# 5

# Network Flow Problems

Network flow problems are the most frequently solved class of optimization problems. They include as special cases combinatorial problems such as the assignment, transportation, max–flow, and shortest path problems, and they arise naturally in the analysis of large systems such as manufacturing, communication, and transportation networks. Among network problems, those involving linear cost are by far the most common. Traditionally, these problems are solved using primal–simplex (see [Dan63]), primal–dual (see [FoF62]), or relaxation methods (see [Ber82c], [BeT85], and Section 5.2). Highly sophisticated codes based on these methods are presently available [AaM76], [BBG77], [BeT85], [GKK74], [GrH80], [KeH80], and [Mul78]. Unfortunately, none of these methods seems inherently well–suited for parallel computation. In this chapter, we consider several methods that are similar in spirit to the Gauss–Seidel and Jacobi methods examined in Chapters 2 and 3. They are based on a dual network flow optimization problem involving a single dual variable per network node. At each iteration, a single node is chosen, and its dual variable or its incident arc flows are changed in an attempt to improve the dual cost. This approach is well suited for massive parallelization, whereby each node is a processor adjusting its own dual variable on the basis of local information communicated by adjacent processors/nodes.

In Section 5.1, we introduce the linear network flow problem (otherwise known as the *transshipment* or *minimum cost flow* problem), along with several important special cases. We formulate a dual problem based on linear programming duality theory. This is

an unconstrained optimization problem with a structure that is suitable for application of the Gauss–Seidel relaxation method discussed in Section 3.2, but with a cost function that violates the differentiability requirement of the analysis of that section. We explain the associated difficulties in Section 5.2, and we discuss potential approaches for overcoming these difficulties. A highly parallelizable method, called *$\epsilon$–relaxation*, is introduced in Section 5.3. A closely related method for the assignment problem, called the *auction* algorithm, is also discussed. The complexity analysis of these methods is given in Section 5.4.

The last two sections of the chapter deal with nonlinear problems. In Section 5.5, we consider nonlinear network flow problems with strictly convex arc costs, and the associated duality theory. Here the dual cost is differentiable, so the application of the Gauss–Seidel relaxation method is much easier than for the linear cost case. Nonetheless, the analysis is nontrivial because the dual cost is not strictly convex, and the results of Section 3.2 for the nonlinear Gauss–Seidel method do not apply. The algorithm can be extended to the class of convex problems with separable cost and linear constraints, also known as *monotropic programming problems* [Roc84], but we will not go into this here; see [TsB87a]. It turns out that the Jacobi version of the relaxation algorithm is not guaranteed to converge to an optimal solution. We will show in Sections 6.6 and 7.2 that modified relaxation algorithms are convergent when either a Jacobi–like or an asynchronous implementation is used. Finally, in Section 5.6, we consider nonlinear multicommodity network flow problems of the type often arising in communication and transportation networks. The method considered here is based on the gradient projection method discussed in Section 3.3.

Throughout this chapter, we consider synchronous algorithms. Totally and partially asynchronous versions are discussed in the next two chapters.

## 5.1 THE LINEAR NETWORK FLOW PROBLEM AND ITS DUAL

The optimal network flow problem with linear arc costs of this section is a special case of a linear programming problem, and the duality formulation to be given is a special case of linear programming duality (see Appendix C).

Consider a directed graph with a set of nodes $N$ and a set of arcs $A$. Each arc $(i,j)$ has associated with it an integer $a_{ij}$ referred to as the cost coefficient of $(i,j)$. Let $f_{ij}$ be the flow of the arc $(i,j)$, and consider the problem

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

$$\text{subject to} \quad \sum_{\{j|(i,j)\in A\}} f_{ij} - \sum_{\{j|(j,i)\in A\}} f_{ji} = s_i, \qquad \forall\, i \in N, \tag{1.1}$$

$$b_{ij} \leq f_{ij} \leq c_{ij}, \qquad \forall\, (i,j) \in A, \tag{1.2}$$

where $a_{ij}$, $b_{ij}$, $c_{ij}$, and $s_i$ are given integers. We refer to $b_{ij}$ and $c_{ij}$, and the interval $[b_{ij}, c_{ij}]$ as the *flow bounds* and the *feasible flow range* of arc $(i, j)$, respectively. We refer to $s_i$ as the *supply* of node $i$ and to $-s_i$ as the *demand* of node $i$. We refer to the constraints (1.1) and (1.2) as the *conservation of flow constraints* and the *capacity constraints* respectively. We assume that $\sum_{i \in N} s_i = 0$; by adding the constraints (1.1), we see that this condition is necessary for problem feasibility. As discussed in Appendix B, we also assume that there is at most one arc in each direction between any pair of nodes; this assumption is made in order to simplify notation and can be easily dispensed with (Exercise 1.5). The numbers of nodes and arcs are denoted $|N|$ and $|A|$, respectively.

The linear network flow problem (LNF) is a classical problem that has been studied extensively. The assignment, max–flow, and shortest path problems are special cases, as shown in Figs. 5.1.1 through 5.1.3. These problems are particularly interesting within our context, each for different reasons.

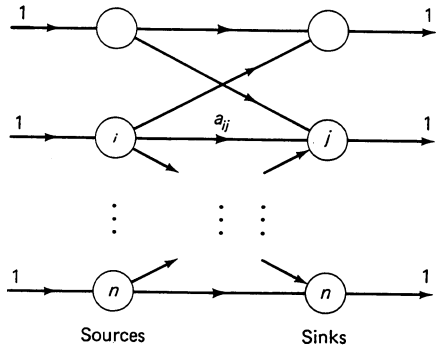



**Figure 5.1.1** In the assignment problem, there are $2n$ nodes, $n$ of which (the sources) have supply of 1 unit of flow, and the other $n$ (the sinks) have a demand of 1 unit. Each arc $(i, j)$ connects a source $i$ to a sink $j$ and has an arc cost coefficient $a_{ij}$. All arcs flows must satisfy $0 \leq f_{ij} \leq B$, where $B$ is a given positive integer. [An alternative and equivalent formulation uses only the constraint $0 \leq f_{ij}$; the resulting problem, however, is not a special case of the linear network flow problem (LNF).] The problem is

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

$$\text{subject to} \quad \sum_{\{j \mid (i,j) \in A\}} f_{ij} = 1, \qquad \forall\, i = 1, \ldots, n,$$

$$\sum_{\{i \mid (i,j) \in A\}} f_{ij} = 1, \qquad \forall\, j = 1, \ldots, n,$$

$$0 \leq f_{ij} \leq B, \qquad \forall\, (i, j) \in A.$$

It will be shown as a by–product of our analysis in the next section that if there exists a feasible solution for problem (LNF), there exists an *integer* optimal solution for both (LNF) and its dual. For the assignment problem, such an optimal solution consists of arc flows that are zero or one, and assigns exactly one source to each sink.

The assignment problem can be viewed as the canonical special case because it can be shown that problem (LNF) can be transformed into an assignment problem (see Fig. 5.1.4). This means that algorithms that can be developed or understood using the rich intuition afforded by the assignment problem can be made to work for the general problem (LNF).

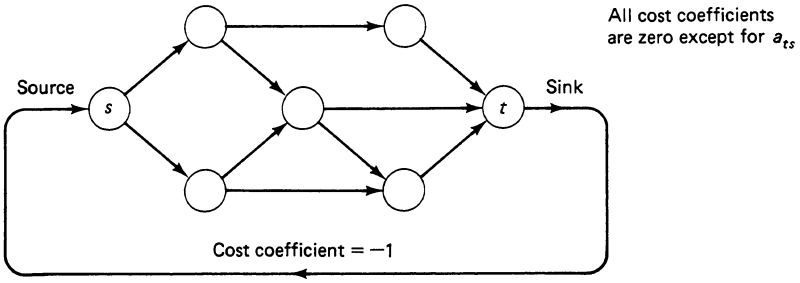



**Figure 5.1.2** In the max–flow problem, there are two special nodes, the *source* ($s$) and the *sink* ($t$), which are connected by an arc $(t, s)$. All arcs have zero cost coefficient except for $(t, s)$ which has a cost coefficient of $-1$. The lower bound for all arc flows is zero, and the upper bound $c_{ts}$ is greater than or equal to $\sum_i c_{it}$. We have $s_i = 0$ for all nodes $i$. At the optimum, the flow $f_{ts}$ equals the maximum flow that can be sent from $s$ to $t$ through the subgraph obtained by deleting arc $(t, s)$.

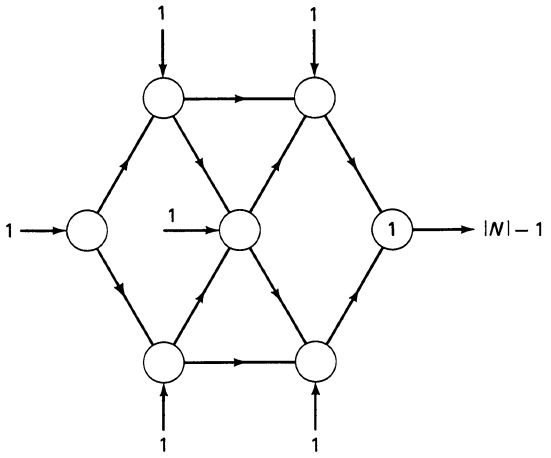



**Figure 5.1.3** In the shortest path problem, each arc $(i, j)$ has a length $a_{ij}$, and the objective is to find a minimum length simple path from every node to node 1. We can write the problem into the form of the linear network flow problem (LNF) as follows

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

$$\text{subject to} \quad \sum_{\{j\,|(i,j)\in A\}} f_{ij} - \sum_{\{j\,|(j,i)\in A\}} f_{ji} = 1,$$

$$\forall\, i = 2, \ldots, |N|,$$

$$\sum_{\{j|(j,1)\in A\}} f_{j1} = |N| - 1,$$

$$0 \le f_{ij} \le |N|, \quad \forall\, (i,j) \in A.$$

This equivalence holds provided every cycle has a positive length.

