PART 2
Asynchronous Algorithms

6

Totally Asynchronous Iterative Algorithms

In this chapter and the next, we discuss asynchronous counterparts of many of the algorithms analyzed earlier. We have in mind a situation where an algorithm is parallelized by separating it into several local algorithms operating concurrently at different processors. The main characteristic of an asynchronous algorithm is that the local algorithms do not have to wait at predetermined points for predetermined messages to become available. We thus allow some processors to compute faster and execute more iterations than others, we allow some processors to communicate more frequently than others, and we allow the communication delays to be substantial and unpredictable. We also allow the communication channels to deliver mesages out of order, that is, in a different order than the one in which they were transmitted.

The advantages one hopes to gain from asynchronism are twofold. First, a reduction of the synchronization penalty and a potential speed advantage over synchronous algorithms in some problems, perhaps at the expense of higher communication complexity (see Subsection 1.4.2, and Sections 6.3 and 6.4). Second, a greater implementation flexibility and tolerance to problem data changes during the algorithm's execution, as discussed in Section 1.4.

On the negative side, we will find that the conditions for validity of an asynchronous algorithm may be more stringent than the corresponding conditions for its synchronous counterpart. Furthermore the detection of termination tends to be somewhat more difficult for asynchronous than for synchronous algorithms (see the discussion of Section 8.1).

An interesting fact is that some asynchronous algorithms, called *totally asynchronous*, or *chaotic*, can tolerate arbitrarily large communication and computation delays, while other asynchronous algorithms, called *partially asynchronous*, are not guaranteed to work unless there is an upper bound on these delays. The convergence mechanisms at work in each of these two cases are genuinely different and so are their analyses. In the present chapter, we concentrate on totally asynchronous algorithms, and in the next chapter we take up the partially asynchronous case.

In the next section, we describe the totally asynchronous algorithmic model in the context of fixed point problems. In Section 6.2, we formulate a general convergence theorem for the natural distributed version of the general fixed point iteration discussed in Subsection 1.2.4. In Section 6.3, we present a number of examples involving mappings that are contractions with respect to a weighted maximum norm. Sufficient conditions for convergence are given for general nonlinear problems, and necessary conditions for convergence are given for linear problems. We also provide a rate of convergence analysis and a comparison between synchronous and asynchronous algorithms. In Section 6.4, we consider iterations involving monotone mappings, including the shortest path problem. We provide examples showing that the asynchronous version of the Bellman–Ford algorithm can require many more message transmissions than its synchronous counterpart, even though it finds the shortest distances at least as fast (and often faster). On the other hand, the number of message transmissions that the asynchronous algorithm requires on the average is probably acceptable in most practical situations. In Section 6.5, we consider linear network flow problems and a distributed asynchronous version of the ϵ–relaxation method of the previous chapter. In Section 6.6 we consider nonlinear network flow problems, and the corresponding asynchronous distributed relaxation method. Finally, in Section 6.7, we consider relaxation methods for solving ordinary differential equations and two–point boundary value problems.

6.1 THE TOTALLY ASYNCHRONOUS ALGORITHMIC MODEL

In this section, we consider a general fixed point problem and provide an algorithm which is a natural distributed version of the fixed point iteration discussed in Subsection 1.2.4. In the next section, we formulate a convergence theorem of broad applicability. Subsequent sections make extensive use of this theorem.

Let $X_1, X_2, \ldots, X_n$ be given sets, and let X be their Cartesian product:

$$X = X_1 \times X_2 \times \cdots \times X_n.$$

Elements of X are written as n–tuples of their "components", that is, for $x \in X$, we write

$$x = (x_1, x_2, \ldots, x_n),$$

where x_i are the corresponding elements of X_i, $i = 1, \ldots, n$. We assume that there is a notion of sequence convergence defined on X. Let $f_i : X \mapsto X_i$ be given functions, and let $f : X \mapsto X$ be the function defined by

$$f(x) = \big(f_1(x), f_2(x), \ldots, f_n(x)\big), \qquad \forall\, x \in X.$$

The problem is to find a fixed point of f, that is, an element $x^* \in X$ with $x^* = f(x^*)$ or, equivalently,

$$x_i^* = f_i(x^*), \qquad \forall\, i = 1, \ldots, n.$$

We now describe a distributed asynchronous version of the iterative method

$$x_i := f_i(x), \qquad i = 1, \ldots, n.$$

Let

$$x_i(t) \;=\; \text{Value of } i\text{th component at time } t.$$

We assume that there is a set of times $T = \{0, 1, 2, \ldots\}$ at which one or more components x_i of x are updated by some processor of a distributed computing system. Let

$$T^i \;=\; \text{Set of times at which } x_i \text{ is updated.}$$

We assume that the processor updating x_i may not have access to the most recent value of the components of x; thus, we assume that

$$x_i(t+1) = f_i\Big(x_1\big(\tau_1^i(t)\big), \ldots, x_n\big(\tau_n^i(t)\big)\Big), \qquad \forall\, t \in T^i, \tag{1.1a}$$

where $\tau_j^i(t)$ are times satisfying

$$0 \le \tau_j^i(t) \le t, \qquad \forall\, t \in T.$$

At all times $t \notin T^i$, x_i is left unchanged:

$$x_i(t+1) = x_i(t), \qquad \forall\, t \notin T^i. \tag{1.1b}$$

The elements of T should be viewed as the indices of the sequence of physical times at which updates take place. The sets T^i, as well as the sequences of physical times that they represent need not be known to any one processor, since their knowledge is not required to execute iteration (1.1). Thus, there is no requirement for a shared global clock or synchronized local clocks at the processors. The difference $\big(t - \tau_j^i(t)\big)$ between the current time t and the time $\tau_j^i(t)$ corresponding to the jth component available at the processor updating $x_i(t)$ can be viewed as a form of communication delay; see the following

examples. A useful conceptual model is that some processor awakes spontaneously at the times $t \in T^i$, receives by some mechanism the values $x_1\big(\tau_1^i(t)\big), \ldots, x_n\big(\tau_n^i(t)\big)$, and then updates x_i using Eq. (1.1a) without knowing any other quantity such as t, $\tau_1^i(t), \ldots, \tau_n^i(t)$ or any element of T^j, $j = 1, \ldots, n$. In a real computational environment, some of these quantities may be known to some of the processors, but as long as they are not used in iteration (1.1a), it is still appropriate to regard them as unknown.

Note that the Jacobi, Gauss–Seidel, and block iterative methods discussed in previous chapters are special cases of the asynchronous iteration (1.1). It will be seen in this chapter that the conditions for satisfactory convergence of this iteration are, generally speaking, not much more stringent than the corresponding conditions for the above methods.

We describe two examples of situations covered by the model:

Example 1.1. *Message–Passing System*

Consider a network of n processors, each having its own local memory. Processor i stores the vector

$$x^i(t) = \big(x_1^i(t), \ldots, x_n^i(t)\big),$$

updates its ith component $x_i^i(t)$ at times $t \in T^i$ according to

$$x_i^i(t+1) = f_i\big(x^i(t)\big),$$

and occasionally, at some unspecified times, communicates its stored value of the ith component x_i^i to the other processors. When processor j receives (after some unspecified communication delay) the value of x_i^i, it stores this value in place of its currently stored ith component of x^j. Immediately after this happens, we have

$$x_i^j(t) = x_i^i\big(\tau_i^j(t)\big),$$

so we can view $\big(t - \tau_i^j(t)\big)$ as a communication delay. It is natural to assume also that $\tau_i^i(t) = t$. The asynchronous iteration model (1.1) applies with the identification

$$x_i(t) \sim x_i^i(t).$$

Thus, the components of the vector

$$x(t) = \big(x_1^1(t), x_2^2(t), \ldots, x_n^n(t)\big)$$

generated by the algorithm are distributed among the processors, with processor i holding $x_i^i(t)$. The distributed vector $x(t)$ and the "local" vectors $x^1(t), \ldots, x^n(t)$ stored at the processors need not be equal at any time; see the example of Fig. 6.1.1. Note that we do not assume that the communication channels between processors preserve the order of the messages transmitted; neither do we assume that a processor can determine whether an update received from another processor is older than the corresponding value stored in its memory.

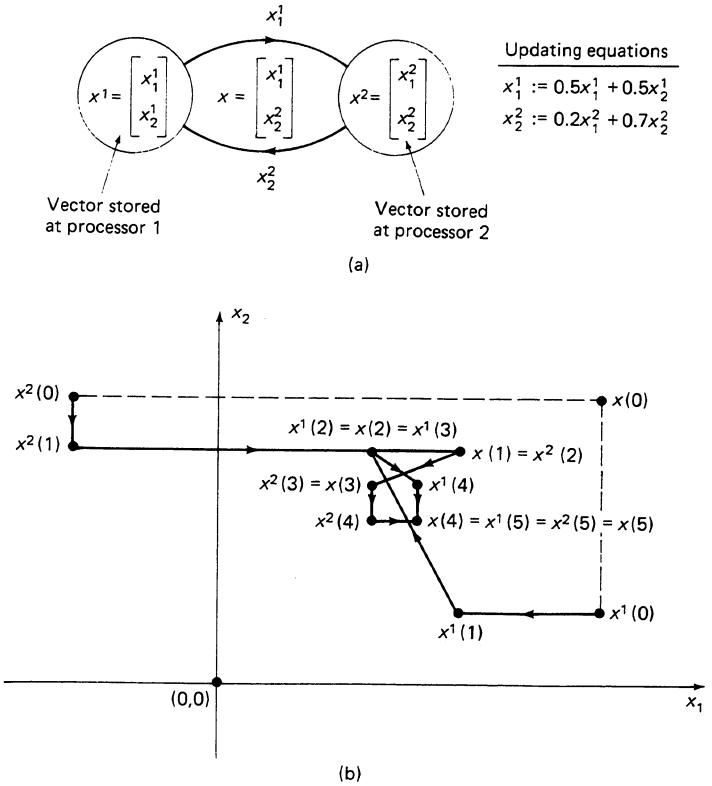


Figure 6.1.1 (a) An example of an asynchronous algorithm involving two processors and a problem with two variables. Processor i, $(i = 1, 2)$, stores $x^i = (x_1^i, x_2^i)$, updates the variable x_i^i at times $t \in T^i$, and transmits it to the other processor. (b) Evolution of the vector $x(t) = \left(x_1^1(t), x_2^2(t)\right)$, and the vectors $x^i(t) = \left(x_1^i(t), x_2^i(t)\right)$, $(i = 1, 2)$, stored at the processors using the iteration

$$x_1 := 0.5x_1 + 0.5x_2, \qquad x_2 := 0.2x_1 + 0.7x_2$$

for the initial conditions $x^i(0)$ shown and the following sequence of events:

t=0: Processor 1 updates x_1^1 and transmits it to processor 2, where it is received at a time between $t = 1$ and $t = 2$. Processor 2 updates x_2^2 and transmits it to processor 1, where it is received at a time between $t = 1$ and $t = 2$.

t=1: Processor 1 updates x_1^1 and transmits it to processor 2, where it is received at a time between $t = 2$ and $t = 3$. [Processor 2 does not update X_2^2 but X_1^2 will change at some time $t \in (1, 2)$ because of the reception of $X_1^1(1)$; for this reason, $x^2(1) \neq x^2(2)$ as shown in the figure.]

t=2: Processor 2, having received $x_1^1(1)$, updates x_2^2 and transmits it to processor 1, where it is received at a time between $t = 3$ and $t = 4$. [Processor 1 does not update X_1^1 and does not receive a value of X_2^2 from Processor 2 at any time $t \in (2, 3)$; for this reason, $X^1(2) = X^1(3)$ as shown in the figure.]

t=3: Processor 1, having received $x_2^2(1)$ [at some time $t \in (1, 2)$], updates x_1^1 and transmits it to processor 2, where it is received at a time between $t = 4$ and $t = 5$. Processor 2, having received $x_1^1(2)$, updates x_2^2 and transmits it to processor 1, where it is received at a time between $t = 4$ and $t = 5$.

t=4: Processor 1 has already received $x_2^2(3)$; no updating takes place.

t=5: Processor 1 has already received $x_2^2(4)$ and processor 2 has already received $x_1^1(4)$.

Example 1.2. *Shared Memory System*

Consider a shared memory multiprocessor, where the memory has registers for $x_1, x_2, \ldots, x_n$, with the ith register storing at time t the value $x_i(t)$. A processor starts reading the values of components from the memory at some time, performs the iteration (1.1a), and eventually, at another time, writes the new value of the corresponding component; see Fig. 6.1.2. It is not necessary to have a one–to–one correspondence between processors and components in this model. Furthermore, it is possible that $\tau_i^i(t) < t$ for $t \in T^i$. Note, however, that by requiring that there can be no more than one processor simultaneously executing an iteration of the ith component, we can be sure that x_i will not change between a time $t \in T^i$ and the corresponding time $\tau_i^i(t)$. Under these circumstances, we can assume without loss of generality that

$$\tau_i^i(t) = t, \qquad \forall\, t \in T^i.$$

This condition will play a significant role in one of the partially asynchronous models of Chapter 7 [see Assumption 7.1(c) in Section 7.1, as well as the discussion following Proposition 3.1 and Example 3.1 in Section 6.3].

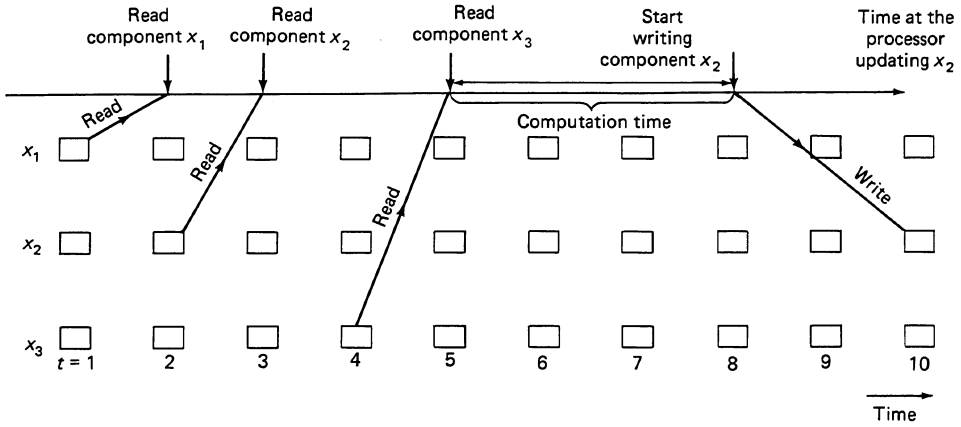


Figure 6.1.2 Illustration of a component update in a shared memory multiprocessor. Here x_2 is viewed as being updated at time $t = 9$ ($9 \in T^2$), with $\tau_1^2(9) = 1$, $\tau_2^2(9) = 2$, and $\tau_4^2(9) = 4$. The updated value of x_2 is entered at the corresponding register at $t = 10$. Several components can be simultaneously in the process of being updated, and the values of $\tau_j^i(t)$ can be unpredictable.

Throughout this chapter, we make the following standing assumption:

Assumption 1.1. (*Total Asynchronism*) The sets T^i are infinite, and if $\{t_k\}$ is a sequence of elements of T^i that tends to infinity, then $\lim_{k\to\infty} \tau_j^i(t_k) = \infty$ for every j.

This assumption guarantees that each component is updated infinitely often, and that old information is eventually purged from the system. More precisely, given any time t_1, there exists a time $t_2 > t_1$ such that

$$\tau_j^i(t) \geq t_1, \qquad \forall\, i, j, \text{ and } t \geq t_2. \tag{1.2}$$

In words, given any time t_1, values of components generated prior to t_1 will not be used in updates after a sufficiently long time t_2. On the other hand, the amounts $t-\tau_j^i(t)$ by which the variables used in iterations are outdated can become unbounded as t increases. This is the main difference with the partial asynchronism assumptions that will be introduced in connection with the asynchronous algorithms of Chapter 7.

6.2 A GENERAL CONVERGENCE THEOREM

In this section, we establish a pattern for proving convergence of the asynchronous algorithm of the previous section. The idea is to provide a set of sufficient conditions, which, when satisfied in a given fixed point problem, guarantee convergence of the algorithm.

Assumption 2.1. There is a sequence of nonempty sets $\{X(k)\}$ with

$$\cdots \subset X(k+1) \subset X(k) \subset \cdots \subset X \tag{2.1}$$

satisfying the following two conditions:

(a) (*Synchronous Convergence Condition*) We have

$$f(x) \in X(k+1), \qquad \forall\ k \text{ and } x \in X(k). \tag{2.2}$$

Furthermore, if $\{y^k\}$ is a sequence such that $y^k \in X(k)$ for every k, then every limit point of $\{y^k\}$ is a fixed point of f.

(b) (*Box Condition*) For every k, there exist sets $X_i(k) \subset X_i$ such that

$$X(k) = X_1(k) \times X_2(k) \times \cdots \times X_n(k).$$

Note that the Synchronous Convergence Condition implies that the limit points of sequences generated by the (synchronous) iteration $x := f(x)$ are fixed points of f, assuming that the initial x belongs to $X(0)$. Note also that the Box Condition implies that by combining components of vectors in $X(k)$, we obtain vectors in $X(k)$, that is, if $x \in X(k)$ and $\bar{x} \in X(k)$, and we replace the ith component of x with the ith component of $\bar{x}$, we obtain an element of $X(k)$. A typical example where the box condition holds is when $X(k)$ is a sphere in $\Re^n$ with respect to some weighted maximum norm.

Our main result is the following:

Proposition 2.1. (*Asynchronous Convergence Theorem*) If the Synchronous Convergence and Box Conditions of Assumption 2.1 hold, and the initial solution estimate

$$x(0) = \big(x_1(0), \ldots, x_n(0)\big)$$

belongs to the set $X(0)$, then every limit point of $\{x(t)\}$ is a fixed point of f.

Proof. We show by induction that for each $k \geq 0$, there is a time t_k such that:

(a) $x(t) \in X(k)$ for all $t \geq t_k$.

(b) For all i and $t \in T^i$ with $t \geq t_k$, we have $x^i(t) \in X(k)$, where

$$x^i(t) = \Big(x_1\big(\tau_1^i(t)\big), x_2\big(\tau_2^i(t)\big), \ldots, x_n\big(\tau_n^i(t)\big)\Big), \qquad \forall\, t \in T^i.$$

[In words, after some time, all solution estimates will be in $X(k)$ and all estimates used in iteration (1.1a) will come from $X(k)$.]

