational inequality problems over product sets: Applications and iterative methods. *Math. Prog.* 31:206–19.

[PaR85] Pan, V., and J. Reif. 1985. Efficient parallel solution of linear systems. *Proc. 17th ACM STOC.*, 143–52.

[PaS82] Papadimitriou, C. H., and K. Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity.* Englewood Cliffs, NJ: Prentice-Hall.

[PaT87a] Papadimitriou, C. H., and J. N. Tsitsiklis. 1987. Unpublished research.

[PaT87b] Papadimitriou, C. H., and J. N. Tsitsiklis. 1987. The complexity of Markov decision processes. *Math. Oper. Res.* 12:441–50.

[PaY88] Papadimitriou, C. H., and M. Yannakakis. 1988. Towards an architecture-independent analysis of parallel algorithms. *Proc. 20th ACM STOC.* 510–3.

[Pea84] Pearl, J. 1984. *Heuristics.* Reading, MA: Addison-Wesley.

[Per83] Perlman, R. 1983. Fault-tolerant broadcast of routing information. *Comput. Netw.* 7:395–405.

[Pet88] Peters, J. 1988. A parallel algorithm for minimal cost network flow problems. Computer Sciences Dept., Technical Report 762, University of Wisconsin-Madison.

[Pol69] Poljak, B. T. 1969. Minimization of unsmooth functionals. *USSR Comput. Math. Phys.* 9:14–29.

[Pol71] Polak, E. 1971. *Computational Methods in Optimization: A Unified Approach.* New York: Academic Press.

[Pol87] Poljak, B. T. 1987. *Introduction to Optimization.* New York: Optimization Software.

[PoT73] Poljak, B. T., and Y. Z. Tsypkin. 1973. Pseudogradient adaptation and training algorithms. *Auto. & Rem. Contr.* 12:83–94.

[PoT74] Poljak, B. T., and N. V. Tretjakov. 1974. An iterative method for linear programming and its economic interpretation. *Matecon.* 10:81–100.

[Pow69] Powell, M. J. D. 1969. A method for nonlinear constraints in minimization problems. In R. Fletcher (ed.), *Optimization*, 283–98. New York: Academic Press.

[Pow73] Powell, M. J. D. 1973. On search directions for minimization algorithms. *Math. Prog.* 4:193–201.

[PrS78] Preparata, F. P., and D. V. Sarwate. 1978. An improved parallel processor bound in fast matrix inversion. *Inf. Proc. Lett.* 7:148–9.

[QuD84] Quinn, M. J., and N. Deo. 1984. Parallel graph algorithms. *Comput. Surv.* 16:338–48.

[Qui87] Quinn, M. J. 1987. *Designing Efficient Algorithms for Parallel Computers.* New York: McGraw-Hill.

[RaK88] Rao, S. K., and T. Kailath. 1988. Regular iterative algorithms and their implementation on processor arrays. *Proc. IEEE.* 76:259–69.

[RAP87] Reed, D. A., L. M. Adams, and M. L. Patrick. 1987. Stencils and problem partitionings: Their influence on the performance of multiple processor systems. *IEEE Trans. Comput.* C-36:845–58.

[RCM75] Robert, F., M. Charnay, and F. Musy. 1975. Iterations chaotiques serie-parallele pour des equations non-lineaires de point fixe. *Aplikace Mat.* 20:1–38.

[Rib87] Ribeiro, C. C. 1987. Parallel computer models and combinatorial algorithms. *Ann. Disc. Math.* 31:325–64.

[Rob76] Robert, F. 1976. Contraction en norme vectorielle: Convergence d'iterations chaotiques pour des equations non lineaires de point fixe a plusieurs variables. *Lin. Algeb. & Appl.* 13:19–35.

[Roc70] Rockafellar, R. T. 1970. *Convex Analysis.* Princeton, NJ: Princeton University Press.

[Roc71] Rockafellar, R. T. 1971. New applications of duality in convex programming. *Proc. 4th Conf. Prob.*, 73–81.