The max–flow problem is characterized by the fact that the range of values of arc cost coefficients is small ($[-1, 0]$; see Fig. 5.1.2), and this will turn out to be significant within the context of the $\epsilon$–relaxation algorithm of Secion 5.3.

Finally, the shortest path problem is fundamental in the analysis of the linear network flow problem (LNF) and often appears as a subroutine in various algorithms for solving (LNF). We will see in the next section that the relaxation method for solving the shortest path problem is closely related to the Bellman–Ford algorithm discussed in Section 4.3.

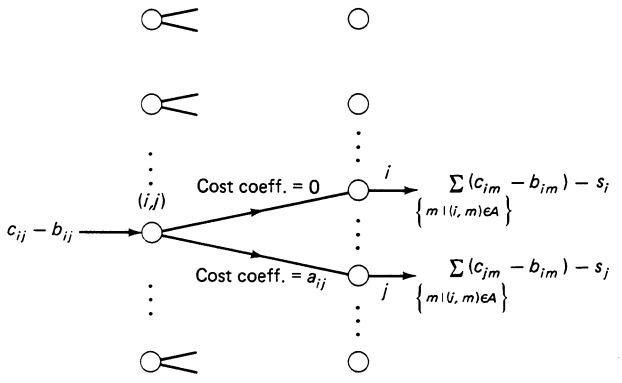



**Figure 5.1.4** Transformation of the general problem (LNF) into a transportation problem, that is, a problem of the form

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

subject to

$$\sum_{\{j \mid (i,j)\in A\}} f_{ij} = \alpha_i, \forall\, i = 1, \ldots, m,$$

$$\sum_{\{i \mid (i,j)\in A\}} f_{ij} = \beta_j, \forall\, j = 1, \ldots, n,$$

$$0 \le f_{ij}, \qquad \forall\, (i,j) \in A,$$

where $\alpha_i$, and $\beta_j$ are positive integers. In the transportation problem we take as sources the arcs of the original network, and as sinks the nodes of the original network. Each transportation problem source has two outgoing arcs with cost coefficients as shown. The supply of each transportation problem source is the feasible flow range length of the corresponding original network arc. The demand of each transportation problem sink is the sum of the feasible flow range lengths of the outgoing arcs from the corresponding original network node minus the supply of that node as shown. An arc flow $f_{ij}$ in (LNF) corresponds to flows equal to $f_{ij}$ and $c_{ij} - b_{ij} - f_{ij}$ on the transportation problem arcs $\big((i,j),j\big)$ and $\big((i,j),i\big)$ respectively. The transportation problem can be converted to an assignment problem by creating $\alpha_i$ unit supply sources ($\beta_j$ unit capacity sinks) for each transportation problem source $i$ (sink $j$ respectively).

### The Dual Problem

We formulate a dual problem associated with (LNF) by associating a Lagrange multiplier $p_i$ with the $i$th conservation of flow constraint (1.1) as discussed in Appendix C. By denoting by $f$ and $p$ the vectors with elements $f_{ij}$, $(i,j) \in A$, and $p_i$, $i \in N$, respectively, we can write the corresponding Lagrangian function as

$$\begin{aligned} L(f,p) &= \sum_{(i,j)\in A} a_{ij} f_{ij} + \sum_{i\in N} p_i \left( \sum_{\{j \mid (j,i)\in A\}} f_{ji} - \sum_{\{j \mid (i,j)\in A\}} f_{ij} + s_i \right) \\ &= \sum_{(i,j)\in A} (a_{ij} + p_j - p_i) f_{ij} + \sum_{i\in N} s_i p_i. \end{aligned} \tag{1.3}$$

The dual function value $q(p)$ at a vector $p$ is obtained by minimizing $L(f,p)$ over all $f$ satisfying the capacity constraints (1.2). This leads to the dual problem

$$\begin{aligned} &\text{maximize} \quad q(p) \\ &\text{subject to no constraint on } p, \end{aligned} \tag{1.4}$$

with the dual functional $q$ given by

$$q(p) = \min_{\substack{b_{ij} \le f_{ij} \le c_{ij} \\ (i,j) \in A}} L(f,p) = \sum_{(i,j) \in A} q_{ij}(p_i - p_j) + \sum_{i \in N} s_i p_i, \tag{1.5}$$

where each $q_{ij}$ is the scalar function defined by

$$q_{ij}(p_i - p_j) = \min_{b_{ij} \le f_{ij} \le c_{ij}} (a_{ij} + p_j - p_i) f_{ij}. \tag{1.6}$$

The function $q_{ij}$ is shown in Fig. 5.1.5. We henceforth refer to (LNF) as the *primal problem*, and note that based on the duality results of Appendix C, the optimal primal cost equals the optimal dual cost. The dual variable $p_i$ will be referred to as the *price* of node $i$.

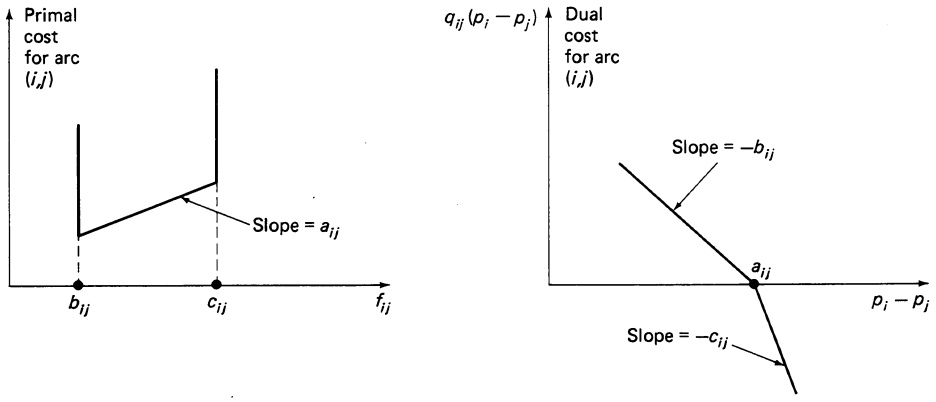



**Figure 5.1.5** Primal and dual costs for arc $(i,j)$. Note that the break points for each function correspond to slopes of linear segments in the other function.

For any flow vector $f$ and node $i$, the scalar

$$g_i = \sum_{\{j \mid (j,i) \in A\}} f_{ji} - \sum_{\{j \mid (i,j) \in A\}} f_{ij} + s_i \tag{1.7}$$

is called the *surplus* of node $i$. It represents the difference of total flow imported and total flow exported by the node. The conservation of flow constraint (1.1) is also written as

$$g_i = 0, \qquad \forall\ i \in N. \tag{1.8}$$

The necessary and sufficient conditions for a pair $(f,p)$ to be an optimal primal and dual solution pair are primal feasibility and complementary slackness [relations (C.16) and (C.17) in Appendix C]. To state these conditions, we first introduce some terminology. For any price vector $p$, we say that an arc $(i,j)$ is

$$\begin{aligned} \text{Inactive if} \quad & p_i < a_{ij} + p_j, \\ \text{Balanced if} \quad & p_i = a_{ij} + p_j, \\ \text{Active if} \quad & p_i > a_{ij} + p_j. \end{aligned}$$

We say that a pair $(f, p)$ satisfies *complementary slackness* (abbreviated CS) if

$$f_{ij} = b_{ij}, \qquad \text{for all inactive arcs } (i,j), \tag{1.9}$$

$$b_{ij} \le f_{ij} \le c_{ij}, \qquad \text{for all balanced arcs } (i,j), \tag{1.10}$$

$$f_{ij} = c_{ij}, \qquad \text{for all active arcs } (i,j), \tag{1.11}$$

(see Fig. 5.1.6). Notice that a pair $(f, p)$ satisfies CS if and only if each $f_{ij}$ attains the minimum in the definition of the dual arc cost of Eq. (1.6). The optimality conditions of Appendix C translated to our problem yield the following result:

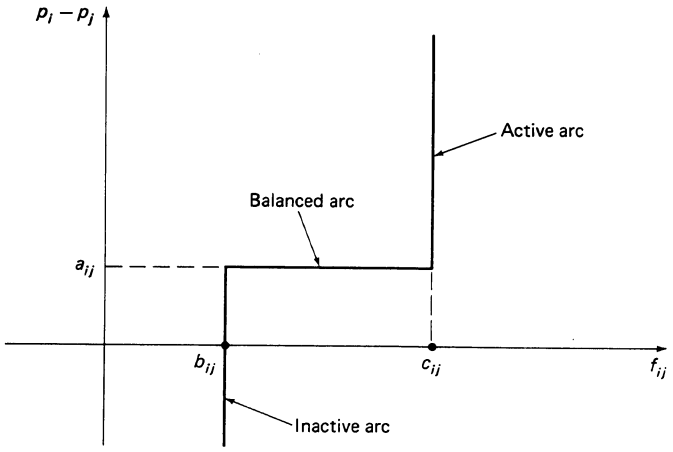



**Figure 5.1.6** Illustration of the complementary slackness conditions (1.9)–(1.11). For each arc $(i,j)$, the pair of flow $f_{ij}$ and price differential $p_i - p_j$ should lie on the diagram shown.

**Proposition 1.1.** (*Optimality Conditions*)

**(a)** A flow vector $f$ is primal optimal if and only if it is primal feasible, and there exists a price vector $p$ that together with $f$ satisfies the CS conditions (1.9)–(1.11).