The induction hypothesis is true for $k = 0$, since the initial estimate is assumed to be in $X(0)$. Assuming it is true for a given k, we will show that there exists a time t_{k+1} with the required properties. For each $i = 1, \ldots, n$, let t^i be the first element of T^i such that $t^i \geq t_k$. Then by the Synchronous Convergence Condition, we have $f\big(x^i(t^i)\big) \in X(k+1)$, implying (in view of the Box Condition) that

$$x_i(t^i + 1) = f_i\big(x^i(t^i)\big) \in X_i(k+1).$$

Similarly, for every $t \in T^i$, $t \geq t^i$, we have $x_i(t+1) \in X_i(k+1)$. Between elements of T^i, $x_i(t)$ does not change. Thus,

$$x_i(t) \in X_i(k+1), \qquad \forall\, t \geq t^i + 1.$$

Let $t'_k = \max_i\{t^i\} + 1$. Then, using the Box Condition we have

$$x(t) \in X(k+1), \qquad \forall\, t \geq t'_k.$$

Finally, since by the Continuing Update Assumption 1.1, we have $\tau_j^i(t) \to \infty$ as $t \to \infty$, $t \in T^i$, we can choose a time $t_{k+1} \geq t'_k$ that is sufficiently large so that $\tau_j^i(t) \geq t'_k$ for all i, j, and $t \in T^i$ with $t \geq t_{k+1}$. We then have, $x_j\big(\tau_j^i(t)\big) \in X_j(k+1)$, for all $t \in T^i$ with $t \geq t_{k+1}$ and all $j = 1, \ldots, n$, which (by the Box Condition) implies that

$$x^i(t) = \Big(x_1\big(\tau_1^i(t)\big), x_2\big(\tau_2^i(t)\big), \ldots, x_n\big(\tau_n^i(t)\big)\Big) \in X(k+1).$$

The induction is complete. **Q.E.D.**

A number of useful extensions of the Asynchronous Convergence Theorem are provided in Exercises 2.1–2.3.

The Asynchronous Convergence Theorem is a powerful aid in showing convergence of totally asynchronous algorithms in a variety of contexts. The challenge in applying the theorem is to identify a suitable sequence of sets $\big\{X(k)\big\}$. In many cases, this is

straightforward, but in other cases, it requires creative analysis. This is reminiscent of the process of identifying a Lyapunov function in the stability analysis of nonlinear dynamic systems ([Bro70] and [Vid78]). In fact, our theorem is conceptually related to Lyapunov stability theorems, with the sets $X(k)$ playing the role of the level sets of a Lyapunov function. When application of the Asynchronous Convergence Theorem to a given algorithm seems unlikely, this is a strong indication that the algorithm does not converge under totally asynchronous conditions. We will make this statement rigorous in the next section for the case of a linear mapping f (see, however, Exercise 2.4 for the case where f is nonlinear).

EXERCISES

2.1. Formulate and prove an appropriate extension of the Asynchronous Convergence Theorem for the case where the update equation is time dependent, that is,

$$x_i(t+1) = f_i\Big(x_1\big(\tau_1^i(t)\big), \ldots, x_n\big(\tau_n^i(t)\big), t\Big), \qquad \forall\, t \in T^i.$$

2.2. In some cases, a component of x may occur several times in the formula for $f(x)$, [e.g., $f_1(x_1, x_2) = x_1x_2 + x_1$, or $f_i(x) = g_i\big(h_1^i(x), h_2^i(x), \ldots, h_m^i(x)\big)$, where g_i and h_j^i are given functions]. It is then possible that the values used for different occurrences of the same component have incurred different delays; for example when the value of f is determined by the values of several *functions* of x and these values become available at a processor with differing delays. This leads to a generalization of the asynchronous iteration (1.1a) described by

$$\begin{aligned} x_i(t+1) =& f_i\Big(x_1\big(\tau_1^i(t,1)\big), \ldots, x_1\big(\tau_1^i(t,m)\big), x_2\big(\tau_2^i(t,1)\big), \ldots, x_2\big(\tau_2^i(t,m)\big), \\ & \ldots, x_n\big(\tau_n^i(t,1)\big), \ldots, x_n\big(\tau_n^i(t,m)\big)\Big), \qquad i = 1, \ldots, n, \end{aligned}$$

where m is the number of occurences of each component. Formulate and prove an appropriate extension of the Asynchronous Convergence Theorem.

2.3. Formulate and prove an appropriate extension of the Asynchronous Convergence Theorem for the case where $f_i(x)$ is a subset of X_i for each $x \in X$ and we want to find an $x \in X$ such that $x_i \in f_i(x)$ for all i.

2.4. This exercise provides an example where the Asynchronous Convergence Theorem does not apply even though the asynchronous iteration (1.1) converges appropriately for all starting points. Consider the function $f : \Re^2 \mapsto \Re^2$ defined by $f_1(x_1, x_2) = 0$ for all (x_1, x_2) and defined by $f_2(x_1, x_2) = 2x_2$ when $x_1 \neq 0$ and $f_2(0, x_2) = x_2/2$ when $x_1 = 0$. Show that a sequence $\{x(t)\}$ generated by the iteration (1.1) converges to the unique fixed point of f. Show also that there is no sequence of sets $\{X(k)\}$ satisfying the Synchronous Convergence and Box Conditions for which $X(0)$ contains a vector $x = (x_1, x_2)$ with $x_1 \neq 0$.

6.3 APPLICATIONS TO PROBLEMS INVOLVING MAXIMUM NORM CONTRACTION MAPPINGS

In this section, we provide several examples of application of the Asynchronous Convergence Theorem of the previous section. All these examples involve mappings that are contractions or pseudocontractions with respect to a weighted maximum norm

$$\|x\|_\infty^w = \max_i \frac{|x_i|}{w_i}$$

on $\Re^n$, where $w \in \Re^n$ is a vector with positive coordinates. The key property of a weighted maximum norm in this regard is that its unit sphere has the box property that is central in Assumption 2.1.

Suppose that $f : \Re^n \mapsto \Re^n$ is a contraction mapping with respect to the norm above, and suppose that $X_i = \Re$ for $i = 1, \ldots, n$ and $X = \Re^n$ in the definition of the algorithmic model of Section 6.1. Define the sets

$$X(k) = \left\{x \in \Re^n \mid \|x - x^*\|_\infty^w \leq \alpha^k \|x(0) - x^*\|_\infty^w\right\},$$

where x^* is the unique fixed point of f, and $\alpha < 1$ is the contraction modulus (cf. Prop. 1.1 in Section 3.1). It is evident that the Synchronous Convergence and the Box Conditions of Assumption 2.1 are satisfied. Therefore, asynchronous convergence in the sense of the theorem of the previous section is guaranteed. A similar conclusion holds if f is a pseudocontraction mapping with respect to the norm above (cf. Prop. 1.2 in Section 3.1).

We now consider several problems, discussed in previous chapters, that involve weighted maximum norm contractions.

6.3.1 Solution of Linear Systems of Equations

Consider the case where

$$f(x) = Ax + b$$

for a given $n \times n$ matrix A and vector $b \in \Re^n$. Thus, we wish to find x^* such that

$$x^* = Ax^* + b$$

by means of the asynchronous version of the relaxation iteration $x := Ax + b$ given by

$$x_i(t+1) = \sum_{j=1}^n a_{ij} x_j\left(\tau_j^i(t)\right) + b_i, \qquad t \in T^i, \tag{3.1}$$

$$x_i(t+1) = x_i(t), \qquad t \notin T^i, \tag{3.2}$$

where a_{ij} is the ijth entry of A. The Asynchronous Convergence Theorem applies provided A corresponds to a weighted maximum norm contraction. In Section 2.6, we saw that this is equivalent to the condition

$$\rho\big(|A|\big) < 1,$$

where $\rho\big(|A|\big)$ is the spectral radius of the matrix $|A|$ having as elements the absolute values $|a_{ij}|$.

As an example, consider the iteration

$$\tilde{\pi}(t+1) = \tilde{\pi}(t)\tilde{P} + \pi_1(0)a$$

for finding the invariant distribution of a Markov chain [cf. Eq. (8.6) in Section 2.8]. Here $\tilde{P}$ is the matrix obtained from the transition probability matrix P of the chain after deleting the first row and the first column, and a is the row vector consisting of the elements p_{12} through p_{1n} of P. Assuming that the transition probability graph contains a positive path from every state to state 1, we have $\rho(\tilde{P}') = \rho(\tilde{P}) < 1$, as shown in Prop. 8.4 of Section 2.8. It follows that $\tilde{P}'$ corresponds to a weighted maximum norm contraction, and the above iteration fulfills the conditions for asynchronous convergence.

The following proposition shows that the condition $\rho\big(|A|\big) < 1$ is also, in effect, necessary for asynchronous convergence.

Proposition 3.1. Let A be an $n \times n$ matrix such that $I - A$ is invertible. Then the following are equivalent:

(i) $\rho\big(|A|\big) < 1$;

(ii) For any initialization $x(0)$, for any $b \in \Re^n$, for any choice of the sets T^i of computation times such that each T^i is infinite, and for any choice of the variables $\tau_j^i(t)$ satisfying $t - 2 \leq \tau_j^i(t) \leq t$ for all t, the sequence $\{x(t)\}$ produced by the asynchronous relaxation iteration (3.1)–(3.2) converges to $(I - A)^{-1}b$.

Proof. The fact that (i) implies (ii) follows from the Asynchronous Convergence Theorem of the previous section. We thus concentrate on the reverse implication. We assume that condition (i) fails to hold, and we will show that condition (ii) also fails to hold, i.e. that there is a choice of b, namely $b = 0$, a choice of $x(0)$, a choice of times $\tau_j^i(t)$, and a choice of sets T^i under which the sequence $x(t)$ produced by the algorithm does not converge to zero.

Since we are assuming that (i) fails to hold, we have $\rho\big(|A|\big) \geq 1$. The Perron–Frobenius theorem [Prop. 6.6(b) in Section 2.6] implies that there exists a vector $w \geq 0$ such that $w \neq 0$ and $|A|w \geq w$.

To explain the proof better, let us temporarily assume that $x(0)$ and $x(1)$ satisfy $x(0) \geq w$ and $x(1) \leq -w$. Suppose that T^i is the set of all nonnegative integers for each i. We define the variables $\tau_j^i(t)$ as follows:

If t is even and nonzero,

$$\tau_j^i(t) = t - 1, \quad \text{if } a_{ij} \geq 0, \tag{3.3}$$

$$\tau_j^i(t) = t - 2, \quad \text{otherwise.} \tag{3.4}$$

If t is odd,

$$\tau_j^i(t) = t - 1, \quad \text{if } a_{ij} \geq 0, \tag{3.5}$$

$$\tau_j^i(t) = t, \quad \text{otherwise.} \tag{3.6}$$

We then have

$$\begin{aligned} x_i(2) &= \sum_{\{j|a_{ij}\geq 0\}} a_{ij}x_j(0) + \sum_{\{j|a_{ij}<0\}} a_{ij}x_j(1) \\ &\geq \sum_{\{j|a_{ij}\geq 0\}} a_{ij}w_j + \sum_{\{j|a_{ij}<0\}} a_{ij}(-w_j) \\ &= \sum_{j=1}^{n} |a_{ij}|w_j \geq w_i, \end{aligned}$$

where the last inequality uses the definition of w. A similar argument yields $x_i(3) \leq -w_i$. Then the argument can be repeated inductively to show that $x_i(t) \geq w_i$ for every even t, and $x_i(t) \leq -w_i$, for every odd t. Clearly then, $x(t)$ does not converge. Nevertheless, this argument is not a proof of the desired result for the following reason. While we are free to choose $x(0)$ to be larger than w, $x(1)$ has to be generated from $x(0)$ according to Eqs. (3.1)–(3.2) and it will usually be impossible to satisfy the inequality $x(1) \leq -w$. The argument that we present in the following is essentially the same as the one just presented, except that we exercise some care in order to get around the inability to choose $x(1)$ arbitrarily.

Since $I - A$ is assumed nonsingular, there exists some $x \in \Re^n$ satisfying $Ax = x - w$. This particular x we take as the initialization $x(0)$. Again, we let each T^i be the set of all nonnegative integers. We let $\tau_j^i(t)$ be defined by (3.3)–(3.6), for $t \geq 1$. We also let $\tau_j^i(0) = 0$. Notice that with this choice of $\tau_j^i(0)$, we have $x(1) = Ax(0)$, which is equal to $x(0) - w$, because of the way we chose $x(0)$. We will now demonstrate that $x(t)$ does not converge by showing that $x(t) - x(t+1) \geq w$ for every even t. We prove this statement by induction on the set of nonnegative even integers. This statement is obviously true for $t = 0$, because $x(0) - x(1) = w$. Let us assume that t is even and $x(t) - x(t+1) \geq w$; we will then show that $x(t+2) - x(t+3) \geq w$.

We have

$$x_i(t+2) = \sum_{\{j|a_{ij}\geq 0\}} a_{ij}x_j(t) + \sum_{\{j|a_{ij}<0\}} a_{ij}x_j(t+1).$$

Similarly,

$$x_i(t+3) = \sum_{\{j|a_{ij}\geq 0\}} a_{ij}x_j(t+1) + \sum_{\{j|a_{ij}<0\}} a_{ij}x_j(t).$$

Subtracting these two equations, we get

$$\begin{aligned} x_i(t+2) - x_i(t+3) &= \sum_{\{j|a_{ij}\geq 0\}} a_{ij}\big[x_j(t) - x_j(t+1)\big] - \sum_{\{j|a_{ij}<0\}} a_{ij}\big[x_j(t) - x_j(t+1)\big] \\ &= \sum_{j=1}^{n} |a_{ij}|\big[x_j(t) - x_j(t+1)\big] \geq \sum_{j=1}^{n} |a_{ij}|w_j \geq w_i \end{aligned}$$

as desired. This completes the induction and the proof is complete. **Q.E.D.**

The proposition shows that "delays" $t - \tau_j^i(t)$ as small as two are sufficient to induce divergence when $\rho(|A|) \geq 1$, even if $\rho(A) < 1$. Notice that the choice of $\tau_j^i(t)$ used in the proof to construct a nonconvergent sequence $x(t)$ does not have the property $\tau_i^i(t) = t$. This may be unnatural in certain contexts, particularly in message–passing systems. It would be therefore preferable if we were able to construct for every matrix A with $\rho(|A|) \geq 1$, an asynchronously generated divergent sequence under the constraint $\tau_i^i(t) = t$. This turns out to be impossible, however, when there is a fixed bound on the delays $t - \tau_j^i(t)$, as will be seen in Chapter 7. In other words, when $\tau_i^i(t) = t$, the required size of $t - \tau_j^i(t)$, for $i \neq j$, to demonstrate nonconvergence may depend on the entire matrix A and not just on $\rho(|A|)$; see Example 3.1 in the next subsection and Example 1.3 in Section 7.1.

6.3.2 Unconstrained Optimization

We now consider asynchronous algorithms for unconstrained optimization. We concentrate on the case of a quadratic cost function F and the gradient method. Our conclusions can be extended to the case of a continuously differentiable convex function F for which the mapping $f(x) = x - \gamma\nabla F(x)$ is a weighted maximum norm contraction for some $\gamma > 0$ (see Prop. 1.11 and Exercise 1.3 in Section 3.1). Similarly, they can be extended to the case of a nonlinear Jacobi method (Prop. 2.6 in Subsection 3.2.4). These extensions are left as exercises for the reader.

The problem is

$$\begin{aligned} &\text{minimize} && F(x) = \tfrac{1}{2}x'Ax - b'x \\ &\text{subject to} && x \in \Re^n, \end{aligned}$$

where A is an $n \times n$ positive definite symmetric matrix and $b \in \Re^n$ is a given vector. Consider the gradient iteration

$$x := x - \gamma \nabla F(x) = x - \gamma(Ax - b),$$

or

$$x := (I - \gamma A)x + \gamma b,$$

where γ is a positive scalar stepsize. Note that for γ sufficiently small, the synchronous version of this iteration is convergent (see Section 3.2), so we have $\rho(I - \gamma A) < 1$.

To guarantee asynchronous convergence, it is sufficient that γ and A are such that $I - \gamma A$ is a weighted maximum norm contraction. This will be true in particular if the maximum row sum of $|I - \gamma A|$ is less than 1:

$$|1 - \gamma a_{ii}| + \sum_{\{j|j \neq i\}} \gamma |a_{ij}| < 1, \qquad i = 1, \ldots, n.$$

This will be so if

$$\gamma \leq \frac{1}{a_{ii}}, \qquad \forall\, i, \tag{3.7}$$

and

$$a_{ii} > \sum_{\{j|j \neq i\}} |a_{ij}|, \qquad \forall\, i. \tag{3.8}$$

The requirement of Eq. (3.7) places an upper bound on the stepsize as in the synchronous case (Section 3.2). The requirement of Eq. (3.8) is a *diagonal dominance* condition on A. Generally, some kind of diagonal dominance condition is needed in nonlinear unconstrained optimization to be able to assert that the gradient iteration mapping is a weighted maximum norm contraction, thereby establishing asynchronous convergence (see Subsection 3.1.3).

The following example shows what can happen if the diagonal dominance condition is violated.

Example 3.1.

Consider the problem

$$\min_{x \in \Re^3} \tfrac{1}{4}\left[(x_1 + x_2 + x_3)^2 + (3 - x_1 - x_2 - x_3)^2 + 2\epsilon(x_1^2 + x_2^2 + x_3^2)\right], \tag{3.9}$$

where

$$0 < \epsilon < 1.$$

For this problem, we have

$$A = \begin{bmatrix} 1+\epsilon & 1 & 1 \\ 1 & 1+\epsilon & 1 \\ 1 & 1 & 1+\epsilon \end{bmatrix},$$

so the diagonal dominance condition (3.8) is violated. Consider now the asynchronous gradient method in a message–passing system where there is a processor assigned to each coordinate. Each processor executes the gradient iteration with a fixed stepsize γ satisfying $0 < \gamma < 1/(1+\epsilon)$ [cf. Eq. (3.7)]. Suppose that we have initially

$$x_1(0) = x_2(0) = x_3(0) = c,$$

for some constant c. Let all processors execute their gradient iteration an equal and very large number of times without receiving any message from the other processors. In doing so, processor i, in effect, is solving the problem

$$\min_{x_i} \tfrac{1}{4}\left[(x_i + 2c)^2 + (3 - x_i - 2c)^2 + 2\epsilon(x_i^2 + 2c^2)\right],$$

that is, the single coordinate problem resulting when the other coordinates are set to the constant c. The solution of this problem is calculated to be

$$x_i = \frac{3}{2(1+\epsilon)} - \frac{2}{(1+\epsilon)}c.$$

After sufficiently many iterations, the coordinates x_i will be arbitrarily close to this value, so if after many iterations, the processors exchange the values of their coordinates, they will all have $x_i = \bar{c}$, with

$$\bar{c} \approx \frac{3}{2(1+\epsilon)} - \frac{2}{(1+\epsilon)}c.$$

For ϵ in the interval $(0, 1)$, this iteration is unstable, so if we repeat the process of a large number of gradient iterations followed by a single interprocessor communication, we will obtain a growing oscillation of the coordinates stored in the processors' memories. This example illustrates that an asynchronous gradient iteration, under a particular sequence of events, can be very similar to the nonlinear Jacobi method of Subsection 3.2.4. In particular, if the nonlinear Jacobi method diverges, then convergence of the asynchronous gradient iteration is very likely to fail.