[Roc76a] Rockafellar, R. T. 1976. Monotone operators and the proximal point algorithm. *SiAM J. Contr. & Optim.* 14:877–98.

[Roc76b] Rockafellar, R. T. 1976. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Math. Oper. Res.* 1:97–116.

[Roc76c] Rockafellar, R. T. 1976. Solving a nonlinear programming problem by way of a dual problem. *Symp. Mat.* 27:135–60.

[Roc80] Rock, H. 1980. Scaling techniques for minimal cost network flows. In V. Page (ed.), *Discrete Structures and Algorithms.* Munich: Carl Hansen. 181–91

[Roc84] Rockafellar, R. T. 1984. *Network Flows and Monotropic Programming.* New York: Wiley.

[RoM51] Robbins, H., and S. Monro. 1951. A stochastic approximation method. *Ann. Math. Stat.* 22:400–7.

[Ros83a] Ross, S. 1983. *Stochastic Processes.* New York: Wiley.

[Ros83b] Ross, S. 1983. *Introduction to Dynamic Programming.* New York: Academic Press.

[RoW87] Rockafellar, R. T., and R. J.-B. Wets. 1987. Scenarios and policy aggregation in optimization under uncertainty. International Institute of Systems Analysis, Working Paper WP-87-119, Laxenburg, Austria.

[Rud76] Rudin, W. 1976. *Real Analysis.* New York: McGraw-Hill.

[Saa86] Saad, Y. 1986. Communication complexity of the Gaussian elimination algorithm on multiprocessors. *Lin. Algeb. & Appl.* 77:315–40.

[SaK77] Sameh, A. H., and D. J. Kuck. 1977. A parallel QR algorithm for symmetric tridiagonal matrices. *IEEE Trans. Comput.* C-26:147–53.

[SaK78] Sameh, A. H., and D. J. Kuck. 1978. On stable parallel linear solvers. *J. ACM.* 25:81–91.

[Sam77] Sameh, A. H. 1977. Numerical parallel algorithms: A survey. In D. J. Kuck, D. H. Lawrie, and A. H. Sameh (eds.), *High Speed Computer and Algorithm Organization*, 207–28. New York: Academic Press.

[Sam81] Sameh, A. H. 1981. Parallel algorithms in numerical linear algebra. Paper presented at the CREST Conference on the Design of Numerical Algorithms for Parallel Processing, Bergamo, Italy.

[Sam85a] Sameh, A. H. 1985. On some parallel algorithms on a ring of processors. *Comput. Phys. Commun.* 37:159–66.

[Sam85b] Samadi, B. 1985. Distributed simulation: Algorithms and performance analysis. PhD dissertation. Computer Science Dept., UCLA.

[San88] Sanders, B. A. 1988. An asynchronous distributed flow control algorithm for rate allocation in computer networks. *IEEE Trans. Comput.* 37:779–87.

[SaS85] Saad, Y., and M. H. Schultz. 1985. Data communication in hypercubes. Dept. of Computer Sciences, Research Report YALEU/DCS/RR-428, Yale University, New Haven, CT.

[SaS86] Saad, Y., and M. H. Schultz. 1986. Data communication in parallel architectures. Dept. of Computer Sciences, Research Report, Yale University, New Haven, CT.

[SaS87] Saad, Y., and M. H. Schultz. 1987. Parallel direct methods for solving banded linear systems. *Lin. Algeb. & Appl.* 88/89:623–50.

[SaS88] Saad, Y., and M. H. Schultz. 1988. Topological properties of hypercubes. *IEEE Trans. Comput.* 37:867–72.

[Sch80] Schwartz, J. T. 1980. Ultracomputers. *ACM Trans. Prog. Lang. & Sys.* 2:484–521.

[Sch84] Schendel, U. 1984. *Introduction to Numerical Methods for Parallel Computers.* Chichester, England: Ellis Horwood.

[Sch87] Schwartz, M. 1987. *Telecommunication Networks.* Reading, MA: Addison-Wesley.