**(b)** A flow vector $f$ and a price vector $p$ are primal and dual optimal, respectively, if and only if they satisfy the conservation of flow constraint (1.8) and the CS conditions (1.9)–(1.11).

An alternative optimality condition is given in Exercise 1.2. Another important property is that if there exists a feasible solution for problem (LNF), then there exist

*integer* optimal solutions to both (LNF) and its dual. We will show this constructively in the next section by means of a method that computes such solutions.

## EXERCISES

**1.1.** Let $p$ be any price vector, and consider a problem that is the same as the linear network flow problem (LNF) except that the cost coefficient of each arc $(i, j)$ is $a_{ij} + p_j - p_i$ instead of $a_{ij}$. Show that this problem has the same optimal solutions as (LNF), and that its optimal cost is less than the optimal cost of (LNF) by $\sum_{i \in N} p_i s_i$.

**1.2.** Show that a flow vector $f$ is optimal for (LNF) if and only if $f$ is feasible, and for every directed cycle $Y$ for which

$$f_{ij} < c_{ij}, \qquad \forall\ (i,j) \in Y^+,$$

$$b_{ij} < f_{ij}, \qquad \forall\ (i,j) \in Y^-,$$

we have

$$\sum_{(i,j) \in Y^+} a_{ij} - \sum_{(i,j) \in Y^-} a_{ij} \geq 0,$$

where $Y^+$ and $Y^-$ are the sets of forward and backward arcs of $Y$, respectively. *Hint:* Use the optimality condition of Prop. 3.1 in Section 3.3, and the Conformal Realization Theorem of Appendix B; see also the proof of Prop. 3.1 in Section 5.3.

**1.3.** In some situations, it is useful to have a price vector for which all arcs are inactive or balanced; for example, to convert the problem to one where all the arc cost coefficients are nonnegative (see Exercise 1.1). This exercise characterizes situations where this is possible, and shows that a corresponding price vector can be obtained by solving an assignment problem.

**(a)** Consider the assignment problem of Fig. 5.1.1 for the case where $B > 1$. Show that the dual problem can be written as

$$\text{maximize} \quad \sum_{i=1}^{n} r_i - \sum_{j=1}^{n} p_j$$

$$\text{subject to} \quad a_{ij} + p_j \geq r_i, \qquad \forall\ (i,j) \in A.$$

**(b)** Consider the linear network problem (LNF). Assume that each cycle that consists of forward arcs (a positive cycle) has a nonnegative arc cost sum. Show that a price vector such that all arcs are inactive or balanced [$a_{ij} + p_j \geq p_i$, for all $(i, j) \in A$] can be found by solving the dual of an assignment problem. Conversely, show that if one can find a price vector such that all arcs are inactive or balanced, then each cycle consisting of forward arcs has a nonnegative arc cost sum. *Hint:* For each node $i$, create a source $i$ and a sink $i'$. For each arc $(i, j)$ with cost $a_{ij}$, create an arc from

source $i$ to sink $j'$ with cost $a_{ij}$. Create also an arc $(i, i')$ with zero cost for each $i$. Use the result of Exercise 1.2 to show that assigning $i$ to $i'$ for each $i$ is an optimal solution if and only if each positive cycle has a nonnegative arc cost sum.

**1.4.** There are several ways to convert problem (LNF) to a problem where the flow is constrained to be a circulation. One way is to introduce a new node $v$ and for every node $i$ with $s_i \neq 0$, introduce an arc $(v, i)$ with zero arc cost coefficient and a feasible arc flow range $[s_i, s_i]$. This exercise provides a different conversion, which has the property that if the zero flow vector satisfies the arc capacity constraints of (LNF), then the zero flow vector is a feasible circulation after the conversion; this is sometimes useful in algorithms and theoretical studies. Introduce two new nodes $v$ and $w$. For each node $i$ with $s_i > 0$, introduce an arc $(v, i)$ with zero arc cost coefficient and feasible arc flow range $[0, s_i]$. For each node $i$ with $s_i < 0$ introduce an arc $(i, w)$ with zero arc cost coefficient and feasible arc flow range $[0, -s_i]$. Introduce an arc $(w, v)$ with feasible arc flow range $[0, c_{wv}]$, where $c_{wv} \geq \frac{1}{2}\sum_{i \in N} |s_i|$. Show that an optimal flow vector of the problem involving the enlarged network yields an optimal flow vector of (LNF) if the arc cost coefficient $a_{wv}$ is less than $-L$, where $L$ is defined as follows: for every simple path $p$ from $v$ to $w$, let $L_p$ be the sum of the forward arc costs of the path minus the backward arc costs of the path, and let $L = \max_p L_p$. *Hint:* Consider an optimal flow vector $f$ of the enlarged network problem, and assume that $f_{wv} < \frac{1}{2}\sum_{i \in N} |s_i|$. Use the result of Exercise 1.2 and the Conformal Realization Theorem to arrive at a contradiction.

**1.5.** Show that the version of problem (LNF) where multiple arcs are allowed between a pair of nodes can be converted to the single arc version by introducing an extra node for each multiple arc.

**1.6.** Show how to convert a network problem with convex piecewise linear arc costs to one with linear arc costs. *Hint:* Convert each arc with a piecewise linear cost into several arcs with linear costs. Use also Exercise 1.5.

**1.7.** **(A Method of Multipliers for Transportation Problems.)** Consider the (capacitated) transportation problem

$$\begin{aligned}
\text{minimize} \quad & \sum_{(i,j) \in A} a_{ij} f_{ij} \\
\text{subject to} \quad & \sum_{\{j \mid (i,j) \in A\}} f_{ij} = \alpha_i, \qquad \forall\, i = 1, \ldots, m, \\
& \sum_{\{i \mid (i,j) \in A\}} f_{ij} = \beta_j, \qquad \forall\, j = 1, \ldots, n, \\
& 0 \leq f_{ij} \leq c_{ij}, \qquad \forall\, (i,j) \in A,
\end{aligned}$$

and the first of the two methods of multipliers introduced in Example 4.5 of Subsection 3.4.4. Show that the iteration that minimizes the Augmented Lagrangian has the form

$$f_{ij} := \left[ f_{ij} + \frac{1}{2c(t)} \big( r_i(t) + p_i(t) - a_{ij} + c(t)(y_i + w_j) \big) \right]^+, \qquad \forall\, (i,j) \in A,$$

where $[\cdot]^+$ denotes projection on the interval $[0, c_{ij}]$, and $y_i$ and $w_j$ are given in terms of $f$ by

$$y_i = \alpha_i - \sum_{\{j|(i,j)\in A\}} f_{ij}, \qquad \forall\, i = 1,\ldots,m,$$

$$w_j = \beta_j - \sum_{\{i|(i,j)\in A\}} f_{ij}, \qquad \forall\, j = 1,\ldots,n.$$

At the end of the minimization yielding $f_{ij}(t+1)$, $y_i(t+1)$, and $w_j(t+1)$, the prices $r_i$ and $p_j$ are updated according to

$$r_i(t+1) = r_i(t) + c(t)y_i(t+1), \qquad \forall\, i = 1,\ldots,m,$$

$$p_j(t+1) = p_j(t) + c(t)w_j(t+1), \qquad \forall\, j = 1,\ldots,n.$$

**1.8. (An Alternating Direction Method for Transportation Problems.)** Consider the transportation problem of Exercise 1.7. Show that the following iteration is a special case of the alternating direction method of multipliers given in Subsection 3.4.4:

$$f_{ij}(t+1) := \Big[f_{ij}(t) + \frac{1}{2c}\big[r_i(t) + p_i(t) - a_{ij} + c(y_i(t) + w_j(t))\big]\Big]^+, \qquad \forall\,(i,j)\in A,$$

$$r_i(t+1) = r_i(t) + cy_i(t+1), \qquad \forall\, i = 1,\ldots,m,$$

$$p_j(t+1) = p_j(t) + cw_j(t+1), \qquad \forall\, j = 1,\ldots,n,$$

where $y_i(t)$ and $w_j(t)$ are given in terms of $f(t)$ by

$$y_i(t) = \frac{1}{d_i}\Big(\alpha_i - \sum_{\{j|(i,j)\in A\}} f_{ij}(t)\Big), \qquad \forall\, i = 1,\ldots,m,$$

$$w_j(t) = \frac{1}{d_j}\Big(\beta_j - \sum_{\{i|(i,j)\in A\}} f_{ij}(t)\Big), \qquad \forall\, j = 1,\ldots,n,$$

and $d_i$ (or $d_j$) is the number of incident arcs to $i$ (or $j$, respectively).

## 5.2 THE RELAXATION METHOD

Consider the dual cost functional given by

$$q(p) = \sum_{(i,j)\in A} q_{ij}(p_i - p_j) + \sum_{i\in N} s_i p_i, \tag{2.1a}$$

where

$$q_{ij}(p_i - p_j) = \begin{cases} (a_{ij} + p_j - p_i)b_{ij}, & \text{if } a_{ij} + p_j - p_i \geq 0, \\ (a_{ij} + p_j - p_i)c_{ij}, & \text{if } a_{ij} + p_j - p_i < 0, \end{cases} \tag{2.1b}$$

[cf. Eqs. (1.5), (1.6) and Fig. 5.1.5]. An important characteristic of this function is that it has a separable form that motivates solution by a Gauss–Seidel relaxation (or coordinate ascent) method. In this section, we explore various possibilities in this direction. The idea is to choose a single node $i$, and to change its price $p_i$ in a direction of improvement of the dual cost while keeping the other prices unchanged. Since $p_i$ appears only in the terms $q_{ij}, (i,j) \in A$, and $q_{ji}, (j,i) \in A$, in the dual cost expression (2.1), the change in $p_i$ requires a relatively small amount of computation.