Note that in the preceding example, we do not have $\tau_i^i(t) < t$, which was the characteristic feature of the example of the proof of Prop. 3.1. The difficulty comes from the fact that the "delays" $t - \tau_j^i(t)$, for $i \neq j$, can be arbitrarily large, in conjunction with the fact $\rho\big(|I - \gamma A|\big) > 1$ for all $\gamma > 0$. It turns out that if we imposed a bound $t - \tau_j^i(t) \leq B$ on the delays, we would be able to find sufficiently small $\bar{\gamma}(B) > 0$ such that for $0 < \gamma \leq \bar{\gamma}(B)$, the gradient iteration converges to the correct solution. The upper bound $\bar{\gamma}(B)$ and the speed of convergence depends on B. An analysis of asynchronous

gradient methods when $\tau_i^j(t) = t$ and the communication delays $t - \tau_j^i(t)$ are bounded will be given in Chapter 7. The preceding example in particular will be reconsidered under a boundedness assumption on the communication delays (see Example 1.3 in Section 7.1).

6.3.3 Constrained Optimization and Variational Inequalities

Consider the problem of finding a solution $x^* \in X$ of the variational inequality

$$(x - x^*)' f(x^*) \geq 0, \qquad \forall\, x \in X, \tag{3.10}$$

where X is the Cartesian product

$$X = X_1 \times X_2 \times \cdots \times X_m \tag{3.11}$$

of nonempty closed convex sets $X_i \subset \Re^{n_i}$, $i = 1, \ldots, m$, and $f : \Re^n \mapsto \Re^n$, $n = n_1 + \cdots + n_m$, is a given function. When f is the gradient of some convex function F, that is, $f(x) = \nabla F(x)$, the variational inequality is equivalent to the optimization problem $\min_{x \in X} F(x)$ (cf. Section 3.5).

We discussed in Subsection 3.5.5 the variational inequality (3.10) and, under certain conditions on f (in effect, generalized diagonal dominance conditions), we proved convergence of several parallelizable algorithms. These are the linearized algorithm (Prop. 5.8, Subsection 3.5.5), the projection algorithm (Prop. 5.9, Subsection 3.5.5), and their nonlinear counterparts (Props. 5.11 and 5.12, Subsection 3.5.6).

In all of these algorithms, the convergence proof consists of establishing that under certain assumptions the corresponding algorithmic mapping, call it T, is a special type of pseudocontraction. In particular, T satisfies

$$\|T(x) - x^*\| \leq \alpha \|x - x^*\|, \qquad \forall\, x \in X,$$

where x^* is the unique solution of the variational inequality (3.10), $\alpha \in (0, 1)$ is some scalar, $\|\cdot\|$ is the block maximum norm $\|x\| = \max_{i=1,\ldots,m} \|x_i\|_i$ with $\|\cdot\|_i$ being some norm on $\Re^{n_i}$, and $x_i \in \Re^{n_i}$ is the ith component of x. By choosing the sets $X(k)$ as

$$X(k) = \left\{ x \in \Re^n \mid \|x - x^*\| \leq \alpha^k \|x(0) - x^*\| \right\},$$

it follows that the Synchronous Convergence and Box Conditions of Assumption 2.1 are satisfied. Therefore, the corresponding iterations converge as desired when executed asynchronously.

6.3.4 Dynamic Programming

Consider the case of a Markovian decision problem of the type discussed in Section 4.3. Here we have [cf. Eq. (3.6) in Section 4.3]

$$f(x) = T(x) = \min_{\mu \in M} \left[c(\mu) + \alpha P(\mu) x \right],$$

where $c(\mu)$ and $P(\mu)$ are the cost vector and transition probability matrix, respectively, corresponding to the stationary policy $\{\mu, \mu, \ldots\}$, and α is the discount factor. We saw in Section 4.3 that T is a contraction mapping with respect to the maximum norm when $0 < \alpha < 1$ (Prop. 3.1 of Section 4.3); see also Exercise 3.3 in Section 4.3 for the case where $\alpha = 1$. In these cases, the iteration

$$x := T(x)$$

converges to the unique fixed point of T when executed asynchronously according to the model of Section 6.1.

6.3.5 Convergence Rate Comparison of Synchronous and Asynchronous Algorithms

We now consider the convergence rate of the asynchronous fixed point algorithm

$$x_i(t+1) = f_i\Big(x_1\big(\tau_1^i(t)\big), \ldots, x_n\big(\tau_n^i(t)\big)\Big), \qquad \forall\, t \geq 0. \tag{3.12}$$

We assume that f is a contraction with respect to the maximum norm, with a unique fixed point denoted by x^*. The reader should have no difficulty in extending the results to the case of a more general (weighted) maximum norm. Our analysis can be understood in terms of the discussion of Subsection 1.4.2, and the example given in that subsection. We show that with bounded communication delays, the convergence rate is geometric and, under certain conditions, it is superior to the convergence rate of the corresponding synchronous iteration.

For simplicity, we will assume that the time required for a variable update is constant and, without loss of generality, equal to one time unit. We are primarily interested in a comparison with the corresponding synchronous algorithm in which the next iteration is executed only after all messages from the previous iteration are received. To make a fair comparison, we will assume that in the asynchronous algorithm, the processors keep computing at the maximum possible speed, while simultaneously transmitting messages to other processors. Thus, all variables are updated according to Eq. (3.12) at every time t. We also assume that the communication delays are bounded; otherwise the algorithm could be arbitrarily slow. We thus assume that there exists an integer B such that

$$t - B \leq \tau_j^i(t) \leq t, \qquad \forall\, i, j, t. \tag{3.13}$$

A characteristic feature of the asynchronous algorithm is that different variables are incorporated in the computations with different communication delays. To simplify the following analysis, we assume only two sizes of communication delays; extensions to more general cases are possible (Exercise 3.2). In particular, we assume that for each index i, there is a nonempty subset of coordinates $F(i)$ and a nonnegative integer b such that

$$0 \leq b < B,$$

$$t - \tau_j^i(t) \leq b, \qquad \forall\, t,\ i = 1, \ldots, n, \text{ and } j \in F(i). \tag{3.14}$$

The interpretation here is that the variables x_j, $j \in F(i)$, are "special" in that they are communicated "fast" to the processor updating x_i (within $b < B$ time units). An example is when each processor i updates only variable x_i, and keeps the latest value of x_i in local memory, in which case, we can take $F(i) = \{i\}$ and $b = 0$. Another example arises in a hierarchical processor interconnection network when $F(i)$ represents a cluster of processors within which communication is fast.

We will consider the following two conditions:

Assumption 3.1. There exist scalars $\alpha \in [0, 1)$ and $A \in [0, 1)$ such that

$$|f_i(x) - x_i^*| \leq \max \left\{ \alpha \max_{j \in F(i)} |x_j - x_j^*|,\ A \max_{j \notin F(i)} |x_j - x_j^*| \right\}, \qquad \forall\, x \in \Re^n,\ i = 1, \ldots, n. \tag{3.15}$$

Assumption 3.2. There exist scalars $\alpha \geq 0$ and $A \geq 0$ with $\alpha + A < 1$, and such that

$$|f_i(x) - x_i^*| \leq \alpha \max_{j \in F(i)} |x_j - x_j^*| + A \max_{j \notin F(i)} |x_j - x_j^*|, \qquad \forall\, x \in \Re^n,\ i = 1, \ldots, n. \tag{3.16}$$

When f is linear of the form $f(x) = Qx + b$, where Q is an $n \times n$ matrix with elements denoted q_{ij}, then Eq. (3.16) holds with

$$\alpha = \max_{i=1,\ldots,n} \sum_{j \in F(i)} |q_{ij}|, \qquad A = \max_{i=1,\ldots,n} \sum_{j \notin F(i)} |q_{ij}|.$$

The primary interest is in the case $A < \alpha$, in which case there is "strong coupling" between x_i and the "special" variables x_j, $j \in F(i)$, which are communicated "fast" to the processor updating x_i [cf. Eq. (3.14)].

The following proposition gives a bound on the convergence rate of the asynchronous iteration (3.12):

Proposition 3.2. The sequence of vectors generated by the asynchronous iteration (3.12) satisfies

$$\|x(t) - x^*\|_\infty \leq \rho_A^t \|x(0) - x^*\|_\infty \tag{3.17}$$

where:

(a) If Assumption 3.1 holds, ρ_A is the unique nonnegative solution of the equation

$$\rho = \max \left\{ \alpha \rho^{-b},\ A \rho^{-B} \right\}. \tag{3.18}$$

(b) If Assumption 3.2 holds, ρ_A is the unique nonnegative solution of the equation

$$\rho = \alpha\rho^{-b} + A\rho^{-B}. \tag{3.19}$$

Proof. Let Assumption 3.1 hold. We use induction. For $t = 0$, the desired relation (3.17) holds. Assume that it holds for all t up to some $\bar{t}$. Since there is an update at $\bar{t}$ by assumption, we have

$$x_i(\bar{t}+1) = f_i\Big(x_1\big(\tau_1^i(\bar{t})\big), \ldots, x_n\big(\tau_n^i(\bar{t})\big)\Big).$$

Therefore, using the assumptions (3.14), (3.15), (3.18) and the induction hypothesis

$$\begin{aligned}|x_i(\bar{t}+1) - x_i^*| &\leq \max\left\{\alpha \max_{j\in F(i)} |x_j\big(\tau_j^i(\bar{t})\big) - x_j^*|,\ A \max_{j\notin F(i)} |x_j\big(\tau_j^i(\bar{t})\big) - x_j^*|\right\} \\ &\leq \max\left\{\alpha\rho_A^{\bar{t}-b},\ A\rho_A^{\bar{t}-B}\right\}\|x(0) - x^*\|_\infty = \rho_A^{\bar{t}+1}\|x(0) - x^*\|_\infty,\end{aligned}$$

and the induction proof is complete. The proof under Assumption 3.2 is entirely similar and is omitted. **Q.E.D.**

Figure 6.3.1 illustrates the method for determining the asynchronous convergence rate coefficient ρ_A and compares it with ρ_S, which is the corresponding coefficient for the synchronous version of the fixed point iteration (3.12). It can be seen that $\rho_A < \rho_S$ if $A < \alpha$ and Assumption 3.1 holds or if $0 < \alpha$ and Assumption 3.2 holds. In these cases the convergence rate estimate of the asynchronous algorithm is superior to that of its synchronous counterpart. Its communication complexity, however, can be worse, as discussed in Subsection 1.4.2.

EXERCISES

3.1. Use the function $f : \Re^2 \mapsto \Re^2$ defined by $f_1(x_1, x_2) = 0$ and $f_2(x_1, x_2) = x_1x_2$ to construct a counterexample to the following statement (which is true when f is linear):
If $X = \Re^n$, f is continuous and has a unique fixed point x^* within $\Re^n$, and all the sequences $\{x(t)\}$ generated by the asynchronous iteration (1.1) converge to x^*, then there must exist $w > 0$ such that

$$\|f(x) - x^*\|_\infty^w < \|x - x^*\|_\infty^w, \qquad \forall\ x \neq x^*.$$

3.2. For each i consider a partition $F_1(i) \cup \cdots \cup F_m(i)$ of the index set $\{1, \ldots, n\}$, where $m \geq 2$ is some integer, and suppose that for some nonnegative integers $b_1, \ldots, b_m$, we have

$$t - \tau_j^i(t) \leq b_k, \qquad \forall\ t,\ i = 1, \ldots, n,\ \text{and}\ j \in F_k(i).$$

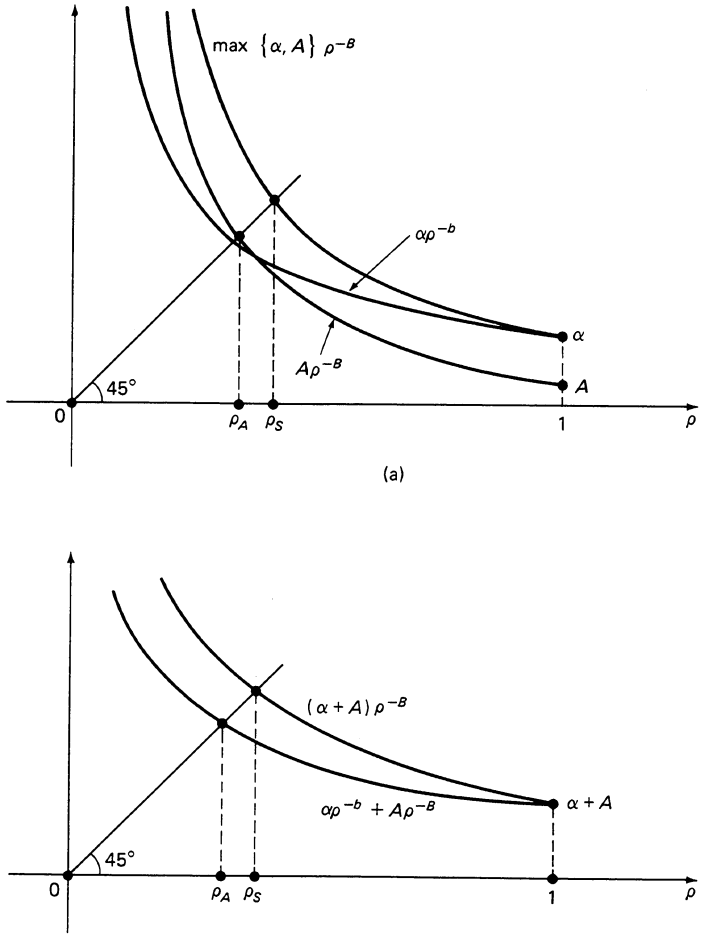


Figure 6.3.1 Comparison of the convergence rates of the asynchronous iteration (3.12) and its synchronous counterpart under (a) Assumption 3.1 and (b) Assumption 3.2. The synchronous algorithm executes an iteration every $(B + 1)$ time units, after the results of all updates from the previous iteration are known to all processors. Its convergence rate coefficient is $\rho_S = \big(\max\{\alpha, A\}\big)^{1/(B+1)}$ under Assumption 3.1, and $\rho_S = (\alpha + A)^{1/(B+1)}$ under Assumption 3.2. The convergence rate of the asynchronous algorithm is superior when $A < \alpha$ and Assumption 3.1 holds or $0 < \alpha$ and Assumption 3.2 holds.

Provide an analog of Prop. 3.2 under the assumption

$$|f_i(x) - x_i^*| \leq \max_{k=1,\ldots,m} \Big\{\alpha_k \max_{j \in F_k(i)} |x_j - x_j^*|\Big\}, \qquad i = 1, \ldots, n,$$

where $\alpha_k \in [0, 1)$, and under the assumption

$$|f_i(x) - x_i^*| \leq \sum_{k=1}^{m} \alpha_k \max_{j \in F_k(i)} |x_j - x_j^*|,$$

where $\sum_{k=1}^{m} \alpha_k < 1$, $\alpha_k \geq 0$.

6.4 APPLICATIONS TO MONOTONE MAPPINGS AND THE SHORTEST PATH PROBLEM

In this section, we show how the Asynchronous Convergence Theorem can be applied to fixed point iterations involving monotone mappings. A special case is the Bellman–Ford algorithm for the shortest path problem discussed in Section 4.3.

Consider the asynchronous algorithmic model of Section 6.1. Suppose that each set X_i is a subset of $[-\infty, +\infty]$, and that f is monotone in the sense that for all x and y in the set $X = \prod_{i=1}^{n} X_i$ we have

$$x \leq y \quad \Longrightarrow \quad f(x) \leq f(y), \tag{4.1}$$

where the inequalities are interpreted coordinatewise.

Suppose also that there exists a unique fixed point x^* of f in X and two vectors $\underline{x}$ and $\bar{x}$ in X such that

$$\underline{x} \leq f(\underline{x}) \leq f(\bar{x}) \leq \bar{x}, \tag{4.2}$$

and

$$\lim_{k \to \infty} f^k(\underline{x}) = \lim_{k \to \infty} f^k(\bar{x}) = x^*. \tag{4.3}$$

In the preceding equation $f^k(\cdot)$ is the composition of f with itself k times, and $f^0(\cdot)$ is the identity mapping. The monotonicity condition (4.2) implies that

$$f^k(\underline{x}) \leq f^{k+1}(\underline{x}) \leq x^* \leq f^{k+1}(\bar{x}) \leq f^k(\bar{x}), \qquad \forall\, k,$$

and, therefore, also implies the convergence condition (4.3) if $X \subset R^n$ and f is a continuous function on X.

Define the sets

$$X(k) = \left\{x \mid f^k(\underline{x}) \leq x \leq f^k(\bar{x})\right\}. \tag{4.4}$$

It is seen that the Synchronous Convergence and Box Conditions of Assumption 2.1 are satisfied, and, therefore, the Asynchronous Convergence Theorem applies.

The main difficulty in using the theorem is to establish the existence of vectors $\underline{x}$ and $\bar{x}$ for which the monotonicity and convergence conditions (4.2) and (4.3) hold. To prove this, one must use special features of the problem at hand.

One situation where the underlying mapping is monotone arises in network flow problems. As an example, consider the constrained matrix problem of Subsection 5.5.4. The relaxation mappings corresponding to the quadratic and logarithmic cost functions given there can be seen to be monotone. As a result, asynchronous convergence can be shown for a modified version of the relaxation method of Section 5.5. This will be shown in more general form in Section 6.6.

The underlying mapping is also monotone in the Markovian decision problems of Section 4.3. Based on the Undiscounted Cost Assumption 3.2 and Props. 3.1 and 3.3 of that section, it is seen that the monotonicity and convergence conditions (4.2) and (4.3) are satisfied for the undiscounted problem with

$$\underline{x}_i = x_i^* - \Delta, \qquad \bar{x}_i = x_i^* + \Delta, \qquad i = 2, \ldots, n, \tag{4.5a}$$

$$\underline{x}_1 = x_1 = \bar{x}_1 = 0, \tag{4.5b}$$

where x_i^* is the optimal cost starting at state i, and Δ is any positive scalar. In the case of a shortest path problem we can also choose $\bar{x}_i = \infty$ for $i \neq 1$ (see Prop. 1.1 in Section 4.1). Therefore, the Asynchronous Convergence Theorem applies, showing that the dynamic programming algorithm converges from arbitrary initial conditions when executed asynchronously. We now discuss the special case of the Bellman–Ford algorithm for the shortest path problem in more detail.

The Shortest Path Problem

Consider the shortest path problem under the Connectivity and Positive Cycle Assumptions 1.1 and 1.2 of Section 4.1 (node 1 is the only destination). It was seen in that section that if the initial condition in the Bellman–Ford algorithm

$$x_i := \min_{j \in A(i)} (a_{ij} + x_j), \qquad i = 2, \ldots, n, \tag{4.6a}$$

$$x_1 = 0, \tag{4.6b}$$

is chosen to be equal to either the vector $\underline{x}$ or the vector $\bar{x}$ of Eq. (4.5), then convergence is obtained in a finite number of iterations. Therefore, for all k sufficiently large, the sets $X(k)$ of Eq. (4.4) consist of just the shortest distance vector x^*. It follows from the Asynchronous Convergence Theorem that the asynchronous version of the Bellman–Ford iteration

$$x_i(t+1) = \min_{j \in A(i)} \left(a_{ij} + x_j\left(\tau_j^i(t)\right) \right), \qquad i = 2, \ldots, n, \ t \in T^i,$$

$$x_1(t+1) = 0,$$

terminates with the correct shortest distances in finite time.