[ScS80] Schwartz, M., and T. E. Stern. 1980. Routing techniques used in computer communication networks. *IEEE Trans. Commun.* COM-28:539–52.

[ScV82] Schiloach, Y., and V. Vishkin. 1982. An O(n^2 log n) parallel max-flow algorithm. *J. Algor.* 3:128–46.

[Seg83] Segall, A. 1983. Distributed network protocols. *IEEE Trans. Inf. Theory.* IT-29:23–35.

[Sen81] Seneta, E. 1981. *Non-Negative Matrices and Markov Chains.* New York: Springer-Verlag.

[Sha79] Shapiro, J. F. 1979. *Mathematical Programming.* New York: Wiley.

[Sib70] Sibony, M. 1970. Methodes iteratives pour les equations et inequations aux derivees partielles nonlineaires de type monotone. *Calcolo.* 7:65–183.

[Sin77] Singh, M. G. 1977. *Dynamical Hierarchical Control.* Amsterdam: North-Holland.

[SMG78] Segall, A., P. M. Merlin, and R. G. Gallager. 1978. A recoverable protocol for loop-free distributed routing. *Proc. ICC.*

[SpG87] Spinelli, J. M., and R. G. Gallager. 1987. Broadcasting topology information in computer networks. Laboratory for Information and Decision Systems Report LIDS-P-1543, Massachusetts Institute of Technology, Cambridge.

[Spi84] Spiteri, P. 1984. Contribution a l'etude de grands systemes non lineaires. Doctoral thesis. L'Universite de Franche-Comte, Besancon, France.

[Spi85a] Spingarn, J. E. 1985. A primal-dual projection method for solving systems of linear inequalities. *Lin. Algeb. & Appl.* 65:45–62.

[Spi85b] Spingarn, J. E. 1985. Applications of the method of partial inverses to convex programming: Decomposition. *Math. Prog.* 32:199–223.

[Spi85c] Spinelli, J. M. 1985. Broadcasting topology and routing information in computer networks. Laboratory for Information and Decision Systems Report LIDS-TH-1470, Massachusetts Institute of Technology, Cambridge.

[Spi86] Spiteri, P. 1986. Parallel asynchronous algorithms for solving boundary value problems. In M. Cosnard et al. (eds.) *Parallel Algorithms and Architectures.* 73–84. New York: Elsevier.

[Spi87] Spingarn, J. E. 1987. A projection method for least-squares solutions to overdetermined systems of linear inequalities. *Lin. Algeb. & Appl.* 86:211–36.

[SSP85] Szymanski, B., Y. Shi, and N. Prywes. 1985. Terminating iterative solution of simultaneous equations in distributed message passing systems. *J. ACM.* 32:287–92.

[Sta85] Stallings, W., 1985. *Data and Computer Communications.* New York: Macmillan.

[Sta87] Stallings, W. 1987. *Local Networks.* New York: Macmillan.

[Ste73] Stewart, G. W. 1973. *Introduction to Matrix Computations.* New York: Academic Press.

[Ste77] Stern, T. E. 1977. A class of decentralized routing algorithms using relaxation. *IEEE Trans. Commun.* COM-25:1092–1102.

[Sto75] Stone, H. S. 1975. Parallel tridiagonal equation solvers. *ACM Trans. Math. Software.* 1:289–307.

[Sto77] Stoilow, E. 1977. The augmented Lagrangian method in two-level static optimization. *Arch. Auto. Telemech.* 22:219–37.

[Str76] Strang, G. 1976. *Linear Algebra and Its Applications.* New York: Academic Press.

[StW70] Stoer, J., and C. Witzgall. 1970. *Convexity and Optimization in Finite Dimensions I.* New York: Springer-Verlag.

[StW75] Stephanopoulos, G., and A. W. Westerberg. 1975. The use of Hestenes' method of multipliers to resolve dual gaps in engineering system optimization. *JOTA*. 15:285–309.

[Sut83] Sutti, C. 1983. Nongradient minimization methods for parallel processing computers. *JOTA*. 39:465–488.