To understand how to implement this type of iteration, we first focus on the dual cost $q(p)$ along a single price coordinate $p_i$, that is, for a given price vector $p$, we consider the function of the scalar $\xi$

$$Q_p^i(\xi) = q\big(p_1, \ldots, p_{i-1}, \xi, p_{i+1}, \ldots, p_{|N|}\big).$$

This function is the sum of linear and piecewise linear functions [cf. Eq. (2.1)], and is, therefore, piecewise linear as shown in Fig. 5.2.1. It is seen that the break points of the function correspond to the values of $\xi$ at which one or more arcs incident to node $i$ are balanced, that is, the values

$$\begin{aligned} &p_j + a_{ij} \qquad \text{for outgoing arcs } (i,j) \in A, \\ &p_j - a_{ji} \qquad \text{for incoming arcs } (j,i) \in A. \end{aligned}$$

At a break point of $Q_p^i(\xi)$, we must distinguish between the left derivative $g_i^-$ and the right derivative $g_i^+$ of $Q_p^i(\xi)$. In view of the dual cost equations (2.1) and Fig. 5.1.5, for given values of $p$ and $\xi$, we have

$$\begin{aligned} g_i^+ = {} & \sum_{(j,i)\text{: active}} c_{ji} + \sum_{(j,i)\text{: inactive or balanced}} b_{ji} \\ & - \sum_{(i,j)\text{: active or balanced}} c_{ij} - \sum_{(i,j)\text{: inactive}} b_{ij} + s_i, \end{aligned} \tag{2.2a}$$

$$\begin{aligned} g_i^- = {} & \sum_{(j,i)\text{: active or balanced}} c_{ji} + \sum_{(j,i)\text{: inactive}} b_{ji} \\ & - \sum_{(i,j)\text{: active}} c_{ij} - \sum_{(i,j)\text{: inactive or balanced}} b_{ij} + s_i. \end{aligned} \tag{2.2b}$$

Note that within an open interval where $Q_p^i(\xi)$ is linear, the right and left derivatives are equal ($g_i^+ = g_i^-$), and there are no balanced arcs. Note also that $g_i^+$ ($g_i^-$) are the minimum (maximum, respectively) surplus of node $i$ over all flow vectors that satisfy complementary slackness (CS) together with $p$ [cf. conditions (1.9)–(1.11) in the preceding section].

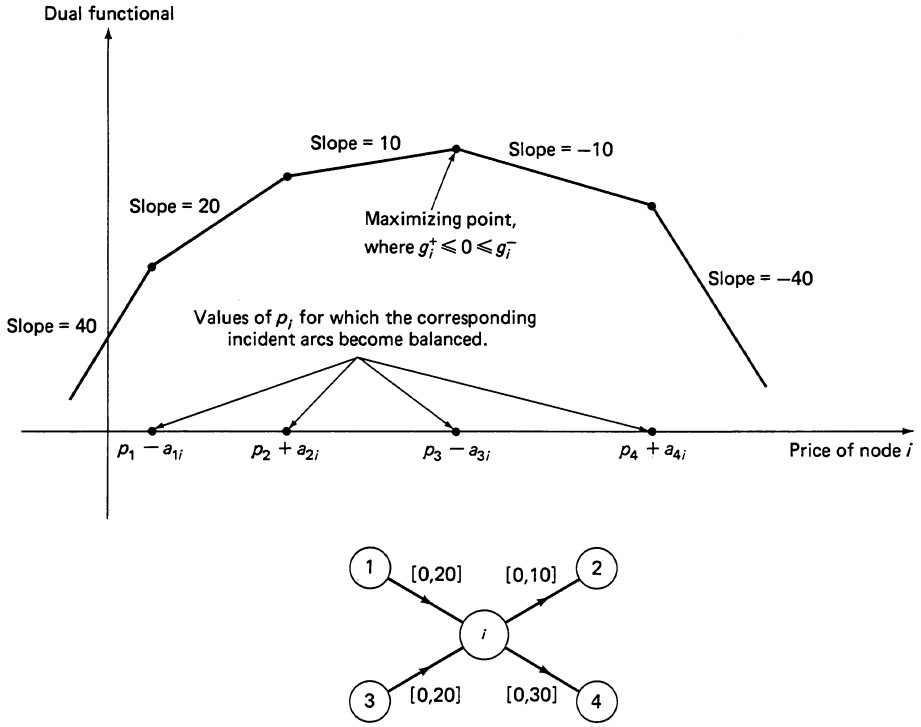



**Figure 5.2.1** Illustration of $Q_p^i(\xi)$, the dual cost along price $p_i$, together with the left and right derivatives $g_i^-$ and $g_i^+$, respectively (all other prices are kept fixed). Here node $i$ has four incident arcs $(1,i)$, $(3,i)$, $(i,2)$, $(i,4)$ with flow ranges [0,20], [0,20], [0,10], and [0,30], respectively, and $s_i = 0$. The break points of the dual cost correspond to the values of $p_i$ at which one or more incident arcs to node $i$ become balanced. For values of $p_i$ strictly between two successive break points, there are no balanced arcs.

A direct analog of the Gauss–Seidel relaxation method of Chapters 2 and 3 is obtained by choosing at each iteration a node $i$ and by changing the price $p_i$ so as to optimize the dual cost, that is,

$$p_i := \arg \max_{\xi} Q_p^i(\xi) = \arg \max_{\xi} q\big(p_1, \ldots, p_{i-1}, \xi, p_{i+1}, \ldots, p_{|N|}\big).$$

In particular, $p_i$ is changed to a break point for which

$$g_i^+ \leq 0 \leq g_i^- \tag{2.3}$$

(see Fig. 5.2.1).

The algorithm terminates when the condition $g_i^+ \leq 0 \leq g_i^-$ holds for all $i \in N$. If the starting prices are integer and the optimal dual value is finite (as it will be when the primal problem is feasible; see Appendix C), the algorithm will terminate after a finite number of iterations. To see this, first note that throughout the algorithm the node prices

and the break points of all functions $Q_p^i(\xi)$ are integer. Next observe that each time a node price changes, there will be an improvement of the dual cost by an integer amount, since both the price increment and the corresponding rate of improvement of the dual cost are integer. Therefore, there can only be a finite number of dual cost improvements and the algorithm must terminate finitely.

The difficulty with this algorithm is that it can terminate at a nonoptimal price vector, depending on the starting price vector and the problem data. This is illustrated graphically in Fig. 5.2.2 and analytically in the context of the shortest path problem in the following subsection.

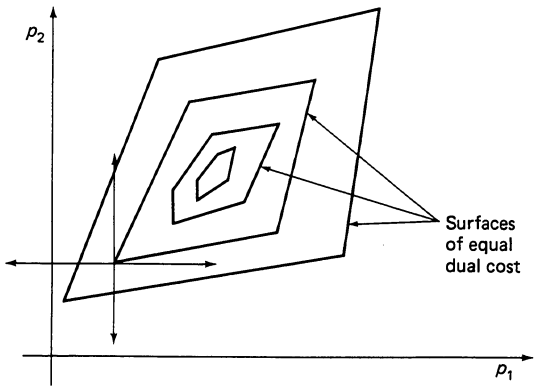



**Figure 5.2.2** The difficulty with the relaxation method. At the indicated corner point, it is impossible to improve the dual cost by changing any *single* price coordinate.

### 5.2.1 Application to the Shortest Path Problem

Consider the shortest path problem formulation shown in Fig. 5.1.3, where node 1 is the destination. It can be shown that if $p^*$ is a dual optimal price vector, then $p_i^* - p_1^*$ is the shortest distance from $i$ to 1. Indeed, by the optimality conditions of Prop. 1.1(b), $p^*$ satisfies CS together with every primal optimal flow vector. We can construct such a flow vector by sending the supply of each node $i$ to the destination node 1 along a shortest path from $i$ to 1, and by accumulating the total flow on each arc. Each arc of this shortest path must be balanced [it cannot be inactive because its flow is positive and it cannot be active because no optimal arc flow can exceed $|N| - 1$ (the sum of all node supplies) while the arc flow upper bound is $|N|$]. By adding the condition $p_i^* - p_j^* = a_{ij}$ for a balanced arc $(i, j)$ along a shortest path, we obtain that $p_i^* - p_1^*$ equals the length of the shortest path.

The left and right derivatives of the dual cost $Q_p^i(\xi)$ along the $i$th price coordinate are given by [cf. Eq. (2.2)]

$$g_i^- = 1 + |N| \cdot [\text{number of active or balanced arcs (j,i)} - \text{number of active arcs (i,j)}], \quad \text{if } i \neq 1,$$

$$g_1^- = 1 - |N| + |N| \cdot [\text{number of active or balanced arcs (j,1)}],$$

$g_i^+ = 1 + |N| \cdot$ [number of active arcs (j,i) $-$ number of active or balanced arcs (i,j)], if $i \neq 1$,

$g_1^+ = 1 - |N| + |N| \cdot$ [number of active arcs (j,1)].

Consider the relaxation algorithm whereby at each iteration, a single node price $p_i$ is changed so as to maximize the dual cost along $p_i$, that is, $p_i$ is set to a point where the condition $g_i^+ \leq 0 \leq g_i^-$ is satisfied [cf. Eq. (2.3)]. It is seen from the above equations that this can be accomplished by setting the price $p_i$ of a node $i \neq 1$ to the smallest breakpoint at which there are fewer active incoming arcs $(j, i)$ than active and balanced outgoing arcs $(i, j)$; the price $p_1$ should be set to the value at which there is no active arc $(j, 1)$, and there are one or more than one balanced arc $(j, 1)$.