We will now consider an implementation of the asynchronous Bellman–Ford algorithm, which is of particular relevance to communication networks; see Chapter 5

of [BeG87]. We will compare this algorithm with its synchronous counterpart for the initial conditions $\bar{x}_i = \infty$, $i \neq 1$, in terms of the amount of time, and communication needed to solve the problem. We assume that at each node $i \neq 1$, there is a processor updating x_i according to the Bellman–Ford iteration (4.6a). The value of x_i available at node i at time t is denoted by $x_i(t)$. For each arc (i, j), $j \in A(i)$, there is a communication link along which j can send messages to i. When x_j is updated and, as a result, its value actually changes, the new value of x_j is immediately sent to every processor i for which (i, j) is an arc. If the updating leaves the value of x_j unchanged, no communication takes place. Let t_{ij} be the time required for a message to traverse arc (i, j) in the direction from j to i. We assume that this time is the same for all messages that traverse (i, j), and, in particular, that the messages are received on each arc in the order they are sent. When a processor i receives a message containing a value of x_j, for some $j \in A(i)$, it stores it in place of the preexisting value of x_j and updates x_i according to the Bellman–Ford iteration (4.6a) using the values of $x_{j'}$ latest received from all $j' \in A(i)$ (if no value of $x_{j'}$ has been received as yet, node i uses $x_{j'} = \infty$ for $j' \neq 1$ and $x_1 = 0$ for $j' = 1$). It is assumed that an update requires negligible time, modeling a situation where an update requires much less time than a message transmission. We adopt the convention that if there is an update at node j at time t, $x_j(t)$ is the value of x_j just *after* the update. Regarding initialization, we assume that each node j sends at time 0 its initial estimate

$$x_j(0) = \begin{cases} 0, & \text{if } j = 1, \\ \infty, & \text{otherwise,} \end{cases} \tag{4.7}$$

to all nodes i with $j \in A(i)$. We say that the algorithm terminates at time t if, for each i, $x_i(t)$ is equal to the shortest distance x_i^* from i to 1. Because of the preceding choice of initial conditions and the monotonicity of the Bellman–Ford iteration (4.6), it is seen that the estimates $x_i(t)$ are monotonically nonincreasing to their final values x_i^*.

To visualize the asynchronous algorithm, it is helpful to consider (somewhat informally) an equivalent algorithmic process. Imagine, for each path p that ends at node 1, a "token" that travels along the path in the reverse direction, starting from node 1 at time 0, and requiring the same time to traverse each link as any other message. If the token reaches a node i on the path at time t, and the sum of the lengths of the arcs that it has traversed is lower than the estimate of the shortest distance of i at the time of arrival, then $x_i(t)$ is set equal to this sum and the token is allowed to proceed to the next node on the path (if any); otherwise the token is discarded and its travel is stopped. (If more than one token arrives at a node simultaneously, only the ones with smallest sum of arc lengths are allowed to proceed to the next node on their respective paths, assuming that they cause a reduction of the estimate of shortest distance of the node.) It can be seen then that link traversals by the tokens can be associated, on a one–to–one basis, with transmissions of messages carrying shortest distance estimates in the asynchronous Bellman–Ford algorithm. Furthermore, for each node i, $x_i(t)$ is equal to the minimum length over all lengths of paths that terminate at i and whose tokens have arrived at i at some time $t' \leq t$.

We denote the length of a path p consisting of arcs $(i_1, i_2), \ldots, (i_k, i_{k+1})$ by

$$L_p = \sum_{m=1}^{k} a_{i_m i_{m+1}}, \tag{4.8}$$

and we define the communication time of p as

$$t_p = \sum_{m=1}^{k} t_{i_m i_{m+1}}. \tag{4.9}$$

Denote for any $i \neq 1$

$$P_i(t) = \text{Set of paths } p \text{ with } t_p \leq t \text{ that start at } i \text{ and end at } 1. \tag{4.10}$$

From the preceding observations, it is seen with a little thought that the shortest distance estimate of node i at time t is

$$x_i(t) = \begin{cases} \infty, & \text{if } P_i(t) \text{ is empty,} \\ \min_{p \in P_i(t)} L_p, & \text{otherwise.} \end{cases} \tag{4.11}$$

Consider now a synchronous version of the algorithm. Here each processor i performs the $(k+1)$st update immediately upon receiving the results of the kth update from all processors $j \in A(i)$, that is, the local synchronization method of Subsection 1.4.1 is used. Note that we require that each processor j send the result of each update to all processors i for which $j \in A(i)$, whether or not this result is different from the result of the previous update. For $i \neq 1$, let P_i^k be the set of paths that start at i, end at 1, and have k arcs or less. Denote

$$k_i(t) = \max\{k \mid t_p \leq t, \ \forall\, p \in P_i^k\}. \tag{4.12}$$

Thus, $k_i(t)$ is the number of synchronous Bellman–Ford iterations reflected by the shortest distance estimate of i at time t. Then it is seen (Exercise 4.3) that the shortest distance estimate at node i at time t for the synchronous algorithm is

$$x_i^S(t) = \begin{cases} \infty, & \text{if } P_i^{k_i(t)}(t) \text{ is empty,} \\ \min_{p \in P_i^{k_i(t)}(t)} L_p, & \text{otherwise.} \end{cases} \tag{4.13}$$

We now observe that for all i and t, we have $P_i^{k_i(t)} \subset P_i(t)$. The reason is that for each path $p \in P_i^{k_i(t)}$, we have from Eq. (4.12) $t_p \leq t$, which, using Eq. (4.10), implies that $p \in P_i(t)$. It follows from Eqs. (4.11) and (4.13) that

$$x_i(t) \leq x_i^S(t), \qquad \forall\, t,$$

and hence the synchronous algorithm cannot terminate faster than the asynchronous algorithm.

Unfortunately, the asynchronous Bellman–Ford algorithm can require many more message transmissions than synchronous versions of the algorithm. Figures 6.4.1 and

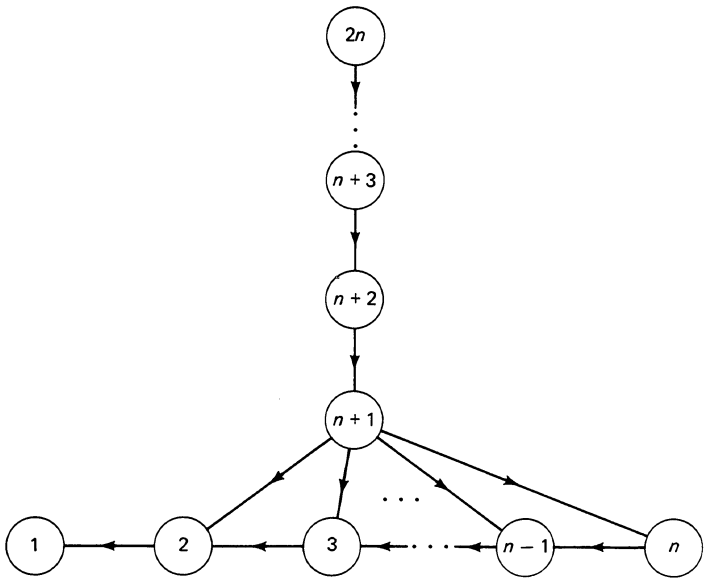


Figure 6.4.1 An example where the asynchronous version of the Bellman–Ford algorithm requires many more message transmissions than a synchronous version (from [BeG87]). All arc lengths are unity. The initial estimates of shortest distance of all nodes are ∞. Consider the following sequence of events:

1. Node 2 updates its shortest distance and communicates the result to node 3. Node 3 updates and communicates the result to 4.... Node $n-1$ updates and communicates the result to node n. Node n updates and communicates the result to $n+1$. (These are $n-1$ messages.)
2. Node $n+1$ updates and communicates the result to node $n+2$.... Node $2n-1$ updates and communicates the result to node $2n$. Node $2n$ updates. (These are $n-1$ messages.)
3. For $i = n-1, n-2, \ldots, 2$, in that order, node i communicates its update to node $n+1$ and the sequence of step 2 is repeated. [These are $n(n-2)$ messages.]

The total number of messages is $n^2 - 2$. Only $\Theta(n)$ messages would be needed in a synchronous version of the algorithm where the global synchronization method with timeouts is used (cf. Subsection 1.4.1), and where a message is sent by a node j to all nodes i with $j \in A(i)$ only when an update changes the value x_j. The difficulty in the asynchronous version is that a great deal of unimportant information arrives at node $n+1$ early and triggers many unnecessary messages starting from $n+1$ and proceeding all the way to $2n$. The total number of distance changes in this example is $\Theta(n^2)$. Generally, for a shortest path problem with n nodes and unity arc lengths it can be shown that in the asynchronous Bellman–Ford algorithm with $x_i(0) = \infty$, $i \neq 1$, there can be at most $n-1$ distance changes for each node (Exercise 4.2). By contrast, only one distance change per node is needed in the synchronous version of the algorithm.

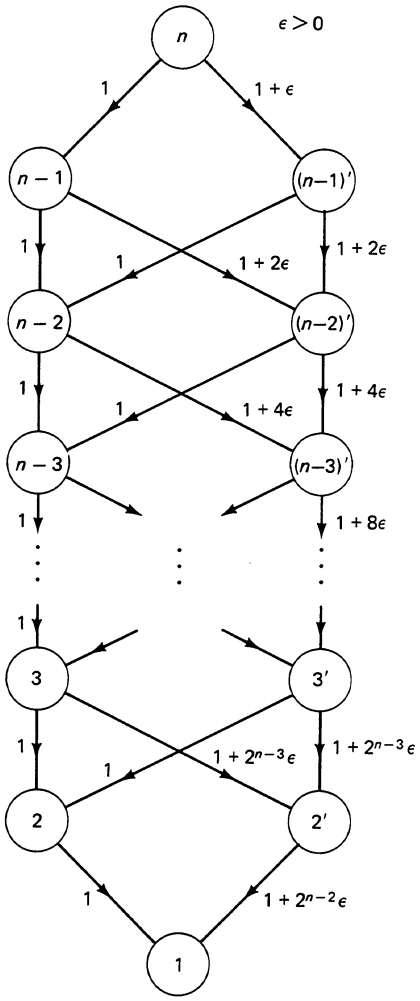


Figure 6.4.2 An example of a shortest path problem where the asynchronous Bellman–Ford algorithm requires an exponentially large number of messages for termination starting from the initial condition $x_i(0) = \infty$, $i \neq 1$ (due to E. M. Gafni and R. G. Gallager, a private communication). The arc lengths are shown next to the arcs. Suppose that the communication delays of the arcs are such that the communication time of a path from each node i to node 1 is larger as the length of the path is smaller, that is, for all paths p and p', we have

$$L_p < L_{p'} \quad \Rightarrow \quad t_p > t_{p'},$$

(such a choice of arc delays is possible for this example). Then the distance $x_n(t)$ of node n will change a number of times equal to the number of paths from n to 1, which is 2^{n-2}.

6.4.2 provide two examples. Both examples represent worst case (and rather unlikely) scenarios.

EXERCISES

4.1. This exercise comes from packet radio communications ([BeG87], p. 277). We have a set of n transmitters that transmit data in intervals of unit time (also called *slots*). Each transmitter i transmits in a slot with probability p_i independently of the history of past transmissions. For each transmitter i, there is a set S_i of other transmitters such that if i transmits simultaneously with one or more transmitters in S_i, the transmission is unsuccessful. Then the success rate (or throughput) of transmitter i is

$$t_i = p_i \prod_{j \in S_i} (1 - p_j).$$

We want to find the probabilities p_i, $i = 1, \ldots, n$, that realize a given set of throughputs t_i, $i = 1, \ldots, n$, that is, solve the system of the above n nonlinear equations for the n unknowns p_j. For this, we use the iteration

$$p_i := \frac{t_i}{\prod_{j \in S_i}(1 - p_j)}$$

implemented in totally asynchronous format. Show that, if there exists a set of feasible probabilities, the iteration will converge to a unique such set of probabilities when initiated at $p_i = 0$ for all i.

4.2. Consider the asynchronous Bellman–Ford algorithm for an n–node shortest path problem with all arc lengths being unity. Assume that initially $x_i(0) = \infty$ for all $i \neq 1$. Show that there are less than n changes of x_i at each node i. *Hint:* Each node's estimate of shortest distance is nonincreasing and can only take the values $1, 2, \ldots, n-1$ and ∞.

4.3. Consider the synchronous Bellman–Ford algorithm with the local synchronization method, and show Eq. (4.13). *Hint:* Define

$$t_i(0) = 0, \qquad \forall\, i = 1, \ldots, n,$$

$$t_i(k+1) = \begin{cases} \max_{\{j \mid (i,j) \in A\}} \{t_{ij} + t_j(k)\}, & \text{if } i \neq 1 \\ 0, & \text{otherwise.} \end{cases}$$

Show that each node $i \neq 1$ will execute its kth iteration at time $t_i(k)$.

4.4. **(Asynchronous Forward Search.)** Consider an asynchronous version of the forward search shortest path algorithm of Exercise 1.10(a) of Section 4.1 (we use the same notation as in that exercise). In this version, d_j is updated at times $t \in T^j$ according to

$$d_j(t+1) = \begin{cases} \min_{i \in L^j(t)} \left(d_i\left(\tau_i^j(t)\right) + a_{ij} \right), & \text{if } \min_{i \in L^j(t)} \left(d_i\left(\tau_i^j(t)\right) + a_{ij} \right) \\ & \qquad < \min \left\{ d_j(t), d_1\left(\tau_1^j(t)\right) - h_j \right\}, \\ d_j(t), & \text{otherwise,} \end{cases} \tag{4.14}$$

where $L^j(t)$ is a subset of the set $\{i \mid j \in A(i)\}$. Assume that $d_s(0) = 0$ and $d_j(0) = \infty$ for all $j \neq s$. Assume also that for every j, each node $i \in \{k \mid j \in A(k)\}$ belongs to infinitely many sets $L^j(t)$, $t \in T^j$ and that the Total Asynchronism Assumption 1.1 holds. [Note, however, that we have $\tau_j^j(t) = t$ in Eq. (4.14).] Show that $d_1(t)$ will be equal to the shortest distance from s to 1 for all sufficiently large t. *Hint:* Show that for each i, $d_i(t)$ is monotonically nonincreasing and can take only a finite number of values.

6.5 LINEAR NETWORK FLOW PROBLEMS

In this section, we discuss some of the asynchronous distributed implementation aspects of the linear network flow problems of Chapter 5. There is a natural asynchronous version of the auction algorithm of Subsection 5.3.1 in an environment where there is

a processor assigned to each person and a processor assigned to each object, and the processors communicate with each other via an interconnection network. Here there is no strict separation between the bidding and the assignment phases, and each unassigned person/processor can bid at arbitrary times on the basis of object price information that can be outdated because of additional bidding of which the person is not informed. Furthermore, the assignment of objects can be decided even if some potential bidders have not been heard from. We can show the same termination properties as for the synchronous version of the algorithm subject to two conditions stated somewhat informally:

(a) An unassigned person/processor will bid for some object within finite time and cannot simultaneously bid for more than one object (i.e., it cannot bid for a second object while waiting for a response regarding the disposition of an earlier bid for another object).

(b) Whenever one or more bids are received that raise the price of an object, then within finite time, the updated price of the object must be communicated (not necessarily simultaneously) to all persons that can be assigned to the object. Furthermore, the new person that is assigned to the object must be informed of this fact simultaneously with receiving the new price.

A similar implementation is possible when there are relatively few processors and each processor is assigned to several persons and objects. There is also a straightforward asynchronous implementation of the auction algorithm in a shared memory machine, which is similar to a synchronous implementation except that there is no strict separation between the bidding and the assignment phases; some processors may be calculating person bids while some other processors may be simultaneously updating object prices. We do not give a precise formulation and a proof of validity of these implementations. We will concentrate instead on the more challenging case of the general linear network flow problem

$$\text{minimize} \quad \sum_{(i,j)\in A} a_{ij} f_{ij} \tag{LNF}$$

$$\text{subject to} \quad \sum_{\{j|(i,j)\in A\}} f_{ij} - \sum_{\{j|(j,i)\in A\}} f_{ji} = s_i, \qquad \forall\, i \in N,$$

$$b_{ij} \le f_{ij} \le c_{ij}, \qquad \forall\, (i,j) \in A,$$

where a_{ij}, b_{ij}, c_{ij}, and s_i are given integers. We introduce a distributed totally asynchronous version of the ϵ–relaxation method of Section 5.3, and show that it converges finitely.

We assume that each node i is a processor that updates its own price and incident arc flows, and exchanges information with its forward adjacent nodes

$$F_i = \{j \mid (i,j) \in A\},$$

and its backward adjacent nodes

$$B_i = \{j \mid (j,i) \in A\}.$$

The following distributed asynchronous implementation applies to both the ϵ–relaxation method of Section 5.3 and to the subproblems of the scaled version discussed in Section 5.4. The information available at node i for any time t is as follows:

$p_i(t)$: The price of node i

$p_j(i,t)$: The price of node $j \in F_i \cup B_i$ communicated by j at some earlier time

$f_{ij}(i,t)$: The estimate of the flow of arc (i,j), $j \in F_i$, available at node i at time t

$f_{ji}(i,t)$: The estimate of the flow of arc (j,i), $j \in B_i$, available at node i at time t

$g_i(t)$: The estimate of the surplus of node i at time t given by

$$g_i(t) = \sum_{(j,i)\in A} f_{ji}(i,t) - \sum_{(i,j)\in A} f_{ij}(i,t) + s_i. \tag{5.1}$$

A more precise description of the algorithmic model is possible, but for brevity, we will keep our description somewhat informal. We assume that for every node i, the quantities just listed do not change except possibly at an increasing sequence of times $t_0, t_1, \ldots$, with $t_m \to \infty$. At each of these times, generically denoted by t, and at each node i, one of three events happens:

Event 1. Node i does nothing.

Event 2. Node i checks $g_i(t)$. If $g_i(t) < 0$, node i does nothing further. Otherwise node i executes either an up iteration or a partial up iteration (cf. Section 5.3) based on the available price and flow information

$$p_i(t), \qquad p_j(i,t),\ j \in F_i \cup B_i, \qquad f_{ij}(i,t),\ j \in F_i, \qquad f_{ji}(i,t),\ j \in B_i,$$

and accordingly changes

$$p_i(t), \qquad f_{ij}(i,t),\ j \in F_i, \qquad f_{ji}(i,t),\ j \in B_i.$$

Event 3. Node i receives, from one or more adjacent nodes $j \in F_i \cup B_i$, a message containing the corresponding price and arc flow $\big(p_j(t'), f_{ij}(j,t')\big)$ (in the case $j \in F_i$) or $\big(p_j(t'), f_{ji}(j,t')\big)$ (in the case $j \in B_i$) stored at j at some earlier time $t' < t$. If

$$p_j(t') < p_j(i,t),$$

node i discards the message and does nothing further. Otherwise node i stores the received value $p_j(t')$ in place of $p_j(i,t)$. In addition, if $j \in F_i$, node i stores $f_{ij}(j,t')$ in place of $f_{ij}(i,t)$ if

$$p_i(t) < p_j(t') + a_{ij}, \qquad \text{and} \qquad f_{ij}(j,t') < f_{ij}(i,t),$$

and otherwise leaves $f_{ij}(i,t)$ unchanged; in the case $j \in B_i$, node i stores $f_{ji}(j,t')$ in place of $f_{ji}(i,t)$ if

$$p_j(t') \geq p_i(t) + a_{ji}, \qquad \text{and} \qquad f_{ji}(j,t') > f_{ji}(i,t),$$

and otherwise leaves $f_{ji}(i,t)$ unchanged. (Thus, in case of a balanced arc, the "tie" is broken in favor of the flow of the start node of the arc.)