[Taj77] Tajibnapis, W. D. 1977. A correctness proof of a topology information maintenance protocol for a distributed computer network. *Commun. ACM*. 20:477–85.

[TaM85] Tanikawa, A., and H. Mukai. 1985. A new technique for nonconvex primal-dual decomposition of a large-scale separable optimization problem. *IEEE Trans. Auto. Contr.* AC-30:133–43.

[Tan71] Tanabe, K. 1971. Projection method for solving a singular system of linear equations and its applications. *Numer. Math.* 17:203–14.

[Tan81] Tanenbaum, A. S. 1981. *Computer Networks*. Englewood Cliffs, NJ: Prentice-Hall.

[Tar85] Tardos, E. 1985. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*. 5:247–55.

[TBA86] Tsitsiklis, J. N., D. P. Bertsekas, and M. Athans. 1986. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Trans. Auto. Contr.* AC-31:803–12.

[TBT88] Tseng, P., D. P. Bertsekas, and J. N. Tsitsiklis. 1988. Partially Asynchronous Parallel Algorithms for Network Flow and Other Problems. Laboratory for Information and Decision Systems Unpublished Report, Massachusetts Institute of Technology, Cambridge.

[Tho87] Thompson, K. M. 1987. A two-stage successive overrelaxation algorithm for solving the linear complementarity problem. Computer Sciences Dept., Technical Report 706, University of Wisconsin-Madison.

[Top85] Topkis, D. M. 1985. Concurrent broadcast for information dissemination. *IEEE Trans. Software Eng.* 13:207–31.

[Top87] Topkis, D. M. 1987. All-to-all broadcast by flooding in communications networks. Graduate School of Management, University of California-Davis. To appear in *IEEE Trans. Comput.*

[TsA84] Tsitsiklis, J. N., and M. Athans. 1984. Convergence and asymptotic agreement in distributed decision problems. *IEEE Trans. Auto. Contr.* AC-29:42–50.

[Tsa86] Tsai, W. K. 1986. Optimal quasi-static routing for virtual circuit networks subjected to stochastic inputs. PhD thesis. Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge.

[TsB86] Tsitsiklis, J. N., and D. P. Bertsekas. 1986. Distributed asynchronous optimal routing in data networks. *IEEE Trans. Auto. Contr.* AC-31:325–32.

[TsB87a] Tseng, P., and D. P. Bertsekas. 1987. Relaxation methods for problems with strictly convex separable costs and linear constraints. *Math. Prog.* 38:303–21.

[TsB87b] Tseng, P., and D. P. Bertsekas. 1987. Relaxation methods for linear programs. *Math. Oper. Res.* 12:569–96.

[TsB87c] Tseng, P., and D. P. Bertsekas. 1987. Relaxation methods for monotropic programs. Laboratory for Information and Decision Systems Report LIDS-P-1697, Massachusetts Institute of Technology, Cambridge. To appear in *Math. Prog.*

[Tse85] Tseng, P. 1985. The relaxation method for a special class of linear programming problems. Laboratory for Information and Decision Systems Report LIDS-P-1467, Massachusets Institute of Technology, Cambridge.

[Tse86] Tseng, P. 1986. Relaxation methods for monotropic programming problems. PhD thesis. Dept. of Electrical Eng. and Comp. Science, Massachusetts Institute of Technology, Cambridge.

[Tse87] Tseng, P. 1987. Private communication.

[Tse88] Tseng, P. 1988. Private communication.

[Tsi84] Tsitsiklis, J. N. 1984. Problems in decentralized decision making and computation. PhD thesis. Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge.

[Tsi87] Tsitsiklis, J. N. 1987. On the stability of asynchronous iterative processes. *Math. Sys. Theory.* 20:137–53.

[TTB86] Tsai, W. K., J. N. Tsitsiklis, and D. P. Bertsekas. 1986. Some issues in distributed asynchronous routing in virtual circuit data networks. *Proc. 25th Conf. Dec. & Contr.*, 1335–7.

[Ull84] Ullman, J. D. 1984. *Computational Aspects of VLSI.* Rockeville, MD: Computer Science Press.