Suppose now that a price vector $p$ is such that all arcs are inactive or balanced, that is,

$$\max_{\{j|(j,i)\in A\}} (p_j - a_{ji}) \leq p_i \leq \min_{\{j|(i,j)\in A\}} (p_j + a_{ij}), \qquad \forall\, i \in N. \tag{2.4}$$

[Such a price vector exists assuming that all cycles have positive length; see Exercise 1.3 in the previous section. In particular, if $a_{ij} \geq 0$ for all arcs $(i, j)$, it is seen that for the zero price vector, all arcs are inactive or balanced.] It can be seen then (Fig. 5.2.3) that, for $i \neq 1$, the relaxation iteration sets $p_i$ to the smallest value such that at least one outgoing arc becomes balanced, and takes the form

$$p_i := \min_{\{j|(i,j)\in A\}} (p_j + a_{ij}), \qquad \forall\, i \neq 1, \tag{2.5}$$

and that, for $i = 1$, the iteration takes the form

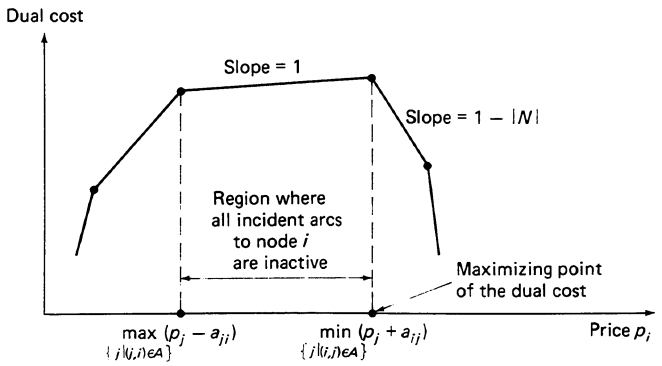



**Figure 5.2.3** Illustration of the relaxation iteration at a node $i \neq 0$ when the condition

$$\max_{\{j|(j,i)\in A\}} (p_j - a_{ji}) \leq p_i \leq \min_{\{j|(i,j)\in A\}} (p_j + a_{ij}), \qquad \forall\, i \in N,$$

[cf. Eq. (2.4)] holds in the shortest path problem.

$$p_1 := \max_{\{j|(j,1)\in A\}}(p_j - a_{j1}). \tag{2.6}$$

It can be seen also that this iteration creates no active arcs. Thus, if initially there are no active arcs, then the same will be true for all iterations, and the relaxation algorithm will be given by the preceding Eqs. (2.5) and (2.6). Under these circumstances, it is seen that the prices $p_i$, $i \neq 1$, cannot decrease as a result of iteration (2.5) since for this to happen, it is necessary that some arc $(i, j)$ is active. Similarly, the price $p_1$ cannot increase as a result of iteration (2.6) since for this to happen, it is necessary that some arc $(j, 1)$ is active. Furthermore, since the prices $p_j$, $j \neq 1$, cannot decrease, it follows that the price $p_1$ can decrease at most once as a result of iteration (2.6). Therefore, following the first execution of iteration (2.6), the price $p_1$ will be constant and the algorithm will become essentially equivalent to the Bellman–Ford algorithm discussed in Section 4.1. Upon termination (which, as discussed earlier, will occur if the shortest path problem has at least one feasible solution), it will yield a vector $p^*$ satisfying

$$p_i^* = \min_{\{j|(i,j)\in A\}}(p_j^* + a_{ij}), \qquad \forall\, i \neq 1.$$

By subtracting $p_1^*$ from both sides of this equation, we see that the scalars

$$x_i^* = p_i^* - p_1^*, \qquad i \in N$$

satisfy Bellman's equation

$$x_i^* = \min_{\{j|(i,j)\in A\}}(a_{ij} + x_j^*), \qquad \forall\, i \neq 1,$$

$$x_1^* = 0.$$

This guarantees that for all nodes $i$, $x_i^*$ is the shortest distance from $i$ to 1 under the condition that all cycles have a positive length (Section 4.1).

In conclusion, the relaxation algorithm of this section, when applied to the shortest path problem, is essentially equivalent to the Bellman–Ford method provided that there are no active arcs for the initial price vector [cf. Eq. (2.4)].

Unfortunately, the relaxation method may not work well for other initial price vectors. Figure 5.2.4 gives an example where it terminates with the wrong answer. It is thus necessary to modify the basic relaxation algorithm in order to improve its convergence properties. Two ways for doing so will be given. The first method, presented in the next subsection, leads to good practical sequential algorithms but does not naturally lend itself to parallelization. The second method is the subject of the next two sections, and is highly parallelizable.

### 5.2.2 Multiple Node Relaxation Method

The idea here is to change the prices of several nodes simultaneously when a search along a single price coordinate does not produce a dual cost improvement (see Fig.

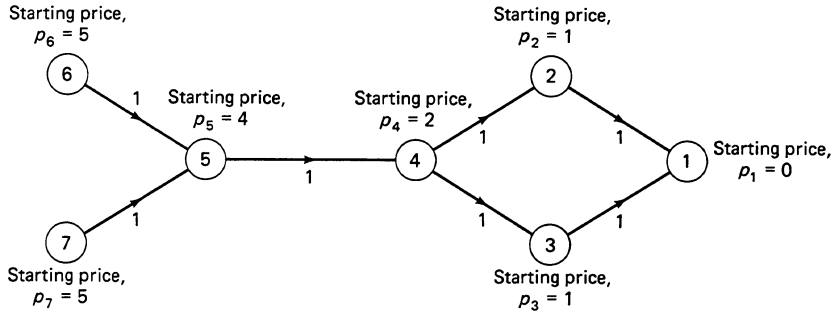



**Figure 5.2.4** Example of a shortest path problem where the relaxation iteration terminates with the wrong answer when the initial price vector violates condition (2.4), that is, there is at least one active arc. The starting prices are as shown. All arcs have unity length, and they are all balanced except for arc (5,4), which is active. It can be verified that for each node $i$, we have $g_i^+ \leq 0 \leq g_i^-$, so the relaxation method terminates without obtaining an optimal price vector.

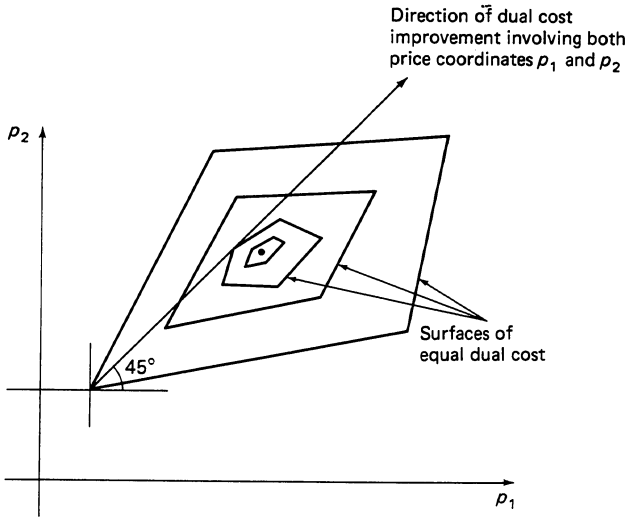



**Figure 5.2.5** Illustration of the idea of the multiple node relaxation method. At the indicated corner point, it is impossible to improve the dual cost by changing a single price coordinate. A dual cost improvement is effected by changing simultaneously both price coordinates by equal amounts; this corresponds to dual ascent along the direction of the form (2.7) involving nodes 1 and 2.

5.2.5). Equivalently, at a nonoptimal price vector $p$, we would like to use some direction of dual cost improvement, similarly as we did in gradient methods for optimization in Chapter 3. A remarkable fact about our problem, which will be shown shortly, is that such a direction of improvement can be found within the finite set of directions having the form

$$d_L = \left(d_1, d_2, \ldots, d_{|N|}\right), \tag{2.7a}$$

where

$$d_i = \begin{cases} 1 & \text{if } i \in L, \\ 0 & \text{if } i \notin L, \end{cases} \tag{2.7b}$$

and $L \subset N$ is the node set of a connected subgraph of the graph $(N, A)$ with $L \neq N$.

From the dual cost expression (2.1) and Fig. 5.1.5, it is seen that the directional derivative of the dual cost along a vector $d_L$ of the form (2.7) is given by

$$\begin{aligned} q'(p; d_L) &= \lim_{\alpha \downarrow 0} \frac{q(p + \alpha d_L) - q(p)}{\alpha} \\ &= \sum_{(j,i):\ \text{active},\ j \notin L, i \in L} c_{ji} \\ &\quad + \sum_{(j,i):\ \text{inactive or balanced},\ j \notin L, i \in L} b_{ji} \\ &\quad - \sum_{(i,j):\ \text{active or balanced},\ i \in L, j \notin L} c_{ij} \\ &\quad - \sum_{(i,j):\ \text{inactive},\ i \in L, j \notin L} b_{ij} \\ &\quad + \sum_{i \in L} s_i. \end{aligned} \tag{2.8}$$

Thus, the directional derivative $q'(p; d_L)$ is the difference between inflow and outflow across the node set $L$ when the flows of the inactive and active arcs are set at their lower and upper bounds, respectively, and the flow of each balanced arc incident to $L$ is set to its lower or upper bound depending on whether the arc is coming into $L$ or coming out of $L$. Equivalently, $q'(p; d_L)$ is the minimum of the total surplus of the nodes in $L$ over all flow vectors that satisfy CS together with $p$.