Let T^i be the set of times for which an update by node i is attempted as in Event 2, and let $T^i(j)$ be the set of times when a message is received at i from j, as in Event 3. We assume the following:

Assumption 5.1. Nodes never stop attempting to execute an up iteration, and receiving messages from all their adjacent nodes, that is, T^i and $T^i(j)$ have an infinite number of elements for all i and $j \in F_i \cup B_i$.

Assumption 5.2. Old information is eventually purged from the system, that is, given any time t_k, there exists a time $t_m \geq t_k$ such that the time of generation of the price and flow information received at any node after t_m (i.e., the time t' in Event 3), exceeds t_k.

Assumption 5.3. For each i, the initial arc flows $f_{ij}(i,t_0)$, $j \in F_i$, and $f_{ji}(i,t_0)$, $j \in B_i$, are integer and satisfy ϵ–CS together with $p_i(t_0)$ and $p_j(i,t_0)$, $j \in F_i \cup B_i$. Furthermore, there holds

$$p_i(t_0) \geq p_i(j,t_0), \qquad \forall\, j \in F_i \cup B_i,$$
$$f_{ij}(i,t_0) \geq f_{ij}(j,t_0), \qquad \forall\, j \in F_i.$$

Assumptions 5.1 and 5.2 are similar to the Total Asynchronism Assumption 1.1 of Section 6.1. One set of initial conditions satisfying Assumption 5.3 but requiring very little cooperation between processors is $p_j(i,t_0) \approx -\infty$ for all i and $j \in F_i \cup B_i$, $f_{ij}(i,t_0) = c_{ij}$ and $f_{ij}(j,t_0) = b_{ij}$ for all i and $j \in F_i$. Assumption 5.3 guarantees that for all $t \geq t_0$,

$$p_i(t) \geq p_i(j,t''), \qquad \forall\, j \in F_i \cup B_i,\ t'' \leq t. \tag{5.2}$$

To see this, note that $p_i(t)$ is monotonically nondecreasing in t and $p_i(j,t'')$ equals $p_i(t')$ for some $t' < t''$.

An important fact is that for all nodes i and times t, $f_{ij}(i,t)$ and $f_{ji}(i,t)$ are integer and satisfy ϵ–CS together with $p_i(t)$ and $p_j(i,t)$, $j \in F_i \cup B_i$. This is seen from Eq. (5.2), the logic of the up iteration, and the rules for accepting information from adjacent nodes.

Another important fact is that for all i and $t > t_0$

$$f_{ij}(i,t) \geq f_{ij}(j,t), \qquad \forall\, j \in F_i, \tag{5.3}$$

that is, the start node of an arc has at least as high an estimate of arc flow as the end node. For a given $(i,j) \in A$, condition (5.3) holds initially by Assumption 5.3 and it is preserved by up iterations since an up iteration at i cannot decrease $f_{ij}(i,t)$ and an up iteration at j cannot increase $f_{ij}(j,t)$. Therefore, Eq. (5.3) can first be violated only at the time of a message reception. To show that this cannot happen throughout the algorithm, we argue by contradiction: suppose Eq. (5.3) first fails to hold at some time t, implying

$$f_{ij}(i,t) < f_{ij}(j,t). \tag{5.4}$$

Let r be the last time up to or including t that i accepted a message from j that reduced i's flow estimate for (i,j), and let r' be the time that this message originated at j. Similarly, let s be the last time up to or including t that j increased its flow estimate for (i,j), and s' be the time the message responsible was sent from i. Using Assumption 5.3, it can be seen that both of these times must exist, for otherwise the violation of Eq. (5.4) cannot have occurred (note also that $t = \max\{r,s\}$). The acceptance of these messages at times r and s implies

$$p_i(r) < p_j(r') + a_{ij}, \tag{5.5}$$

$$p_i(s') \geq p_j(s) + a_{ij}. \tag{5.6}$$

It also follows that $r' \leq s$ and $s' \leq r$ or, again, the violation at time t could not have occurred. By the monotonicity of prices, it then follows that

$$p_i(r') \leq p_i(s), \qquad p_i(s') \leq p_i(r).$$

Substituting these into Eqs. (5.5) and (5.6), respectively, one obtains

$$p_i(r) < p_j(s) + a_{ij}, \tag{5.7a}$$

$$p_i(r) \geq p_j(s) + a_{ij}, \tag{5.7b}$$

a contradiction. Thus, Eq. (5.3) must always hold for all $(i,j) \in A$.

Note that once a node i has nonnegative surplus $g_i(t) \geq 0$, it maintains a nonnegative surplus for all subsequent times. The reason is that an up iteration at i can at most

decrease $g_i(t)$ to zero, whereas, in view of the rules for accepting a communicated arc flow, a message exchange with an adjacent node j can only increase $g_i(t)$. Note also that from Eq. (5.3) we obtain

$$\sum_i g_i(t) \leq 0, \qquad \forall\, t \geq t_0. \tag{5.8}$$

This implies that at any time t, there is at least one node i with negative surplus $g_i(t)$ if there is a node with positive surplus. This node i must not have executed any up iteration up to t, and, therefore, its price $p_i(t)$ must still be equal to the initial price $p_i(t_0)$.

We say that the algorithm *terminates* if there is a time t_k such that for all $t \geq t_k$, we have

$$g_i(t) = 0, \qquad \forall\, i \in N, \tag{5.9}$$

$$f_{ij}(i,t) = f_{ij}(j,t), \qquad \forall\, (i,j) \in A, \tag{5.10}$$

$$p_j(t) = p_j(i,t), \qquad \forall\, j \in F_i \cup B_i, \tag{5.11}$$

a flow–price vector pair satisfying ϵ–CS is then obtained. Termination can be detected by using a method to be discussed in Section 8.1.

Proposition 5.1. If problem (LNF) is feasible and Assumptions 5.1–5.3 hold, the algorithm terminates.

Proof. Suppose no up iterations are executed at any node after some time t^*. Then Eq. (5.9) must hold for large enough t. Because no up iterations occur after t^*, all the $p_i(t)$ must remain constant after t^*, and Assumption 5.1, Eq. (5.2), and the message acceptance rules imply Eq. (5.11). After t^*, furthermore, no flow estimates can change except by message reception. By Eq. (5.11), the nodes will eventually agree on whether each arc is active, inactive, or balanced. The message reception rules, Eq. (5.3), Assumptions 5.1 and 5.2 then imply the eventual agreement on arc flows as in Eq. (5.10). (Eventually, the start node of each inactive arc will accept the flow of the end node, and the end node of a balanced or active arc will accept the flow of the start node.)

We assume, therefore, the contrary, that is, that up iterations are executed indefinitely, and, hence, for every t, there is a time $t' > t$ and a node i such that $g_i(t') > 0$. There are two possibilities: the first is that $p_i(t)$ converges to a finite value p_i for every i. In this case, we assume without loss of generality that there is at least one node i at which an infinite number of up iterations are executed and an adjacent arc (i,j) whose flow $f_{ij}(i,t)$ is changed by an integer amount an infinite number of times with (i,j) being ϵ^+–balanced. For this to happen, there must be a reduction of $f_{ij}(i,t)$ through communication from j an infinite number of times. This means that $f_{ij}(j,t)$ is reduced an infinite number of times, which can happen only if an infinite number of up iterations are executed at j with (i,j) being ϵ^-–balanced. But this is impossible since, when p_i and p_j converge, arc (i,j) cannot become both ϵ^+–balanced and ϵ^-–balanced an infinite number of times.

The second possibility is that there is a nonempty subset of nodes N^∞ whose prices increase to ∞. From Eq. (5.8), it is seen that there is at least one node that has negative surplus for all t and, therefore, also a constant price. It follows that N^∞ is a strict subset of N. Since the algorithm maintains ϵ–CS, we have for all sufficiently large t that

$$f_{ij}(i,t) = f_{ij}(j,t) = c_{ij} \qquad \forall\ (i,j) \in A \ \text{ with } \ i \in N^\infty,\ j \in N^\infty,$$
$$f_{ji}(i,t) = f_{ji}(j,t) = b_{ji} \qquad \forall\ (j,i) \in A \ \text{ with } \ i \in N^\infty,\ j \in N^\infty.$$

Note now that all nodes in N^∞ have nonnegative surplus, and each must have positive surplus infinitely often. Adding Eq. (5.1) for all i in N^∞, and using both Eq. (5.3) and the preceding relations, we find that the sum of c_{ij} over all $(i,j) \in A$ with $i \in N^\infty$ and $j \in N^\infty$ is less than the sum of b_{ji} over all $(j,i) \in A$ with $i \in N^\infty$ and $j \in N^\infty$, plus the sum of s_i over $i \in N^\infty$. Therefore, there can be no feasible solution, violating the hypothesis. It follows that the algorithm must terminate. **Q.E.D.**

6.6 NONLINEAR NETWORK FLOW PROBLEMS

In this section, we consider the nonlinear network flow problem of Section 5.5

$$\text{minimize} \quad \sum_{(i,j)\in A} a_{ij}(f_{ij}) \tag{CNF}$$

$$\text{subject to} \quad \sum_{\{j|(i,j)\in A\}} f_{ij} - \sum_{\{j|(j,i)\in A\}} f_{ji} = s_i, \qquad \forall\ i \in N, \tag{6.1}$$

$$b_{ij} \le f_{ij} \le c_{ij}, \qquad \forall\ (i,j) \in A, \tag{6.2}$$

under the following assumption (made also in Section 5.5).

Assumption 6.1. (*Strict Convexity*) The functions a_{ij}, $(i,j) \in A$, are strictly convex real–valued functions, and problem (CNF) has at least one feasible solution.

Our purpose is to develop a distributed asynchronous version of the relaxation method described in Section 5.5 and to analyze its convergence properties.

We showed that the method of Section 5.5 converges appropriately when implemented in a Gauss–Seidel version. Figure 5.5.5 of Section 5.5 shows that this is not true when the method is implemented in a Jacobi version, and, therefore, it cannot be true for an asynchronous version. The difficulty is due to the structure of the optimal dual solution set; in particular, by adding the same constant to all coordinates of an optimal price vector, we still obtain an optimal price vector. We are thus motivated to consider the variation of the method where a single price, say p_1, is fixed, thereby effectively restricting the dual optimal solution set.

Consider the dual functional q given by

$$q(p) = \sum_{(i,j)\in A} q_{ij}(p_i - p_j) + \sum_{i\in N} s_i p_i, \tag{6.3}$$

where

$$q_{ij}(p_i - p_j) = \min_{b_{ij}\le f_{ij}\le c_{ij}} \{a_{ij}(f_{ij}) - (p_i - p_j)f_{ij}\}. \tag{6.4}$$

Consider also the following version of the dual problem:

$$\begin{aligned} &\text{maximize} \quad q(p) \\ &\text{subject to} \quad p \in P, \end{aligned} \tag{6.5}$$

where P is the *reduced set of price vectors*

$$P = \{p \in \Re^{|N|} \mid p_1 = c\},$$

and c is a fixed scalar. If we add the same constant to all prices, the dual function value (6.3) does not change (since, for feasibility, $\sum_{i\in N} s_i = 0$). Therefore, the constrained version (6.5) of the dual problem is equivalent to the unconstrained version considered in Section 5.5 in the sense that the optimal values of the two problems are equal, and the optimal price vectors of the latter problem are obtained from the optimal price vectors of the former by adding a multiple of the vector $(1, 1, \ldots, 1)$.

Recall the definition of the surplus of a node i:

$$g_i = \sum_{\{j|(j,i)\in A\}} f_{ji} - \sum_{\{j|(i,j)\in A\}} f_{ij} + s_i. \tag{6.6}$$

In Section 5.5, we calculated the gradient of the dual functional as

$$\frac{\partial q(p)}{\partial p_i} = g_i(p), \tag{6.7}$$

where

$$\begin{aligned} g_i(p) = {} &\text{Surplus of node } i \text{ corresponding to the unique } f \\ &\text{satisfying complementary slackness together with } p. \end{aligned} \tag{6.8}$$

For $i = 1, \ldots, |N|$, consider the point–to–set mapping R_i, which assigns to a price vector $p \in P$ the interval of all prices ξ that maximize the dual cost along the ith price starting from p, that is,

$$R_i(p) = \{\xi \mid g_i(p_1, \ldots, p_{i-1}, \xi, p_{i+1}, \ldots, p_{|N|}) = 0\}$$

$$= \left\{ \xi \Big| \sum_{\{j|(j,i)\in A\}} \nabla q_{ji}(p_j - \xi) = \sum_{\{j|(i,j)\in A\}} \nabla q_{ij}(\xi - p_j) + s_i \right\}. \tag{6.9}$$

We call $R_i(p)$ the *ith relaxation mapping*, and note that $R_i(p)$ is nonempty for all p, as shown in Section 5.5 following Assumption 5.2. We will need a more precise characterization of $R_i(p)$. To this end we note that by assumption there exists a flow vector that satisfies both the conservation of flow constraints (6.1) and the capacity constraints (6.2), thereby implying that

$$\sum_{\{j|(i,j)\in A\}} b_{ij} - \sum_{\{j|(j,i)\in A\}} c_{ji} \le s_i \le \sum_{\{j|(i,j)\in A\}} c_{ij} - \sum_{\{j|(j,i)\in A\}} b_{ji}.$$

The following lemma shows that the form of $R_i(p)$ depends on whether equality holds in the above inequalities.

Lemma 6.1. For all $p \in \Re^{|N|}$ the set of real numbers $R_i(p)$ of Eq. (6.9) is nonempty closed and convex. Furthermore, it is bounded below if and only if

$$\sum_{\{j|(i,j)\in A\}} b_{ij} - \sum_{\{j|(j,i)\in A\}} c_{ji} < s_i,$$

and it is bounded above if and only if

$$s_i < \sum_{\{j|(i,j)\in A\}} c_{ij} - \sum_{\{j|(j,i)\in A\}} b_{ji}.$$

Proof. We have for all i and p,

$$\lim_{\xi\to-\infty} g_i(p_1, \ldots, p_{i-1}, \xi, p_{i+1}, \ldots, p_{|N|}) = \sum_{\{j|(j,i)\in A\}} c_{ji} - \sum_{\{j|(i,j)\in A\}} b_{ij} + s_i,$$

$$\lim_{\xi\to\infty} g_i(p_1, \ldots, p_{i-1}, \xi, p_{i+1}, \ldots, p_{|N|}) = \sum_{\{j|(j,i)\in A\}} b_{ji} - \sum_{\{j|(i,j)\in A\}} c_{ij} + s_i.$$

Furthermore $R_i(p)$ is nonempty and $g_i(p)$ is the ith partial derivative $\partial q(p)/\partial p_i$ of the concave dual function q. These facts imply the desired form of $R_i(p)$. **Q.E.D.**

We now define the (point–to–point) mappings

$$\overline{R}_i(p) = \max_{\xi\in R_i(p)} \xi, \qquad \forall\, i \text{ such that } s_i < \sum_{\{j|(i,j)\in A\}} c_{ij} - \sum_{\{j|(j,i)\in A\}} b_{ji}, \tag{6.10}$$

$$\underline{R}_i(p) = \min_{\xi \in R_i(p)} \xi, \qquad \forall\, i \text{ such that } \sum_{\{j \mid (i,j) \in A\}} b_{ij} - \sum_{\{j \mid (j,i) \in A\}} c_{ji} < s_i. \quad (6.11)$$

We call $\overline{R}_i$ the *ith maximal relaxation mapping*, and $\underline{R}_i$ the *ith minimal relaxation mapping*. They give the maximal and minimal maximizing points, respectively, of the dual cost along the ith coordinate with all other coordinates held fixed; these points exist for all i satisfying the conditions of Eqs. (6.10) and (6.11) in view of Lemma 6.1. When $\overline{R}_i$ is defined for all $i = 2, \ldots, |N|$, we denote by $\overline{R}$ the (point–to–point) mapping from $\Re^{|N|}$ to $\Re^{|N|}$ defined by

$$\overline{R}(p) = \big[c, \overline{R}_2(p), \ldots, \overline{R}_{|N|}(p)\big]. \quad (6.12)$$

Similarly, when $\underline{R}_i$ is defined for all $i = 2, \ldots, |N|$, we denote by $\underline{R}$ the (point–to–point) mapping from $\Re^{|N|}$ to $\Re^{|N|}$ defined by

$$\underline{R}(p) = \big[c, \underline{R}_2(p), \ldots, \underline{R}_{|N|}(p)\big]. \quad (6.13)$$

We also denote by $\overline{R}^k$ and $\underline{R}^k$ the composition of $\overline{R}$ and $\underline{R}$, respectively, with themselves k times.