[UrD86] Uresin, A., and M. Dubois. 1986. Generalized asynchronous iterations. *Proc. Conf. Algor. & Hardware Parallel Proc.*

[UrD88a] Uresin, A., and M. Dubois. 1988. Sufficient conditions for the convergence of asynchronous iterations. Computer Research Institute, Technical Report, University of Southern California-Los Angeles. To appear in *Parallel Comput.*

[UrD88b] Uresin, A., and M. Dubois. 1988. Parallel asynchronous algorithms for discrete data. Computer Research Institute, Technical Report CRI-88-05, University of Southern California-Los Angeles.

[VaB81] Valiant, L. G., and G. J. Brebner, 1981. Universal schemes for parallel communication. *Proc. 13th Annual ACM STOC.*, 263–77.

[Val82] Valiant, L. G. 1982. A scheme for fast parallel communication. *SiAM J. Comput.* Vol. 11:350–61.

[Var62] Varga, R. S. 1962. *Matrix Iterative Methods.* Englewood Cliffs, NJ: Prentice-Hall.

[VeP87] Verdu, S., and V. Poor. 1987. Abstract dynamic programming models under commutativity conditions. *SiAM J. Contr. & Optim.* 25:990–1006.

[Ver87] Verjus, J. P. 1987. On the proof of a distributed algorithm. *Inf. Proc. Lett.* 25:145–7.

[Vid78] Vidyasagar, M. 1978. *Nonlinear Systems Analysis.* Englewood Cliffs, NJ: Prentice-Hall.

[Whi82] Whittle, P. 1982. *Optimization Over Time*, Volume 1. New York: Wiley.

[Whi83] Whittle, P. 1983. *Optimization Over Time*, Volume 2. New York: Wiley.

[WiH80] Wing, O., and S. W. Huang. 1980. A computation model of parallel solution of linear equations. *IEEE Trans. Comput.* C-29:632–7.

[Wis71] Wismer, D. A., (ed.). 1971. *Optimization Methods for Large-Scale Systems with Applications*. New York: McGraw-Hill.

[WNM78] Watanabe, N., Y. Nishimura, and M. Matsubara. 1978. Decomposition in large system optimization using the method of multipliers. *JOTA*. 25:181–93.

[WoH85] Wong, E., and B. Hajek. 1985. *Stochastic Processes in Engineering Systems*. New York: Springer-Verlag.

[WuF84] Wu, C.-L., and T.-Y. Feng (eds.). 1984. *Interconnection Networks for Parallel and Distributed Processing*. New York: IEEE Computer Society Press.

[You71] Young, D. M. 1971. *Iterative Solution of Large Linear Systems*. New York: Academic Press.

[Zan69] Zangwill, W. I. 1969. *Nonlinear Programming: A Unified Approach*. Englewood Cliffs, NJ: Prentice-Hall.

[ZeL87] Zenios, S. A., and R. A. Lasken, 1987. Nonlinear network optimization on a massively parallel connection machine. Decision Sciences Dept., Report 87-08-03, The Wharton School, University of Pennsylvania, Philadelphia. To appear in *Ann. Oper. Res.*

[ZeM86] Zenios, S. A., and J. M. Mulvey. 1986. Relaxation techniques for strictly convex network problems. In C. L. Monma (ed.), *Algorithms and Software for Optimization, Annals of Operations Research*, Volume 5, Bazel, Switzerland: A. C. Baltzer.

[Zou76] Zoutendijk, G. 1976. *Mathematical Programming Methods*. Amsterdam: North-Holland.