In principle, we could search over the set of vectors $d_L$ of the form (2.7) and find a direction of improvement, that is, one for which the directional derivative (2.8) is positive, but this brute force approach requires computation that is exponential in $|N|$ because there are exponentially many vectors of the form (2.7). A more effective alternative, which we now describe, is based on maintaining a flow vector $f$ satisfying CS together with $p$. This helps to organize the search for a direction of improvement. From the definition of the surplus, Eq. (1.7), we obtain

$$\sum_{i \in L} g_i = \sum_{\{(j,i) \in A \mid j \notin L, i \in L\}} f_{ji} - \sum_{\{(i,j) \in A \mid i \in L, j \notin L\}} f_{ij} + \sum_{i \in L} s_i. \tag{2.9}$$

Therefore, if $f$ satisfies CS together with $p$, we have using Eqs. (2.8) and (2.9)

$$\sum_{i\in L} g_i = q'(p; d_L) + \sum_{(j,i):\ \text{balanced},\ j\notin L, i\in L} (f_{ji} - b_{ji}) + \sum_{(i,j):\ \text{balanced},\ i\in L, j\notin L} (c_{ij} - f_{ij}) \geq q'(p; d_L). \tag{2.10}$$

We see, therefore, that only a node set $L$ that has positive total surplus is a candidate for generating a direction $d_L$ of dual cost improvement. The next lemma makes this idea precise and provides the basis for the subsequent algorithm.

**Lemma 2.1.** Suppose that $f$ and $p$ satisfy the CS conditions. Let $d_L$ be a direction vector of the form (2.7), and assume that

$$\sum_{i\in L} g_i > 0.$$

Then either $d_L$ is a dual ascent direction, that is,

$$q'(p; d_L) > 0,$$

or else there exist nodes $i \in L$ and $j \notin L$ such that either $(i, j)$ is a balanced arc and $f_{ij} < c_{ij}$, or $(j, i)$ is a balanced arc and $b_{ji} < f_{ji}$.

***Proof.*** Follows from Eq. (2.10). **Q.E.D.**

The typical iteration of the following algorithm starts with an integer price–flow vector pair satisfying the CS conditions, and operates in steps. At the beginning of each step, we have a connected subset of nodes $L$ such that

$$\sum_{i\in L} g_i > 0;$$

initially $L$ consists of an arbitrary node $i_1$ with positive surplus. According to the preceding lemma, there are two possibilities: Either (a) $d_L$ given by Eq. (2.7) is a direction of dual cost improvement, or (b) $L$ can be enlarged by adding a node $j \notin L$ with the property described in the lemma. In case (b), there are two possibilities: either (b1) $g_j \geq 0$, in which case,

$$\sum_{i\in L\cup\{j\}} g_i > 0,$$

and the process can be continued with

$$L \cup \{j\}$$

replacing $L$, or (b2) $g_j < 0$, in which case, it can be seen that there is a path originating at the starting node $i_1$ and ending at node $j$ with the property that all arcs on the path have room for a flow increase in the direction from $i_1$ to $j$. Such a path is called an *augmenting path* (see Fig. 5.2.6). By increasing the flow of the forward arcs (direction from $i_1$ to $j$) of the path and by decreasing the flow of the backward arcs (direction from $j$ to $i_1$) of the path, we can bring both surpluses $g_{i_1}$ and $g_j$ closer to zero by an integer amount while leaving the surplus of all other nodes unaffected and maintaining CS. Once $\sum_{i \in N} |g_i|$ is reduced to zero, the corresponding flow and price vectors are optimal by Prop. 5.1, so it follows that in a finite number of iterations, a direction of dual cost improvement will be found at any nonoptimal price vector.

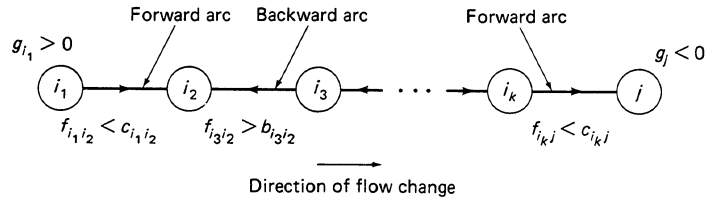



**Figure 5.2.6** Illustration of an augmenting path. The initial node $i_1$ and the final node $j$ have positive and negative surplus, respectively. Each arc on the path has room for flow change in the direction from $i_1$ to $j$. A flow change of magnitude $\delta > 0$ in this direction reduces the total absolute surplus $\sum_{i \in N} |g_i|$ by $2\delta$ provided $\delta \leq \min\{g_{i_1}, -g_j\}$.

We now formalize the preceding procedure. The algorithm starts with any integer pair $(f, p)$ satisfying CS. One possibility is to choose arbitrarily the integer vector $p$ and to set $f_{ij} = b_{ij}$ if $(i, j)$ is inactive or balanced, and $f_{ij} = c_{ij}$ otherwise. (Prior knowledge could be built into the initial choice of $p$ and $f$ based, for example, on the results of an earlier optimization.) It will be seen that the algorithm preserves the integrality and CS property of the pair $(f, p)$ throughout.

At the start of the typical iteration, we have an integer pair $(f, p)$ satisfying CS, and we select a node $i_1$ with positive surplus. The iteration indicates that the primal problem is infeasible, or else indicates that $(f, p)$ is optimal, or else transforms this pair into another pair $(f, p)$ satisfying CS. In the latter case the iteration terminates under two possible circumstances:

**(a)** When we find an augmenting path, in which case the flows of the arcs of the path are changed as illustrated in Fig. 5.2.6 (see Step 4 below). In this case the price vector (and hence also the dual cost) does not change, but the total absolute surplus $\sum_{i \in N} |g_i|$ is reduced by an integer amount.

**(b)** When we find a node set $L$ such that $d_L$ [cf. Eq. (2.7)] is a dual ascent direction. In this case the prices of the nodes in $L$ are increased by an integer amount and the dual cost is improved by an integer amount (see Step 5 below). The amount of price increase is the minimum required to either make some arc $(i, j)$ with $i \in L$, $j \notin L$ balanced from inactive or to make some arc $(j, i)$ with $j \notin L$, $i \in L$ balanced

from active. With a little thought, it is seen that the price increase corresponds to moving from the current price vector to the next breakpoint of the (piecewise linear) dual function along the direction $d_L$.

To implement the iteration efficiently, we introduce a simple data structure of *labels* (more sophisticated data structures are possible, but for simplicity we will not go into this). The label of a node $i$ is simply the ID number of another node (or "0" if $i$ is the starting node $i_1$ of the iteration), or else an indication that node $i$ has no label. In the former case, node $i$ is said to be *labeled*, whereas in the latter case, it is said to be *unlabeled*. At the start of the iteration all nodes are unlabeled except for the starting node $i_1$. Labels are helpful in constructing an augmenting path once a node $j$ with negative surplus is identified. The corresponding augmenting path $(j, i_k, i_{k-1}, \ldots, i_2, i_1)$ (cf. Fig. 5.2.6) is constructed backwards by taking $i_k$ to be the label of $j$, and for $m = k, \ldots, 2$, by taking $i_{m-1}$ to be the label of $i_m$, until the starting node $i_1$ is encountered (and recognized because of its distinctive label "0"). This construction is used in Step 4 below.

During an iteration, the current node set $L$ that is a candidate for yielding a dual ascent direction $d_L$ is maintained in a list. In the following algorithm, the nodes of $L$ are said to have been *scanned*, loosely indicating the fact that these nodes have been "selected" and "examined" during the iteration. The nodes which are not yet scanned, that is, do not belong to the current set $L$, are said to be *unscanned*. A node must be labeled before it can be scanned and the algorithm also maintains a list of the nodes that are labeled but unscanned. This list is used to keep track of the set of nodes that are candidates for entering the set $L$.

### Multiple Node Relaxation Iteration

**Step 1:** (*Initialization*) Choose a node $i_1$ with $g_{i_1} > 0$. (The iteration can be started also from a node $i_1$ with $g_{i_1} < 0$; the steps are similar.) If no such node can be found terminate the algorithm; the current pair $(f, p)$ is primal and dual optimal. Else give the label "0" to $i_1$, set $L$ :=empty, and go to Step 2.

**Step 2:** (*Choose a node to scan*) Select a labeled but unscanned node $i$, set $L := L \cup \{i\}$, and go to Step 3.

**Step 3:** (*Scan a node*) Scan the labeled node $i$ as follows: give the label "$i$" to all unlabeled nodes $j$ such that either $(j, i)$ is balanced and $f_{ji} > b_{ji}$ or $(i, j)$ is balanced and $f_{ij} < c_{ij}$. If

$$q'(p; d_L) > 0,$$

[cf. Eq. (2.8)] go to Step 5. Else if for any of the nodes $j$ given the label "$i$" earlier in this step, we have $g_j < 0$, go to Step 4. Else go to Step 2.

**Step 4:** (*Flow Augmentation*) An augmenting path $H$ has been found that begins at the starting node $i_1$ and ends at the node $j$ identified in Step 3. The path is constructed by tracing labels backwards starting from $j$, and is such that we have

$$f_{mn} < c_{mn}, \qquad \forall\ (m,n) \in H^+,$$
$$f_{mn} > b_{mn}, \qquad \forall\ (m,n) \in H^-,$$

where $H^+$ and $H^-$ are the sets of forward and backward arcs of $H$, respectively, given by

$$H^+ = \big\{(m,n) \in H \mid (m,n) \text{ is oriented in the direction from } i_1 \text{ to } j\big\},$$
$$H^- = \big\{(m,n) \in H \mid (m,n) \text{ is oriented in the direction from } j \text{ to } i_1\big\}.$$

Let

$$\delta = \min\Big\{g_{i_1}, -g_j, \big\{c_{mn}-f_{mn} \mid (m,n) \in H^+\big\}, \big\{f_{mn}-b_{mn} \mid (m,n) \in H^-\big\}\Big\}.$$

Increase by $\delta$ the flows of all arcs $(m,n) \in H^+$, decrease by $\delta$ the flows of all arcs $(m,n) \in H^-$, and go to the next iteration.