A key fact about the mappings $\overline{R}_i$ and $\underline{R}_i$ is that they are continuous and monotone. This can be visualized as in Figs. 6.6.1 and 6.6.2, and is shown in the following proposition:

Proposition 6.1. The ith maximal and minimal relaxation mappings $\overline{R}_i$ and $\underline{R}_i$, given by Eqs. (6.10) and (6.11), are continuous on $\Re^{|N|}$. Furthermore $\overline{R}_i$ and $\underline{R}_i$ are monotone on $\Re^{|N|}$ in the sense that for all p and $p' \in \Re^{|N|}$ we have

$$\overline{R}_i(p') \leq \overline{R}_i(p) \qquad \text{if } \; p' \leq p, \quad (6.14)$$

$$\underline{R}_i(p') \leq \underline{R}_i(p) \qquad \text{if } \; p' \leq p. \quad (6.15)$$

Proof. To show continuity of $\overline{R}_i$, we argue by contradiction. Suppose there exists a sequence $\{p^k\}$ that converges to a vector p such that the corresponding sequence $\{\overline{R}_i(p^k)\}$ does not converge to $\overline{R}_i(p)$. By passing to a subsequence, if necessary, suppose that for some $\epsilon > 0$, we have

$$\overline{R}_i(p) \geq \overline{R}_i(p^k) + \epsilon, \qquad \forall\, k, \quad (6.16)$$

(the proof is similar if $\epsilon < 0$ and the inequality is reversed). By the definition of $\overline{R}_i$, we have

$$\sum_{\{j \mid (j,i) \in A\}} \nabla q_{ji}\big(p_j - \overline{R}_i(p)\big) = \sum_{\{j \mid (i,j) \in A\}} \nabla q_{ij}\big(\overline{R}_i(p) - p_j\big) + s_i, \quad (6.17)$$

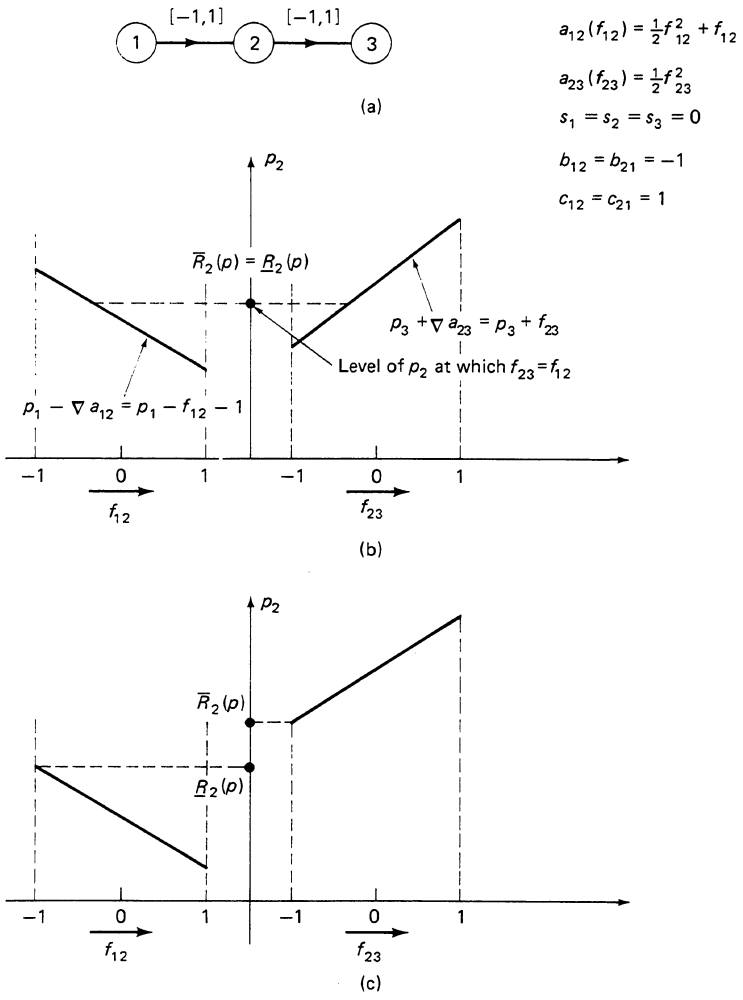


Figure 6.6.1 Illustration of mappings $\underline{R}_i$, $\overline{R}_i$, and R_i. (a) An example problem involving three nodes with zero supplies and two arcs with the arc costs and flow bounds shown. (b) Here $R_2(p)$ consists of a single element. (c) Here $\underline{R}_2(p) < \overline{R}_2(p)$ and $R_2(p)$ consists of the interval $\left[\underline{R}_2(p), \overline{R}_2(p)\right]$. In both (b) and (c), it is evident that $\overline{R}_2(p)$ and $\underline{R}_2(p)$ change continuously and monotonically with p_1 and p_3.

$$\sum_{\{j|(j,i)\in A\}} \nabla q_{ji}\left(p_j^k - \overline{R}_i(p^k)\right) = \sum_{\{j|(i,j)\in A\}} \nabla q_{ij}\left(\overline{R}_i(p^k) - p_j^k\right) + s_i, \quad \forall\, k. \tag{6.18}$$

Since $p^k \to p$, it follows using Eq. (6.16) that for sufficiently large k, we have

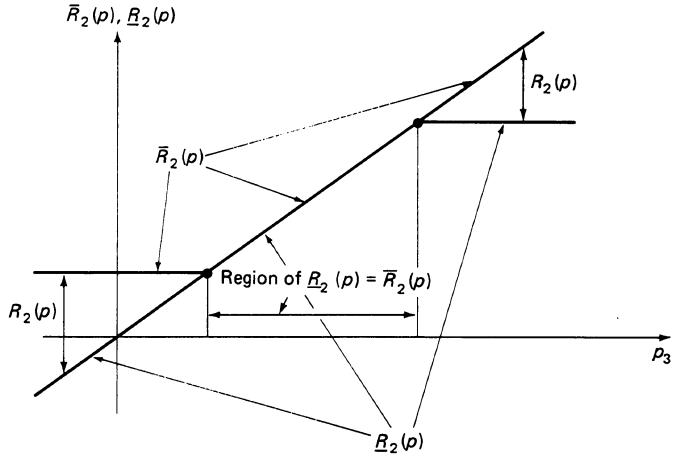


Figure 6.6.2 Illustration of continuity and monotonicity of $\underline{R}_2(p)$ and $\overline{R}_2(p)$ in the example of Fig. 6.1. The figure shows $\underline{R}_2(p)$, $\overline{R}_2(p)$, and $R_2(p)$ as p_1 is fixed and p_3 varies.

$$p_j^k - \overline{R}_i(p^k) > p_j - \overline{R}_i(p), \qquad \forall\ (j,i) \in A,$$
$$\overline{R}_i(p^k) - p_j^k < \overline{R}_i(p) - p_j, \qquad \forall\ (i,j) \in A.$$

Therefore, for sufficiently large k, we have, using the concavity of q_{ji} and q_{ij},

$$\nabla q_{ji}\big(p_j^k - \overline{R}_i(p^k)\big) \le \nabla q_{ji}\big(p_j - \overline{R}_i(p)\big), \qquad \forall\ (j,i) \in A,$$
$$\nabla q_{ij}\big(\overline{R}_i(p^k) - p_j^k\big) \ge \nabla q_{ij}\big(\overline{R}_i(p) - p_j\big), \qquad \forall\ (i,j) \in A.$$

Using these relations together with Eqs. (6.17) and (6.18), we obtain for all sufficiently large k

$$\nabla q_{ji}\big(p_j - \overline{R}_i(p)\big) = \nabla q_{ji}\big(p_j^k - \overline{R}_i(p^k)\big), \qquad \forall\ (j,i) \in A, \tag{6.19}$$
$$\nabla q_{ij}\big(\overline{R}_i(p) - p_j\big) = \nabla q_{ij}\big(\overline{R}_i(p^k) - p_j^k\big), \qquad \forall\ (i,j) \in A. \tag{6.20}$$

Consider the intervals I_{ji} and I_{ij} given by

$$I_{ji} = \Big\{t \mid \nabla q_{ji}(t) = \nabla q_{ji}\big(p_j - \overline{R}_i(p)\big)\Big\}, \qquad \forall\ (j,i) \in A, \tag{6.21}$$
$$I_{ij} = \Big\{t \mid \nabla q_{ij}(t) = \nabla q_{ij}\big(\overline{R}_i(p) - p_j\big)\Big\}, \qquad \forall\ (i,j) \in A. \tag{6.22}$$

For k sufficiently large so that Eqs. (6.19) and (6.20) hold, we have

$$\overline{R}_i(p) = \max\big\{\xi \mid \xi \in (p_j - I_{ji}),\ (j,i) \in A,\ \xi \in (I_{ij} - p_j),\ (i,j) \in A\big\},$$
$$\overline{R}_i(p^k) = \max\big\{\xi \mid \xi \in (p_j^k - I_{ji}),\ (j,i) \in A,\ \xi \in (I_{ij} - p_j^k),\ (i,j) \in A\big\}.$$

Since $p^k \to p$, it is evident from these relations that $\overline{R}_i(p^k) \to \overline{R}_i(p)$, thereby contradicting the hypothesis $\overline{R}_i(p) \geq \overline{R}_i(p^k) + \epsilon$ for all k [cf. Eq. (6.16)].

To show monotonicity of $\overline{R}_i$, let $p \in P$ and $p' \in P$ be such that $p \geq p'$. The concavity of q_{ij} and q_{ji} implies that

$$\nabla q_{ji}\big(p_j - \overline{R}_i(p')\big) \leq \nabla q_{ji}\big(p'_j - \overline{R}_i(p')\big), \qquad \forall\ (j,i) \in A, \tag{6.23}$$

$$\nabla q_{ij}\big(\overline{R}_i(p') - p_j\big) \geq \nabla q_{ij}\big(\overline{R}_i(p') - p'_j\big), \qquad \forall\ (i,j) \in A. \tag{6.24}$$

Thus,

$$\begin{aligned} 0 &= s_i + \sum_{\{j|(i,j)\in A\}} \nabla q_{ij}\big(\overline{R}_i(p') - p'_j\big) - \sum_{\{j|(j,i)\in A\}} \nabla q_{ji}\big(p'_j - \overline{R}_i(p')\big) \\ &\leq s_i + \sum_{\{j|(i,j)\in A\}} \nabla q_{ij}\big(\overline{R}_i(p') - p_j\big) - \sum_{\{j|(j,i)\in A\}} \nabla q_{ji}\big(p_j - \overline{R}_i(p')\big). \end{aligned}$$

Since $s_i + \sum_{\{j|(i,j)\in A\}} \nabla q_{ij}\big(\xi - p_j\big) - \sum_{\{j|(j,i)\in A\}} \nabla q_{ji}\big(p_j - \xi\big)$ is negative for any $\xi > \overline{R}_i(p)$, we must have $\overline{R}_i(p') \leq \overline{R}_i(p)$.

The proof of continuity and monotonicity of $\underline{R}_i$ is analogous to the one just given for $\overline{R}_i$ and is omitted. **Q.E.D.**

We now consider the *restricted dual optimal solution set* defined as

$$P^* = \big\{p^* \in P \mid q(p^*) = \max_{p\in P} q(p)\big\}.$$

In order to characterize this set, we introduce some terminology. We say that P^* is *bounded above* if there exists $\hat{p} \in P$ such that $p^* \leq \hat{p}$ for all $p^* \in P^*$, and we say that P^* is *bounded below* if there exists $\hat{p} \in P$ such that $\hat{p} \leq p^*$ for all $p^* \in P^*$. Note that if P^* is bounded above, then the set $R_i(p^*)$ is bounded above for all i and $p^* \in P^*$, so by Lemma 6.1, we have

$$s_i < \sum_{\{j|(i,j)\in A\}} c_{ij} - \sum_{\{j|(j,i)\in A\}} b_{ji}, \qquad \forall\ i = 2,\ldots,|N|, \tag{6.25}$$

and the maximal relaxation mapping $\overline{R}_i$ is defined for all $i = 2,\ldots,|N|$ [cf. Eq. (6.10)]. Similarly, if P^* is bounded below, then the minimal relaxation mapping $\underline{R}_i$ is defined for all $i = 2,\ldots,|N|$.

The following proposition shows an interesting property of P^*.

Proposition 6.2. If the restricted dual optimal solution set P^* is bounded above, then there exists a maximal element of P^*, that is, an element $\overline{p} \in P^*$ such that

$$p^* \leq \overline{p}, \qquad \forall\ p^* \in P^*.$$

Similarly, if P^* is bounded below, then there exists a minimal element of P^*, that is, an element $\underline{p} \in P^*$ such that

$$\underline{p} \leq p^*, \qquad \forall\, p^* \in P^*.$$

Proof. Assume that P^* is bounded above. Then, since P^* is also closed, it must contain elements $p^1,\ p^2, \ldots, p^{|N|}$ such that for all $p \in P^*$ and $i = 1, 2, \ldots, |N|$ we have $p_i^i \geq p_i$. We claim that the vector $\overline{p}$ given by

$$\overline{p}_i = p_i^i, \qquad \forall\, i = 1, 2, \ldots, |N|,$$

is the maximal element of P^*. By construction, $\overline{p} \geq p$ for all $p \in P^*$, so it only remains to show that $\overline{p} \in P^*$. Since $\overline{p}_i = \max\left\{p_i^1, \ldots, p_i^{|N|}\right\}$, it suffices to show that for any $p' \in P^*$ and $p'' \in P^*$, the vector $\hat{p}$ with coordinates given by

$$\hat{p}_i = \max\{p_i', p_i''\}, \qquad \forall\, i = 1, 2, \ldots, |N|, \tag{6.26}$$

belongs to P^*. Denote $I = \{i \mid p_i' > p_i''\}$ and let f^* be the unique optimal flow vector. The concavity of q_{ij} and the optimality of p' and p'' imply that

$$-f_{ji}^* = \nabla q_{ji}(p_j' - p_i') \geq \nabla q_{ji}(p_j'' - p_i') \geq \nabla q_{ji}(p_j'' - p_i'') = -f_{ji}^*,$$
$$\forall\, (j,i) \in A \text{ with } i \in I,\ j \notin I, \tag{6.27}$$

$$-f_{ij}^* = \nabla q_{ij}(p_i' - p_j') \leq \nabla q_{ij}(p_i' - p_j'') \leq \nabla q_{ij}(p_i'' - p_j'') = -f_{ij}^*,$$
$$\forall\, (i,j) \in A \text{ with } i \in I,\ j \notin I. \tag{6.28}$$

We also have

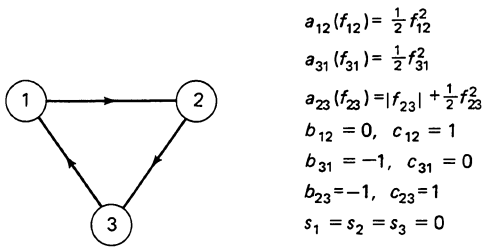
$$\nabla q_{ij}(p_i' - p_j') = -f_{ij}^*, \qquad \forall\, (i,j) \in A \text{ with } i \in I,\ j \in I, \tag{6.29}$$

$$\nabla q_{ij}(p_i'' - p_j'') = -f_{ij}^*, \qquad \forall\, (i,j) \in A \text{ with } i \notin I,\ j \notin I. \tag{6.30}$$

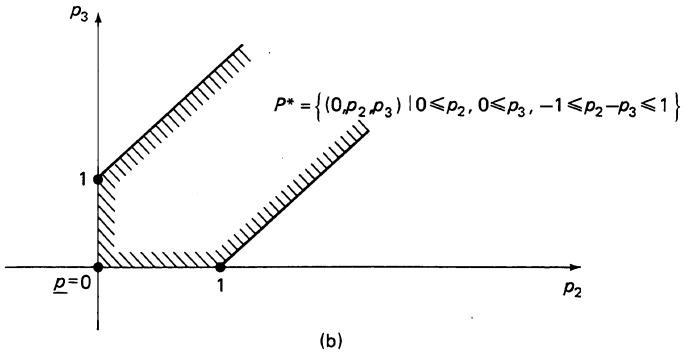
By combining Eqs. (6.26) through (6.30) we obtain $\nabla q_{ij}(\hat{p}_i - \hat{p}_j) = -f_{ij}^*$ for all $(i,j) \in A$. Therefore $\hat{p} \in P^*$.

The proof of existence of a minimal element $\underline{p}$ is similar. **Q.E.D.**

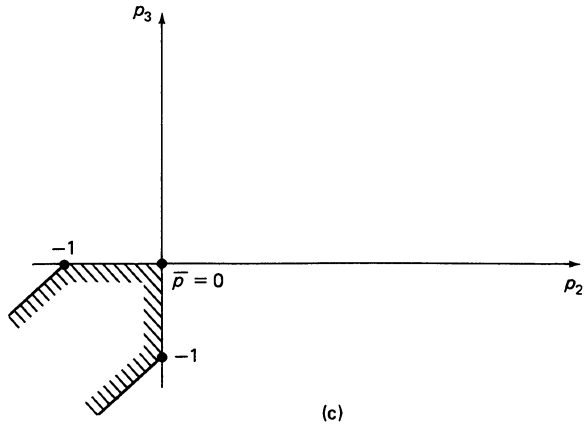
Figures 6.6.3 and 6.6.4 provide examples of problems where P^* is bounded above and/or below and illustrates the maximal and/or minimal elements of P^*. Exercise 6.1 provides conditions under which boundedness above and/or below of P^* can be verified. A typical case where P^* is unbounded above as well as below occurs when the graph contains two disconnected components; in this case, the dual function value is unchanged if a constant is added to the prices of the nodes of the first component and a different constant is added to the prices of the nodes of the second component.



(a)



(b)



(c)

Figure 6.6.3 Illustration of situations where P^* is unbounded above or below; cf. Exercise 6.1. (a) An example problem with data as shown. The optimal flow vector is $(0, 0, 0)$. (b) The form of P^* for the problem of part (a). (c) The form of P^* when the flow bounds of arcs (1,2) and (3,1) are changed to $b_{12} = -1$, $c_{12} = 0$, and $b_{31} = 0$, $c_{31} = 1$, respectively.

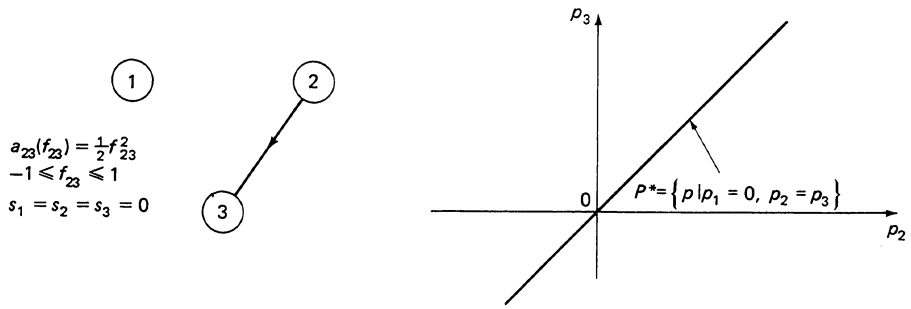


Figure 6.6.4 An example where the reduced dual optimal solution set P^* is not bounded below or above.

We now consider a distributed asynchronous algorithm based on the relaxation mapping R_i. This algorithm is cast into the algorithmic model of Section 6.1 by taking $X_1 = \{c\}$, X_i equal to the real line for $i = 2, \ldots, |N|$, $x = p$, and

$$f_1(p) = c \qquad \text{for } i = 1,$$
$$f_i(p) \in R_i(p) \qquad \text{for } i = 2, \ldots, |N|.$$

We call this algorithm the *asynchronous relaxation method* (ARM). The price vector generated by ARM at time t is denoted $p(t)$, and its ith coordinate is denoted $p_i(t)$. We have the following proposition:

Proposition 6.3. Let the initial vector $p(0)$ be arbitrary subject to $p_1(0) = c$.

(a) If the restricted dual optimal solution set P^* is bounded above, every limit point of a sequence $\{p(t)\}$ generated by the ARM belongs to the set

$$\overline{P} = \{p \in P \mid p \leq \overline{p}\}, \tag{6.31}$$

where $\overline{p}$ is the maximal element of P^* (cf. Prop. 6.2).

(b) If P^* is bounded below, every limit point of a sequence $\{p(t)\}$ generated by the ARM belongs to the set

$$\underline{P} = \{p \in P \mid \underline{p} \leq p\}, \tag{6.32}$$

where $\underline{p}$ is the minimal element of P^* (cf. Prop. 6.2).

(c) If P^* consists of a unique element p^*, we have

$$\lim_{t \to \infty} p(t) = p^*. \tag{6.33}$$

Proof.