Index

Corrections (1997)

p. 19 (Fig. 1.2.5) Change A_3 to A^3

p. 22 (Fig. 1.2.8) Change top $4, 1$ to $4, 0$

p. 42 (Fig. 1.3.4) Change top $\sum_{i=1}^{5} a_i$ to $\sum_{i=1}^{9} a_i$

p. 65 (-9) Change "any single" to "an optimal single"

p. 67 (Table 3.3) The upper bound $p - 1$ for the total exchange has been reduced to $p/2$; see the paper "Optimal Communication Algorithms for Hypercubes," J. of Parallel and Distributed Computation, Vol. 11, 1991, pp. 263-275, by D. P. Bertsekas, C. Ozveren, G. Stamoulis, P. Tseng, and J. N. Tsitsiklis.

p. 80 (+15) Change "at every time ..." to "at the first time unit, each node sends its packet to all its neighbors. Afterwards, at every time ..."

p. 86 (+7) Change "bo" to "be"

p. 86 (+9) Change $2 \log n$ to $4 \log n$

p. 86 (-1) Change "2 time units using the links illustrated" to "4 time units using the transfers illustrated"

p. 88 (+5) Change "processors." to "processors arranged in a square array."

p. 103 (-9) Change "that" to "than"

p. 106 (+13) Change ρ_S^{t-D} to ρ_S^t

p. 130 (+17) Change $L'DL$ to LDL'

p. 130 (+19) Delete "square"

p. 179 (+5) Change "[Ash]" to "[Ash70]"

p. 240 (Proof of part (c)) This proof works for the case where the penalty parameter sequence $c(t)$ is bounded above. If $c(t) \to \infty$ a slightly different argument is needed.

p. 264 (+16) Change λ_j to λ_i

p. 271 (+5) Change 3.13 to 3.1.3

p. 307 (-2) Change $\min\{x_{i(k+1)}^k, x_{(k+1)j}^k\}$ to $\max\{x_{i(k+1)}^k, x_{(k+1)j}^k\}$

p. 327 (+13) Change $[x \mid x \geq 0, x_1 = 0]$ to $\{x \mid x \geq 0, x_1 = 0\}$

p. 335 (+17) Change "capacity" to "demand"

p. 339 (-3) Change p_i to p_j

p. 340 (+10) Change ":=" to "="; change p_i to p_j

p. 363 (+2) Change b_{ij} to b_{ji}

p. 368 (-3) Change "assigment" to "assignment"

p. 375 (+6) Change "$\epsilon < 1/n$" to "$\epsilon < 1/m$, where m is the number of similarity classes"

p. 375 (-9) Change "Exercise 3.7" to "Exercise 3.6"

p. 386 (+16) Change $O(|N|^3+$ to $O(M|N|^3+$

p. 390 (Exercise 4.5) Change all occurances of "8" to "6"

p. 392 (Figure 5.5.1) Change "Slope $= b_{ij}$" to "Slope $= -b_{ij}$"; change "Slope $= c_{ij}$" to "Slope $= -c_{ij}$"

p. 396 (-3) In the middle figure change $q(t_{ij})$ to $q_{ij}(t_{ij})$

p. 386 (+16) Change $O(|N|^3+$ to $O(M|N|^3+$

p. 408 (-9) Change $f_{ij} \leq 0$ to $f_{ij} < 0$

p. 423 (-7) Change "result" to "results"

p. 429 (-7,-8,-12,-13) Change all X to x

p. 430 (-10) Change $\tau_4^2(9)$ to $\tau_3^2(9)$

p. 459 (+3) Change $R_i(p)$ to R_i

p. 548 (+12) Change "rerouting" to "routing"

p. 569 (+6) Change "of a large" to "for a large"

p. 571 For a recent treatment of the problem of termination detection, including additional methods and references, see the paper "Finite Termination of Asynchronous Iterative Algorithms," Parallel Computing, Vol. 22, 1996, pp. 39-56, by S. A. Savari and D. P. Bertsekas.

p. 618 (+3) Change [GaB87] to [GaB86]

p. 620 (+8) Change $\ln n$ to $\ln x$

p. 620 (+17) Delete "$< x, y >$"

p. 649 (-4) Include as an additional assumption that Z is closed

p. 659 (-8) Change "A set C" to "A nonempty set C"

p. 659 (-1) A proof of the closure of the cone C appears in many sources, including "Nonlinear Programming," by D. P. Bertsekas, Athena Scientific, 1995, p. 580.