**Step 5:** (*Price Change*) Set

$$f_{ij} := c_{ij}, \qquad \forall \text{ balanced arcs } (i,j) \text{ with } i \in L, j \notin L, j \in M, \quad (2.11a)$$
$$f_{ji} := b_{ji}, \qquad \forall \text{ balanced arcs } (j,i) \text{ with } i \in L, j \notin L, j \in M, \quad (2.11b)$$

where $L$ is the set of scanned nodes constructed in Step 2, and $M$ is the set of currently labeled nodes. Let

$$\gamma = \min_{\xi \in S^+ \cup S^-} \xi, \tag{2.12}$$

where

$$S^+ = \big\{p_j + a_{ij} - p_i \mid (i,j)\text{: inactive}, i \in L, j \notin L\big\},$$
$$S^- = \big\{p_j - a_{ji} - p_i \mid (j,i)\text{: active}, i \in L, j \notin L\big\}.$$

Set

$$p_i := \begin{cases} p_i + \gamma, & \text{if } i \in L, \\ 0, & \text{otherwise.} \end{cases} \tag{2.13}$$

Go to the next iteration. (*Note:* If there is no active arc $(i,j)$ with $i \in L$, $j \notin L$, or inactive arc $(j,i)$ with $i \in L$, $j \notin L$, then the price increment $\gamma$ can be taken infinite, which indicates that the dual optimal value is infinite or, equivalently, that the primal problem is infeasible; see Exercise 2.1).

It is clear that all operations of the algorithm preserve the integrality of the price–flow vector pair. To see that CS is also maintained, note that a flow augmentation step

changes only flows of balanced arcs and, therefore, cannot destroy CS; the flow change of Eq. (2.11) and the price change of Eq. (2.13) maintain CS because they set the flows of the balanced arcs that the price change renders active (or inactive) to the corresponding upper (or lower) bounds. Assuming the optimal dual value is finite (otherwise the primal problem is infeasible), termination of the algorithm is guaranteed by the fact that each price change improves the dual cost by an integer amount. Furthermore, it is impossible to have an infinite number of flow augmentation steps, since each of these reduces the total absolute surplus by an integer amount. Finally, termination can occur only when all nodes have zero surplus, which together with CS guarantees optimality of the final price–flow vector pair. Since the algorithm maintains integrality of flows and prices, the optimal flow and price vector obtained upon termination will be integer. We have thus shown constructively that if problem (LNF) is feasible, there exist optimal *integer* flow and price vectors. If the problem is infeasible, this can be detected either in Step 5 (see the note given there) or through the fact that some prices increase to infinity (see also Exercise 2.1).

Figure 5.2.7 traces the steps of the algorithm for a simple shortest path example. The figure illustrates that in an efficient implementation of the algorithm, more than one iterations can be "combined" in a single iteration in order to save computation time.

The stepsize $\gamma$ of Eq. (2.12) corresponds to the first break point of the dual function along the ascent direction $d_L$. It is also possible to calculate through a line search an optimal stepsize that maximizes the dual function along $d_L$. We leave it for the reader to verify that this computation can be done quite economically, using Eq. (2.8) or Eq. (2.10) to test the sign of the directional derivative of the dual function at successive break points along $d_L$. Computational experience shows that a line search is beneficial in practice. Consider now the case where there is a price change via Step 5 and the set $L$ consists of just the starting node $i_1$. This happens when the multiple node iteration scans $i_1$ and finds (at the first time Step 3 is entered) that the corresponding coordinate direction leads to a dual cost improvement $[q'(p; d_{\{i_1\}}) > 0]$. If line search of the type just described is performed, the price $p_{i_1}$ is changed to a break point where $g_{i_1}^+ \leq 0 \leq g_{i_1}^-$. In this case the multinode iteration becomes identical with the (single node) relaxation iteration examined earlier in this section (cf. Fig. 5.2.1). Computational experience shows that such single node iterations are very frequent in the early stages of the algorithm and account for most of the total dual cost improvement, but become much less frequent near the algorithm's termination.

In practice, the method should be implemented using iterations that start from both positive and negative surplus nodes. This seems to improve substantially the performance of the method. It can be shown that the algorithm terminates properly under these circumstances (Exercise 2.2). Another important practical issue has to do with the initial choice of flows and prices. One possibility is to try to choose an initial price vector that is as close to optimal as possible (for example, using the results of some earlier optimization); one can then choose the arc flows to satisfy the CS conditions.

Computational experiments show that the method based on multiple node relaxation iterations has excellent practical performance. Its computational complexity, however,

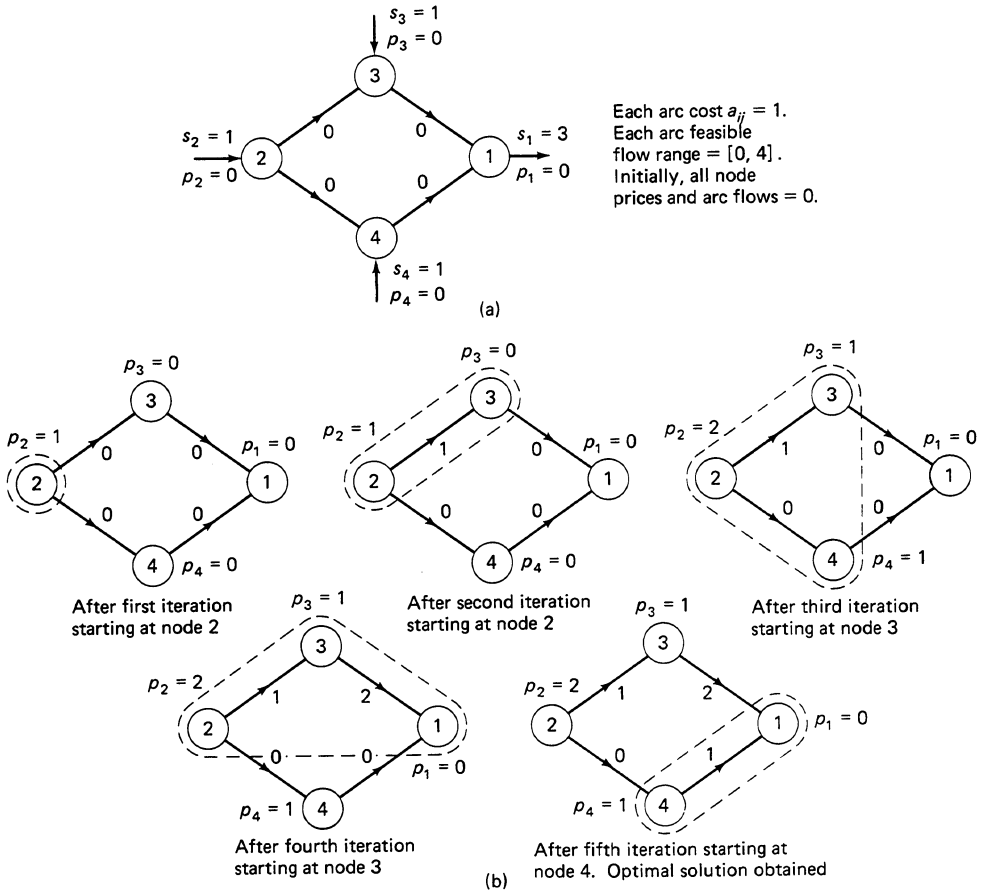



**Figure 5.2.7** Example of a solution of a shortest path problem (cf. Fig. 5.1.3) using the multiple node relaxation method. The numbers next to the arcs are the arc flows. All problem data and initial conditions are as shown in part (a). The problem is solved in five iterations as shown in part (b). [The iteration sequence generated by the algorithm depends on the rule for choosing the starting node of an iteration; the sequence shown in part (b) is one possible iteration sequence.] The broken curves indicate the set of labeled nodes in each iteration. A description of the iterations follows:

**(1)** Iteration 1 starts at node 2 and raises the price of node 2 by one unit through Step 5.

**(2)** Iteration 2 starts at node 2, identifies the augmenting path $2 \rightarrow 3$, and increases the flow of arc $(2, 3)$ by one unit through Step 4.

**(3)** Iteration 3 starts at node 3, labels node 2, and then node 4. It then raises the prices of nodes 2, 3, and 4 by one unit through Step 5.

**(4)** Iteration 4 starts at node 3, identifies the augmenting path $3 \rightarrow 1$, and increases the flow of arc $(3, 1)$ by two units through Step 4.

**(5)** Iteration 5 starts at node 4, identifies the augmenting path $4 \rightarrow 1$, and increases the flow of arc $(4, 1)$ by one unit through Step 4.

A more efficient implementation of the algorithm tries to identify an augmenting path following a price change. Then iterations 1 and 2 can be combined in a single iteration. Iterations 3 and 4 can be similarly combined.

depends on the size of the arc cost coefficients. To see this, consider the shortest path problem with positive arc lengths. The Bellman–Ford algorithm starting from the zero initial conditions is a special case of the relaxation method, as discussed earlier, and in Section 4.1 we saw that its complexity depends on the maximum arc length (cf. Fig. 4.1.2 in Chapter 4). It is possible to improve the theoretical complexity of the method by using the technique of scaling the arc cost coefficients, which is discussed in Section 5.4 (see Exercise 4.2).

We finally note that multiple node relaxation iterations are less suited for parallelization than the earlier single node iterations. The difficulty is that while multiple node iterations can start simultaneously from two different nodes with positive surplus, it is possible that the corresponding sets of labeled nodes may intersect at some point, in which case, only one of the two iterations can proceed. This has an adverse effect on the degree of parallelism afforded by the algorithm. It is still possible, however, to construct a parallel implementation of the method by using an arbitration mechanism that allows only one out of two or more simultaneous and interfering multiple node iterations to proceed while the others are temporarily suspended. The details of this are somewhat complicated and will not be discussed.