(a) Let $\hat{p}$ be the vector defined by $\hat{p}_i = \overline{p}_i + \delta$ for all $i = 1, 2, \ldots, |N|$, where $\delta > 0$ is a positive constant chosen large enough so that $p(0) \leq \hat{p}$. Consider the mapping $\overline{R}$ of Eq. (6.12). We have using the definition (6.9) of $R_i(p)$,

$$\overline{R}_i(\hat{p}) = \overline{R}_i(\overline{p}) + \delta = \overline{p}_i + \delta, \qquad \forall\, i = 2, \ldots, |N|,$$

while the first coordinates of $\overline{R}(\hat{p})$ and $\overline{R}(\overline{p})$ are both equal to c. Therefore we have $\overline{p} \leq \overline{R}(\hat{p}) \leq \hat{p}$ and by using the monotonicity of the components of the mapping $\overline{R}$, we obtain

$$\overline{p} \leq \overline{R}^{k+1}(\hat{p}) \leq \overline{R}^k(\hat{p}), \qquad \forall\, k.$$

From this relation, we see that the sequence $\{\overline{R}^k(\hat{p})\}$ converges to some p, and by the continuity of $\overline{R}$, we must have $p = \overline{R}(p)$ as well as $p \geq \overline{p}$. Since $p = \overline{R}(p)$ implies that $p \in P^*$ and since $\overline{p}$ is the maximal element of P^*, we see that $\overline{p} = p = \lim_{k\to\infty} \overline{R}^k(\hat{p})$. We now consider the sets

$$\overline{P}(k) = \left\{p \in P \mid p \leq \overline{R}^k(\hat{p})\right\}, \qquad k = 0, 1, \ldots, \tag{6.34}$$

and note that the sequence $\left\{\overline{P}(k)\right\}$ is nested, and its intersection is the set $\overline{P} = \{p \in P \mid p \leq \overline{p}\}$ of Eq. (6.31). We apply the Asynchronous Convergence Theorem of Section 6.2 with the sets $\overline{P}(k)$ in place of $X(k)$. It is evident that these sets satisfy the Synchronous Convergence and Box Conditions of Assumption 2.1, so the result follows.

(b) Similar with the proof of part (a).

(c) Follows from parts (a) and (b), since $\overline{p}$ and $\underline{p}$ coincide with p^*. **Q.E.D.**

Proposition 6.3 shows that the ARM has satisfactory convergence properties when P^* has a unique element. One way to check this condition is to consider the optimal solution f^* of the primal problem (CNF), and the subset of arcs

$$\tilde{A} = \left\{(i,j) \in A \mid b_{ij} < f^*_{ij} < c_{ij} \text{ and } a_{ij} \text{ is differentiable at } f^*_{ij}\right\}.$$

For every optimal price vector p^*, we must have $p^*_i - p^*_j = \nabla a_{ij}(f^*_{ij})$ for all $(i,j) \in \tilde{A}$, so if the graph $(N, \tilde{A})$ is connected, it is seen that P^* must consist of a unique element. In order to improve the convergence properties when P^* has more then one element, it is necessary to modify the ARM so that a price relaxation at node i replaces p_i with $\overline{R}_i(p)$ [not just any element of $R_i(p)$]. We call this method the *maximal ARM*. If in place of $\overline{R}_i(p)$ we use $\underline{R}_i(p)$ the resulting method is called the *minimal ARM*.

Proposition 6.4.

(a) Assume that the restricted dual optimal solution set P^* is bounded above and $\overline{p}$ is its maximal element. If the initial vector $p(0)$ satisfies $p_1(0) = c$ and $\overline{p} \leq p(0)$, then a sequence generated by the maximal ARM converges to $\overline{p}$.

(b) Assume that P^* is bounded below and $\underline{p}$ is its minimal element. If the initial vector $p(0)$ satisfies $p_1(0) = c$ and $p(0) \leq \underline{p}$, then a sequence generated by the minimal ARM converges to $\underline{p}$.

Proof. The proof of part (a) is identical to the one of Prop. 6.3(a) except that the sets $P(k)$ of Eq. (6.34) are replaced by $\{p \in P \mid \overline{p} \leq p \leq \overline{R}^k(\hat{p})\}$. The proof of part (b) is similar. **Q.E.D.**

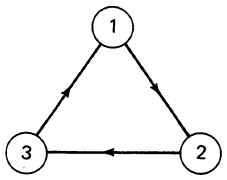


Figure 6.6.5 A three node network example. The arc cost functions and the lower and upper arc flow bounds are

$$a_{12}(f_{12}) = |f_{12}| + (f_{12})^2, \quad -1 \leq f_{12} \leq 1,$$

$$a_{23}(f_{23}) = (f_{23})^2, \quad -1 \leq f_{23} \leq 1,$$

$$a_{31}(f_{31}) = |f_{23}| + (f_{31})^2, \quad -1 \leq f_{31} \leq 1.$$

The node supplies are zero. The optimal primal solution is $f^*_{12} = f^*_{23} = f^*_{31} = 0$. The reduced dual optimal solution set for $c = 0$ is

$$P^* = \{p \mid p_1 = 0,\ p_2 = p_3,$$

$$-1 \leq p_2 \leq 1,\ -1 \leq p_3 \leq 1\}.$$

To illustrate the results of Props. 6.3 and 6.4, consider the example of Fig. 6.6.5. The regions of initial price vectors for which the ARM, the maximal ARM, and the minimal ARM converge to the optimal solutions are shown in Fig. 6.6.6. This example shows that the results of Props. 6.3 and 6.4 cannot be improved even for the special case of the synchronous Jacobi relaxation version. A method to overcome the difficulty in this example is discussed in Subsection 7.2.3.

EXERCISES

6.1. Show that the restricted dual optimal solution set P^* is unbounded above if and only if there exists a nonempty subset $S \subset N$ such that $1 \notin S$ and for all $(i, j) \in A$ with $i \in S$ and $j \notin S$, we have $f^*_{ij} = c_{ij}$, and for all $(i, j) \in A$ with $i \in S$ and $j \notin S$, we have $f^*_{ij} = b_{ij}$. Formulate a similar condition for P^* to be unbounded below. *Hint:* If P^* is unbounded

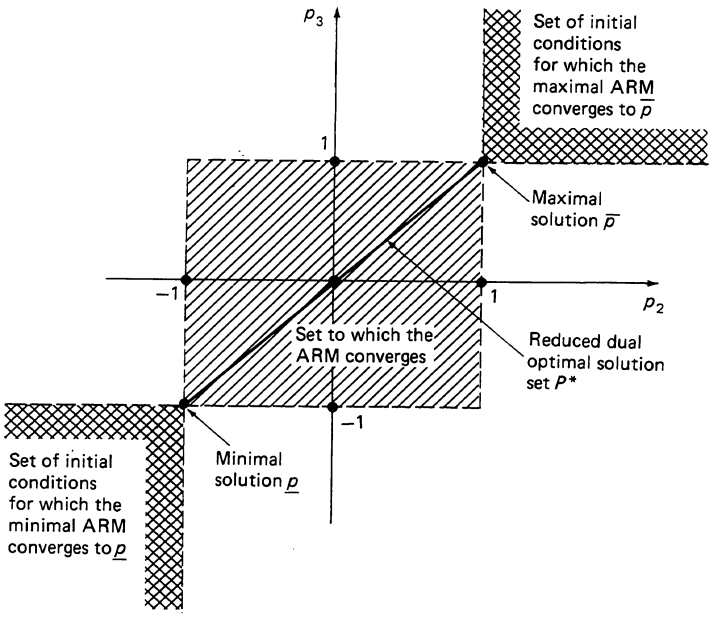


Figure 6.6.6 The structure of the reduced dual optimal solution set, the convergence regions of the ARM, the maximal ARM, and the minimal ARM for the example of Fig. 6.6.5.

above, there must exist vectors p^* and u in $\Re^{|N|}$ such that $p^* + \alpha u \in P^*$ for all $\alpha \geq 0$, and, furthermore, the set $S = \{i \in N \mid u_i > 0\}$ is nonempty. Use the fact that the unique optimal flow vector f^* must satisfy complementary slackness with $p^* + \alpha u$ for all $\alpha \geq 0$ in order to show that the set S has the required property. Conversely, if S has the stated property, and $p^* \in P^*$, then $p^* + \alpha u \in P^*$ for all $\alpha \geq 0$, where the ith coordinate of u is 1 if $i \in S$ and is 0 otherwise.

6.7 ASYNCHRONOUS RELAXATION FOR ORDINARY DIFFERENTIAL EQUATIONS AND TWO-POINT BOUNDARY VALUE PROBLEMS

Consider a dynamical system consisting of m interconnected subsystems. Let $x_i(t) \in R^{n_i}$ be the state vector of the ith subsystem at time t. Suppose that the initial state $x_i(0)$ of each subsystem has been fixed, and suppose that the dynamics of the system are described by the following set of linear differential equations:

$$\frac{dx_i}{dt}(t) + D_i(t)x_i(t) = \sum_{j=1}^{m} B_{ij}(t)x_j(t) + u_i(t), \qquad i = 1, \ldots, m. \tag{7.1}$$

Here $D_i(t)$ and $B_{ij}(t)$ are matrices of dimensions $n_i \times n_i$ and $n_i \times n_j$, respectively, and $u_i(t)$ is a known vector of dimension n_i. The matrix $D_i(t)$ is supposed to describe the internal dynamics of the ith subsystem, and the matrices $B_{ij}(t)$ are meant to describe the interaction of the ith subsystem with other subsystems. Let $n = \sum_{i=1}^{m} n_i$ be the dimension of the overall system. Let $D(t)$ be the block–diagonal matrix of size $n \times n$, which has $D_i(t)$ as its ith block; let $B(t)$ be the $n \times n$ matrix consisting of the blocks $B_{ij}(t)$. With this notation, the system (7.1) can be written compactly as

$$\frac{dx}{dt}(t) + D(t)x(t) = B(t)x(t) + u(t). \tag{7.2}$$

We will assume throughout this section that each element of $D_i(\cdot)$, $B_{ij}(\cdot)$ and $u_i(\cdot)$ is a continuous and bounded function of time, over the time interval $[0, \infty)$. This guarantees that the system (7.1) has a unique solution over $[0, \infty)$ for any initial conditions, as discussed in Appendix A (Prop. A.44).

In this section, we describe an asynchronous relaxation algorithm for solving numerically the system (7.1) over a time interval of the form $[0, T]$, where T is a positive real number, or over the time interval $[0, \infty)$. Relaxation algorithms for solving ordinary differential equations are based on the following idea. Suppose that we have available some approximation of the trajectories of the different subsystems. We use these approximate trajectories in the right–hand side of the differential equation (7.1), and then solve this equation for the trajectory of the ith subsystem. (This step is called *relaxing* the ith equation.) This is done for each subsystem, and then the procedure is repeated until eventually convergence to a set of sufficiently good approximate solutions is obtained. The situation is the same as when solving systems of equations. The only difference is that the objects that we are solving for are not vectors in a finite dimensional vector space but belong to an infinite dimensional space of time functions. Similarly, as with systems of equations with a finite number of unknowns, there are several variations of the above described algorithm. The equations (7.1) can be relaxed one after the other, which corresponds to a Gauss–Seidel algorithm, or they can be relaxed simultaneously, which corresponds to a Jacobi algorithm. More generally, these equations can be relaxed in an arbitrary order by a set of parallel processors operating asynchronously, according to the model introduced in Section 6.1.

6.7.1 The Asynchronous Relaxation Algorithm

Let X_i be the set of functions $x_i(\cdot)$ that are continuous over the time interval of interest $\big([0, T]$ or $[0, \infty)\big)$, and whose initial value $x_i(0)$ agrees with the given initial conditions. Let $X = X_1 \times \cdots \times X_m$. We define a mapping $f_i : X \mapsto X_i$ as follows. Given functions $x_j(\cdot) \in X_j$, $j = 1, ..., m$, let

$$y_i(\cdot) = f_i\big(x_1(\cdot), \ldots, x_m(\cdot)\big)$$

be the solution of the differential equation

$$\frac{dy_i}{dt}(t) + D_i(t)y_i(t) = \sum_{j=1}^{m} B_{ij}(t)x_j(t) + u_i(t), \tag{7.3}$$

subject to the initial condition $y_i(0) = x_i(0)$. In view of the continuity assumption on D_i, B_{ij}, and u_i, the solution $y_i(\cdot)$ of this differential equation exists, is unique, and belongs to X_i. Furthermore, $y(\cdot)$ has the following explicit representation (see Prop. A.44 in Appendix A)

$$y_i(t) = \int_0^t \Phi_i(t,\tau)\left[\sum_{j=1}^{m} B_{ij}(\tau)x_j(\tau) + u_i(\tau)\right] d\tau + \Phi_i(t,0)x_i(0), \tag{7.4}$$

where $\Phi_i(t,\tau)$ is the matrix–valued function that satisfies the differential equation

$$\frac{\partial \Phi_i(t,\tau)}{\partial t} = -D_i(t)\Phi_i(t,\tau), \tag{7.5}$$

and the initial condition $\Phi_i(\tau,\tau) = I$, where I is the $n_i \times n_i$ identity matrix. Let $f : X \mapsto X$ be the mapping whose ith component is the mapping f_i, and let $x^*(\cdot)$ be the solution of the system (7.1) subject to the given initial conditions. Then $x^*(\cdot)$ satisfies the integral equation

$$x_i^*(t) = \int_0^t \Phi_i(t,\tau)\left[\sum_{j=1}^{m} B_{ij}(\tau)x_j^*(\tau) + u_i(\tau)\right] d\tau + \Phi_i(t,0)x_i(0), \tag{7.6}$$

and from Eq. (7.4) it is seen that $x^*(\cdot)$ is a fixed point of f.

Having defined the mapping f, we consider the corresponding asynchronous algorithm in accordance with the model of Section 6.1. We assume that the algorithm is initialized with functions $x_i(\cdot)$ belonging to X_i for each i. Application of the mapping f_i corresponds to updating the ith component x_i by a processor. In the present context, each component x_i is a vector time function, an infinite dimensional object. This does not pose any mathematical difficulty because the Asynchronous Convergence Theorem of Section 6.2 has been proved without assuming that the underlying space X is finite dimensional.

In practice, one should have available a numerical method for solving the differential equation (7.3). This issue is not directly related to the issues of parallelism and asynchronism and has to be dealt with even in a uniprocessor environment. We do not discuss this issue further and refer the reader to textbooks on numerical analysis [CoL55], [Hen64], and [Hil74].

Pointwise Convergence Over the Interval $[0, \infty)$

We now prove that the sequence of trajectories produced by the asynchronous relaxation algorithm converges pointwise to $x^*(\cdot)$. We will need the following lemma, which is proved in Prop. A.44 of Appendix A.

Lemma 7.1. For every i, there exist constants $C_i \geq 0$ and $c_i \geq 0$ such that

$$||\Phi_i(t,\tau)||_\infty \leq C_i e^{c_i(t-\tau)}, \qquad \forall\, t \geq \tau, \tag{7.7}$$

where $||\cdot||_\infty$ is the matrix norm induced by the maximum norm.

Let A be an upper bound on $\|B_{ij}(t)\|_\infty$, let $K > 0$ be an arbitrary constant, and let g and G be constants larger than all the constants c_i and C_i of Lemma 7.1, respectively. For any nonnegative integer k, let

$$X(k) = \left\{ x(\cdot) \in X \,\middle|\, ||x(t) - x^*(t)||_\infty \leq K \frac{(mGAt)^k}{k!} e^{gt},\ \forall\, t \geq 0 \right\}.$$

The sets $X(k)$ satisfy the Box Condition of Section 6.2 and will be used in a subsequent application of the Asynchronous Convergence Theorem. The following lemma shows that the Synchronous Convergence Condition of Section 6.2 is also satisfied.

Lemma 7.2. The mapping f maps $X(k)$ into $X(k+1)$ for every $k \geq 0$.

Proof. Let us assume that $x(\cdot) \in X(k)$ and let $y(\cdot) = f\big(x(\cdot)\big)$. Using the representations (7.4) and (7.6), Lemma 7.1, and the definition of A, we have

$$\begin{aligned}
||y_i(t) - x_i^*(t)||_\infty &= \left\| \int_0^t \Phi_i(t,\tau) \left[\sum_{j=1}^m B_{ij}(\tau)\big(x_j(\tau) - x_j^*(\tau)\big) \right] d\tau \right\|_\infty \\
&\leq \int_0^t G e^{g(t-\tau)} mAK \frac{(mGA\tau)^k}{k!} e^{g\tau}\, d\tau \\
&= K(mGA)^{k+1} e^{gt} \int_0^t \frac{\tau^k}{k!}\, d\tau \\
&= K(mGA)^{k+1} e^{gt} \frac{t^{k+1}}{(k+1)!}.
\end{aligned}$$

Q.E.D.

It can be shown by using an argument similar to the one of Prop. A.44 in Appendix A, that there exist positive constants f and F such that $||x^*(t)||_\infty \leq F e^{ft}$. Suppose that the algorithm is initialized with a time function $z(\cdot) \in X$ that is bounded by some

constant H. It follows that $||z(t) - x^*(t)||_\infty \leq Ke^{ft}$, where $K = H + F$. Therefore, if the algorithm is initialized with a bounded and continuous time function satisfying the given initial conditions, this time function belongs to the set $X(0)$ for a suitable choice of the constants K, g, G. We can then apply the Asynchronous Convergence Theorem, showing that the algorithm will eventually enter and stay in the set $X(k)$ for any k. In particular, for any fixed t, the maximum norm of $\big(x(t) - x^*(t)\big)$ eventually becomes smaller than $K(mGAt)^k e^{gt}/k!$, which converges to zero as $k \to \infty$. This proves the following:

Proposition 7.1. The sequence of time functions produced by the asynchronous algorithm based on the mapping f, initialized with a bounded and continuous function satisfying the given initial conditions, converges pointwise to the solution of the system (7.1).

Uniform Convergence on $[0, T]$

If we are interested only in a finite interval $[0, T]$, with $T < \infty$, we can define the sets $X(k)$ by

$$X(k) = \left\{ x(\cdot) \in X \,\middle|\, ||x(t) - x^*(t)||_\infty \leq K\frac{(mGAt)^k}{k!}e^{gt},\ \forall\, t \in [0, T] \right\}.$$

We then repeat the argument in Lemma 7.2 and conclude again that f maps $X(k)$ into $X(k+1)$, and, therefore, the time functions produced by the algorithm eventually enter every set $X(k)$. This implies that, for every k, the quantity $\max_{t\in[0,T]} ||x(t) - x^*(t)||_\infty$ eventually becomes smaller than $K(mGAT)^k e^{gT}/k!$, which converges to zero as $k \to \infty$. We have thus proved the following:

Proposition 7.2. If $T < \infty$, then the sequence of time functions generated by the asynchronous algorithm, initialized by an arbitrary continuous function satisfying the given initial conditions, converges uniformly to the solution of the system (7.1).

Uniform Convergence on $[0, \infty)$

The uniform convergence result of Prop. 7.2 seems encouraging because in practice we would only want to solve the system (7.1) over a finite time interval. However, the speed of convergence implicit in this result is unsatisfactory. Suppose that we are interested in obtaining a true solution within some $\epsilon > 0$. The argument preceding Prop. 7.2 suggests that we should wait until $x(\cdot)$ enters the set $X(k)$, where k is large enough so that $K(mGAT)^k e^{gT}/k! < \epsilon$. If we choose k according to this requirement, it is clear that k should be chosen larger when T is larger. This suggests that a larger number of iterations is needed in order to solve the system (7.1) over a larger time interval. Suppose now that the asynchronous algorithm were to converge uniformly over the interval $[0, \infty)$. Then, convergence over a smaller time interval could only be faster, implying that for any finite time interval $[0, T]$, the algorithm would converge uniformly at least as fast as for the interval $[0, \infty)$. In other words, the number of iterations required to attain a

certain accuracy would have a bound independent of T. This is the main reason for our interest in uniform convergence over the interval $[0, \infty)$.

To be able to show uniform convergence over $[0, \infty)$, it is essential to impose several additional assumptions on the system (7.1). There is a need for some type of stability assumption to guarantee that the solution $x^*(t)$ stays bounded as $t \to \infty$; otherwise, uniform convergence may not occur. There is also a need for further assumptions of the type needed for relaxation methods for (algebraic) systems of linear equations, such as diagonal dominance conditions (cf. Sections 2.6 and 6.3). The reason for this will become apparent shortly, in the proof of Prop. 7.4, where we will show an intimate connection between the limit (as $t \to \infty$) of the solution $x^*(t)$ of the differential system (7.1), and the solution of an associated linear (algebraic) system of equations. Diagonal dominance conditions needed for asynchronous convergence of relaxation methods for the linear algebraic system will be translated into corresponding conditions for convergence of relaxation methods for the differential system (7.1). The reader may find it easier to understand why stability and diagonal dominance conditions are not needed for pointwise convergence by considering the corresponding case of a system of difference equations discussed at the end of the present section.

For the remainder of this subsection we restrict the set of functions X_i to include only bounded functions; that is, for $i = 1, \ldots, m$, X_i is the set of all bounded and continuous functions $x_i(\cdot)$ satisfying the given initial conditions. We introduce some convenient norms. Let $w \in R^n$ be a vector of the form $w = (w_1, \ldots, w_m)$, where $w_i \in R^{n_i}$ are some positive vectors. For any $x_i \in R^{n_i}$, let $||x_i||_\infty^{w_i}$ be its weighted maximum norm, as defined in Appendix A. We define a norm $|| \cdot ||_i$ on X_i by letting, for any $x_i(\cdot) \in X_i$,

$$||x_i(\cdot)||_i = \sup_{t \in [0,\infty)} ||x_i(t)||_\infty^{w_i}. \tag{7.8}$$

We finally define a norm $|| \cdot ||$ on X by letting, for any $x(\cdot) \in X$,

$$||x(\cdot)|| = \max_i ||x_i(\cdot)||_i.$$

We are interested in conditions on $D_i(\cdot)$ and $B_{ij}(\cdot)$ for which the iteration mapping f is a contraction with respect to the $|| \cdot ||$ norm; in that case, convergence will follow from the theory of Sections 6.2 and 6.3. We first assume that the functions $D_i(\cdot)$, $B_{ij}(\cdot)$ are constant (independent of time), and $n_i = 1$ for all i, that is, each subsystem is one–dimensional. In addition, we assume that $D_i > 0$ for all i. † The solution of Eq. (7.5) is then $\Phi_i(t, \tau) = e^{-(t-\tau)D_i}$, and by using Eqs. (7.4) and (7.6), we obtain

$$y_i(t) - x_i^*(t) = \int_0^t e^{-(t-\tau)D_i} \sum_{j=1}^m B_{ij}\big(x_j(\tau) - x_j^*(\tau)\big) d\tau. \tag{7.9}$$

† This asssumption is motivated by the following consideration. If $D_i \leq 0$, then even if $x(\cdot)$ and $u(\cdot)$ are bounded functions, $f_i\big(x(\cdot)\big)$ will be in general unbounded and uniform convergence becomes impossible, except for a few trivial cases.

We divide both sides of this equation by w_i, and then take absolute values to obtain

$$\begin{aligned}\left|\frac{y_i(t)-x_i^*(t)}{w_i}\right| &\le \int_0^t e^{-(t-\tau)D_i}\sum_{j=1}^m \left|B_{ij}\frac{w_j}{w_i}\right|\cdot\left|\frac{x_j(\tau)-x_j^*(\tau)}{w_j}\right| d\tau \\ &\le \int_0^t e^{-(t-\tau)D_i}\sum_{j=1}^m \left|B_{ij}\frac{w_j}{w_i}\right|\cdot ||x(\cdot)-x^*(\cdot)||\, d\tau \\ &= \frac{1}{D_i}\left(1-e^{-tD_i}\right)\sum_{j=1}^m \left|B_{ij}\frac{w_j}{w_i}\right|\cdot ||x(\cdot)-x^*(\cdot)|| \\ &\le \frac{1}{D_i}\sum_{j=1}^m \left|B_{ij}\frac{w_j}{w_i}\right|\cdot ||x(\cdot)-x^*(\cdot)||.\end{aligned} \tag{7.10}$$

Taking the maximum over all i and t, it follows that the function f that maps $x(\cdot)$ to $y(\cdot)$ is a pseudocontraction (with respect to the norm $||\cdot||$) if

$$\frac{1}{D_i}\sum_{j=1}^m |B_{ij}|\frac{w_j}{w_i} < 1, \qquad \forall\, i. \tag{7.11}$$

[In fact f is seen to be a contraction due to its linearity properties; cf. Eq. (7.4).] In accordance with the notation introduced in the beginning of this section, let D be a diagonal matrix, with D_i being the ith diagonal element, and let $|B|$ be a matrix whose ijth element is equal to $|B_{ij}|$. The inequality (7.11) is equivalent to

$$D^{-1}|B|w < w. \tag{7.12}$$

Asynchronous convergence is obtained if the above condition holds for *some* choice of $w > 0$. We recall Corollary 6.1 of Section 2.6 which states that this condition holds for some choice of $w > 0$ if and only if $\rho\big(D^{-1}|B|\big) < 1$. We have, therefore, proved the following:

Proposition 7.3. Assume that $n_i = 1$ for all i, that the functions $D_i(\cdot)$ and $B_{ij}(t)$ do not depend on time, that $D_i > 0$ for all i, and that $\rho\big(D^{-1}|B|\big) < 1$. Then the sequence of time functions generated by the asynchronous algorithm, initialized with any bounded and continuous function satisfying the given initial conditions, converges uniformly over the interval $[0, \infty)$ to the solution of the system (7.1).

Interestingly enough, the above proposition has a converse that provides necessary conditions for uniform convergence on $[0, \infty)$.

Proposition 7.4. Suppose that $n_i = 1$ for all i, that the functions $D(\cdot)$ and $B_{ij}(\cdot)$ do not depend on time, and that $D_i > 0$ for all i. If the sequence of time functions

generated by the asynchronous algorithm converges uniformly over the time interval $[0, \infty)$ to the solution of the system (7.1) for every choice of bounded and continuous functions $U_i(\cdot)$ and every initial choice of functions $x_i(\cdot) \in X_i$, then $\rho\big(D^{-1}|B|\big) < 1$.

Proof. By assumption, the algorithm converges in the special case where $u_i(t) = 0$ for all i and t. We take this to be the case. Let Y_i be the set of time functions $x_i(\cdot) \in X_i$ for which the limit of $x_i(t)$, as $t \to \infty$, exists. Let $Y = Y_1 \times \cdots \times Y_m$. For any $x(\cdot) \in Y$, we denote by $x(\infty)$ the value of this limit. It is an easy consequence of our assumption $D_i > 0$ that the mapping f maps Y into itself. [This can be proved by using Eq. (7.4).] In particular,

$$\big(f(x)\big)(\infty) = D^{-1}Bx(\infty).$$

Thus, considering only the limiting values of the time functions generated in the course of the asynchronous algorithm, we see that they are identical to those generated by an asynchronous execution of the finite–dimensional iteration

$$x(\infty) := D^{-1}Bx(\infty). \tag{7.13}$$

Since we have assumed that the algorithm converges for every initial choice of time functions [and therefore, for every initial choice of $x(\infty)$], the asynchronous iteration (7.13) also converges. This is an iteration for the solution of a system of linear equations of the type considered in Section 6.3. Proposition 3.1 of that section is, therefore, applicable and shows that $\rho\big(D^{-1}|B|\big) < 1$. **Q.E.D.**

We will now generalize Prop. 7.3 to the multidimensional and time–varying case. We shall need some more elaborate notation: we use $x_{i,p}$ to denote the pth component of any vector $x_i \in R^{n_i}$. Accordingly, let $D_{i,pq}(t)$, $B_{ij,pq}(t)$ be the pqth entry of $D_i(t)$ and $B_{ij}(t)$, respectively.

Proposition 7.5. Assume that there exists a positive vector $w \in R^n$ and some $\epsilon > 0$ such that

$$D_{i,pp}(t)(1-\epsilon)w_{i,p} > \sum_{\{q \mid q \neq p\}} |D_{i,pq}(t)|w_{i,q} + \sum_{j=1}^{m}\sum_{q=1}^{n_j} |B_{ij,pq}(t)|w_{j,q}, \quad \forall\, i, j, p, t. \tag{7.14}$$

Then the mapping f is a contraction with respect to the norm $||\cdot||$, and uniform convergence on $[0, \infty)$ of the asynchronous algorithm follows.

Proof. In order to keep the proof simple, we assume that the input $u(t)$ and the initial conditions $x(0)$ are all zero. [It then follows that $x^*(t) = 0$ for all t.] The proof for the general case is identical provided that $x(t)$ and $y(t)$ are replaced throughout by $x(t) - x^*(t)$ and $y(t) - x^*(t)$, respectively. Let us assume that $||x(\cdot)|| \leq 1$ and let $y(\cdot) = f\big(x(\cdot)\big)$. Suppose that at some time t we have $||y(t)|| \leq 1$. Let us consider

the ith subsystem and suppose that some component $y_{i,p}(t)$ of $y(t)$ satisfies $y_{i,p}(t) \in \big[(1-\epsilon)w_{i,p}, w_{i,p}\big]$. We then claim that $(dy_{i,p}/dt)(t) < 0$. Indeed,

$$\frac{dy_{i,p}}{dt}(t) = -D_{i,pp}(t)y_{i,p}(t) - \sum_{\{q|q\neq p\}} D_{i,pq}(t)y_{i,q}(t) + \sum_{j=1}^{m}\sum_{q=1}^{n_j} B_{ij,pq}(t)x_{j,q}(t)$$

$$\leq -D_{i,pp}(t)(1-\epsilon)w_{i,p} + \sum_{\{q|q\neq p\}} |D_{i,pq}(t)|w_{i,q} + \sum_{j=1}^{m}\sum_{q=1}^{n_j} |B_{ij,pq}(t)|w_{j,q} < 0,$$

because of assumption (7.14). A similar argument shows that if $||y(t)|| \leq 1$ and $y_{i,p}(t) \in \big[-w_{i,p}, -(1-\epsilon)w_{i,p}\big]$, then $(dy_{i,p}/dt)(t) > 0$.

At the time origin, we have $||y(0)|| = 0$, since $y(\cdot)$ is initialized at zero and the above proved inequalities on the derivative of $y(\cdot)$ suggest that $||y(t)||$ can never exceed $(1-\epsilon)$. We prove this formally. Suppose that this statement is false. Then there exists some $\delta > 0$, some indices i and p and some time t such that $|y_{i,p}(t)| \geq (1-\epsilon+\delta)w_{i,p}$. Let t_2 be the first time at which this event occurs and let i and p be the corresponding indices. Let t_1 be the last time t before t_2 at which $|y_{i,p}(t)| = (1-\epsilon)w_{i,p}$. During the time interval $[t_1, t_2]$, we have $||y(t)|| \leq 1$ and $|y_{i,p}(t)| \in \big[(1-\epsilon)w_{i,p}, w_{i,p}\big]$. Hence, by our previous observations, we have $d|y_{i,p}(s)|/dt < 0$, for all $s \in [t_1, t_2]$. Thus, $|y_{i,p}(t_2)| \leq (1-\epsilon)w_{i,p}$, which is a contradiction and proves the desired result.

We conclude that $||y(t)||$ never exceeds $(1-\epsilon)$. Thus, $\|f(x(\cdot))\| \leq (1-\epsilon)\|x(\cdot)\|$ for every $x \in X$ satisfying $\|x\| \leq 1$ and, since f is linear, it follows that f is a contraction with respect to the norm $\|\cdot\|$. **Q.E.D.**

6.7.2 Two–Point Boundary Value Problems

Two–point boundary value problems are similar to the problem of solving the system of differential equations (7.1) over a finite time interval $[0, T]$, except that we are not given initial conditions for each subsystem. Rather, we are given initial conditions for some of the subsystems and final conditions for the remaining ones. Let I be the set of all i for which an initial condition for $x_i(0)$ is given and let J be the set of all i for which a final condition $x_i(T)$ is given.

A relaxation algorithm is also applicable in this case. The equation for $x_i(\cdot)$, $i \in I$, is integrated forward, while the equation for $x_j(\cdot)$, $j \in J$, is integrated backward in time, starting with the final condition at time T. Unlike the case of initial value problems treated earlier, pointwise convergence becomes a nontrivial issue; not even the synchronous (Jacobi–like) relaxation algorithm is guaranteed to converge. Convergence at a rate independent of T is obtained in the following proposition, under diagonal dominance conditions similar to those assumed in Prop. 7.5.

Proposition 7.6. Assume that the diagonal entries of D are positive for those subsystems for which initial conditions are prescribed and negative for those subsystems

for which final conditions are prescribed. Assume that there exists a vector $w > 0$ and some $\epsilon > 0$ such that

$$|D_{i,pp}(t)|(1-\epsilon)w_{i,p} > \sum_{\{q|q\neq p\}} |D_{i,pq}(t)|w_{i,q} + \sum_{j=1}^{m}\sum_{q=1}^{n_j} |B_{ij,pq}(t)|w_{j,q}, \quad \forall\, i,j,p,t. \tag{7.15}$$

Then the two–point boundary value problem has a unique solution, and the sequence of time functions generated by the asynchronous algorithm converges uniformly to it.

Proof. We argue that the mapping f is a contraction with respect to the norm $||\cdot||$. This will show that the equation $f\big(x^*(\cdot)\big) = x^*(\cdot)$ has a unique solution, and will also imply asynchronous convergence based on the theory of Sections 6.2 and 6.3.

We assume that $||x(t) - x^*(t)|| \leq 1$ for all t, and we want to prove that $||y_i(t) - x_i^*(t)||_i \leq (1-\epsilon)$ for all t, where y_i is the time function obtained by solving the equation for the ith subsystem. If we have initial conditions for the ith subsystem, then the argument is identical to the argument in the proof of Prop. 7.5. If, instead, final conditions are given, then the same argument is again applicable, except that time is reversed. **Q.E.D.**

6.7.3 The Discrete Time Case

All of the preceding continuous time results have discrete time counterparts in which the differential equation (7.1) is replaced by the difference equation

$$x_i(t+1) + D_i(t)x_i(t) = \sum_{j=1}^{m} B_{ij}(t)x_j(t) + u_i(t), \tag{7.16}$$

together with initial conditions prescribing $x(0)$. We collect below the relevant results, and we leave the proof as an exercise because it is a simple repetition of the proof of the continuous time results. Let us simply say that the pointwise convergence result is somewhat trivial. The reason is that once every equation is relaxed, then $x(1)$ assumes the correct value and never changes thereafter. Proceeding by induction, for any t, the value of $x(t)$ generated by the algorithm will in finite time become exactly equal to the corresponding value of the solution of the difference equation (7.16). In general, however, it will take at least t iterations for $x(t)$ to be obtained. As in the case of continuous time systems, we are interested in uniform convergence over $[0,\infty)$, because in that case, the value of $x(t)$ is obtained within a desired accuracy after a number of iterations that does not depend on t.

Proposition 7.7. Consider the asynchronous relaxation algorithm for solving the system of difference equations (7.16).

(a) For any t, the value of $x(t)$ generated by the asynchronous algorithm becomes eventually equal to the corresponding value for the true solution.

(b) Assume that there exists a vector $w > 0$ and a scalar $\epsilon > 0$ such that $\big(|D(t)| + |B(t)|\big)w \leq (1-\epsilon)w$ for all t. Then the sequence of time functions generated by the asynchronous relaxation algorithm, initialized with an arbitrary bounded function, converges uniformly to the solution of Eq. (7.16).

(c) Assume that each subsystem is time–invariant and one–dimensional, that is, $n_i = 1$, and that $|D_i| < 1$ for each i. Then the condition in part (b) is equivalent to the condition $\rho\big((I-|D|)^{-1}|B|\big) < 1$. Furthermore, the condition $\rho\big(|(I+D)^{-1}B|\big) < 1$ is necessary for uniform convergence on the interval $[0, \infty)$. *Note:* When the entries of D are all negative (which occurs when the difference equation arises from a simple discretization of a differential equation) the preceding necessary condition and the preceding sufficient condition coincide.

NOTES AND SOURCES

6.1 Totally asynchronous algorithmic models were introduced in [ChM69] in the context of iterative solution of linear systems of equations. Sometimes these models are also referred to as *chaotic relaxation models*. They have been subsequently studied by several authors [Bau78], [Ber82d], [Ber83], [Boj84b], [Don71], [MiS85], [Mie75], [Mit87], [RCM75], [Rob76], [Tsi87], [UrD86], [UrD88a], and [UrD88b].

6.2 The Asynchronous Convergence Theorem is due to [Ber83]. Necessary conditions for asynchronous convergence are discussed in [Tsi87].

6.3 Totally asynchronous relaxation was studied in [Rob76] and [Bau78] in connection with nonlinear problems involving maximum norm contractions. A study of some time–varying nonlinear asynchronous iterations that asymptotically approximate a maximum norm contraction is given in [LiB87].

6.3.1 The observation that the invariant distribution of a Markov chain can be found by a totally asynchronous algorithm after fixing the value of a single coordinate of the distribution vector is new. Earlier work [LuM86], which is discussed in Section 7.3, gave a partially asynchronous algorithm in which all elements of the distribution vector are updated. In practice it may be better to perform a few iterations on all coordinates before fixing the value of a single coordinate. Proposition 3.1 is due to [ChM69]; we give a somewhat different proof.

6.3.3 The material in this subsection is new. For related results see [Spi84].

6.3.5 The rate of convergence material in this subsection is new.

6.4 Totally asynchronous relaxation methods involving monotone mappings such as those arising in Dynamic Programming and the shortest path problem were studied in [Ber82d]. The asynchronous Bellman–Ford algorithm has been used in the context

of the original routing algorithm in the ARPANET (1969) and subsequently in several other data communication networks (see [BeG87], [McW77], and [MRR80] for details). An analysis of some related asynchronous shortest path routing algorithms is given in [Taj77] for the case where all arcs have unit length.

6.5 The material in this section is due to [Ber86a] and [BeE87a].

6.6 The material in this section is due to [BeE87b].

6.7 Relaxation methods for differential equations are used extensively in electric circuit simulation problems (see [NeS83] and [LRS82]). Parallel synchronous methods for solving ordinary differential equations are discussed in [Gea86] and [Gea87].

6.7.1 Asynchronous relaxation methods for differential equations were proposed in [Mit87], where Props. 7.3 and 7.5 and their generalizations to nonlinear equations were shown. The pointwise convergence result of Prop. 7.1, and the necessary condition of Prop. 7.4 are new.

6.7.2 The results in this subsection are new. Asynchronous algorithms for two–point boundary value problems arising in optimal control have been studied in [LMS86] and [Spi86].