## EXERCISES

**2.1.** **[Ber86a]** This exercise illustrates a type of argument that is central in the complexity analysis of relaxation algorithms for linear network flow problems. Consider the multiple node relaxation algorithm, let $p_i^0$ be the initial price of node $i$, and let $S$ be the set of nodes that have negative surplus initially. For every simple path $p$ that ends at a node $j \in S$, let $L_p$ be the sum of the costs of the forward arcs of the path minus the sum of the costs of the backward arcs of the path, and let $L = \max_p L_p$. Assume that only nodes with positive surplus are chosen as starting nodes in the relaxation iteration. Show that, if the problem is feasible, then during the course of the algorithm, the price of any positive surplus node cannot exceed its initial price by more than $L + \max_{j \in S} p_j^0 - \min_{i \in N} p_i^0$. *Hint:* Use the fact that at any point in the algorithm the prices of all nodes with negative surplus have not changed since the start of the algorithm. Show also that if $i$ is a node with positive surplus, there must exist some node with negative surplus $j$ and a path starting at $i$ and ending at $j$ such that all forward arcs of the path are inactive or balanced, and all backward arcs of the path are active or balanced.

**2.2.** Write the form of the multiple node relaxation iteration starting from *both* positive and negative surplus nodes. Show that the method terminates at an optimal flow–price vector pair if a feasible solution exists.

**2.3.** **(The Primal–Dual Method [FoF62].)** The purpose of this exercise is to clarify the relation of the multiple node relaxation method and two versions of a classical ascent method for solving the dual problem, which is commonly referred to as the *primal–dual* method.

**(a)** One version of the primal–dual method for (LNF) can be obtained through a small (but significant) change in the description of the multiple node iteration. Simply replace the statement "If $q'(p; d_L) > 0$ [cf. Eq. (2.8)] go to Step 5" of Step 3 by the

statement "If the set of scanned nodes is equal to the set of labeled nodes go to Step 5". Show that the resulting method terminates in a finite number of iterations.

**(b)** Another version of the primal–dual method is obtained by making the change described in part (a), and also by giving in Step 1 the label "0" to all nodes $i$ with positive surplus instead of just to a single node with positive surplus. Show that the resulting method also terminates in a finite number of iterations, and that the direction of ascent used in Step 5 maximizes the directional derivative $q'(p; d_L)$ over all vectors $d_L$ of the form (2.7).

## 5.3 THE $\epsilon$ – RELAXATION METHOD

In the preceding section, we discussed one possible method for resolving the difficulty due to the nondifferentiability of the dual cost illustrated in Fig. 5.2.2. In this section, we consider an alternative method. The main idea is illustrated in Fig. 5.3.1. We allow single node price changes even if these worsen the dual cost. The rationale is that if the cost deterioration is small, then the algorithm can approach eventually the optimal solution. Indeed, we will show that this is so, and in fact an *exact* solution of the problem can be obtained in a finite number of iterations owing to the integer nature of the problem data. A key idea is that each price change improves the dual cost of a perturbed problem, where some of the arc cost coefficients are modified by a small amount $\epsilon$. Implementation of this idea is based on a notion of approximate complementary slackness, which we now introduce.

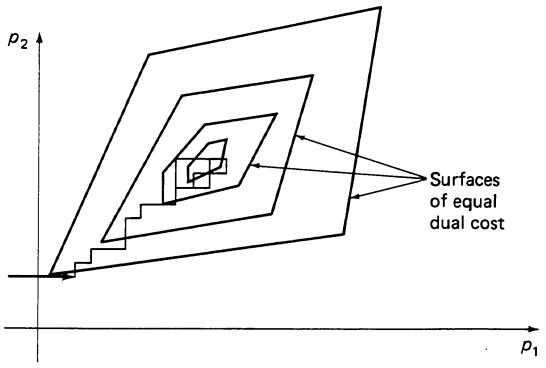



**Figure 5.3.1** Illustration of the idea of the $\epsilon$–relaxation method. By making small changes in the coordinate directions, it is possible to approach the optimal solution even if each step does not result in a dual cost improvement. The method eventually reaches a small neighborhood of the optimal solution.

For any price vector $p$ and $\epsilon > 0$, we say that an arc $(i, j)$ is

$$\epsilon - \text{Inactive} \qquad \text{if } p_i < a_{ij} + p_j - \epsilon, \tag{3.1a}$$

$$\epsilon^- - \text{Balanced} \qquad \text{if } p_i = a_{ij} + p_j - \epsilon, \tag{3.1b}$$

$$\epsilon - \text{Balanced} \qquad \text{if } a_{ij} + p_j - \epsilon \leq p_i \leq a_{ij} + p_j + \epsilon, \tag{3.1c}$$

$$\epsilon^+ - \text{Balanced} \qquad \text{if } p_i = a_{ij} + p_j + \epsilon, \tag{3.1d}$$

$$\epsilon - \text{Active} \qquad \text{if } p_i > a_{ij} + p_j + \epsilon. \tag{3.1e}$$

Given $\epsilon \geq 0$, we say that a vector pair $(f, p)$ satisfies *$\epsilon$–complementary slackness* ($\epsilon$–CS) if for each arc $(i, j)$,

$$f_{ij} = b_{ij} \qquad \text{if } (i,j) \text{ is } \epsilon\text{–inactive,} \tag{3.2a}$$

$$b_{ij} \leq f_{ij} \leq c_{ij} \qquad \text{if } (i,j) \text{ is } \epsilon\text{–balanced,} \tag{3.2b}$$

$$f_{ij} = c_{ij} \qquad \text{if } (i,j) \text{ is } \epsilon\text{–active.} \tag{3.2c}$$

An equivalent statement of the $\epsilon$–CS conditions is that the flows $f_{ij}$ satisfy the capacity constraints (1.2), and that

$$f_{ij} < c_{ij} \quad \Rightarrow \quad p_i - p_j \leq a_{ij} + \epsilon, \tag{3.3a}$$

$$b_{ij} < f_{ij} \quad \Rightarrow \quad p_i - p_j \geq a_{ij} - \epsilon \tag{3.3b}$$

(see Fig. 5.3.2). A useful way to think about $\epsilon$–CS is that if the pair $(f, p)$ satisfies it, then the primal cost to be obtained by moving flow around a cycle $Y$ without violating the capacity constraints decreases at a rate that is at most $|Y|\epsilon$, where $|Y|$ is the number of arcs of $Y$. This fact is the essence of the proof of Prop. 3.1 that follows.

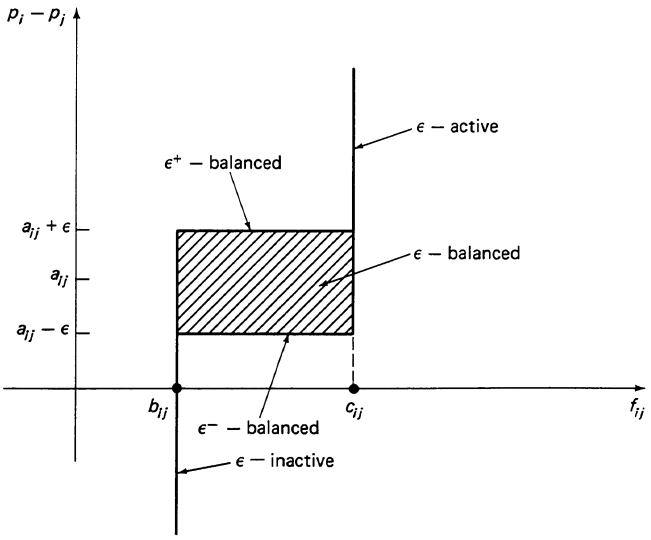



**Figure 5.3.2** Illustration of $\epsilon$–CS. All pairs of arc flows $f_{ij}$ and price differentials $p_i - p_j$ should either lie on the thick–line diagram or in the shaded area between the thick lines.

The algorithm to be described shortly maintains at all times a price vector $p$ and a flow vector $f$ satisfying $\epsilon$–CS. It terminates when the flow vector $f$ satisfies the primal feasibility condition $g_i = 0$ for all $i \in N$. A key fact is that if $\epsilon$ is sufficiently small, then the final flow vector $f$ is optimal. The proof given uses our earlier assumption that the arc cost coefficients $a_{ij}$ are integer, but does not use the assumption that the flow bounds $b_{ij}$ and $c_{ij}$ are also integer.

**Proposition 3.1.** If $\epsilon < 1/|N|$ and the flow vector $f$ together with the price vector $p$ satisfy $\epsilon$–CS and primal feasibility ($g_i = 0$ for all $i \in N$), then $f$ is optimal for (LNF).

***Proof.*** If $f$ is not optimal, there must exist a nonzero flow vector $y = \{y_{ij} \mid (i,j) \in A\}$ such that $f + y$ is primal feasible, and has lower cost than $f$, that is, $y$ is a circulation and

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

$$\sum_{(i,j)\in A} a_{ij} y_{ij} < 0.$$

By the Conformal Realization Theorem (Appendix B) the circulation $y$ can be decomposed into the sum of a finite number of simple circulations $y^1, \ldots, y^m$ that conform to $y$ ($y^k_{ij} > 0$ or $y^k_{ij} < 0$ implies $y_{ij} > 0$ or $y_{ij} < 0$ respectively). Therefore $f + y^k$ is primal feasible for all $k = 1, \ldots, m$, and at least one circulation $y^k$ defines a direction of descent, that is,

$$\sum_{(i,j)\in A} a_{ij} y^k_{ij} < 0.$$

Let $Y^+$ and $Y^-$ be the sets of arcs $(i,j)$ of the cycle corresponding to $y^k$ for which $y^k_{ij} > 0$ and $y^k_{ij} < 0$, respectively. Since $y^k$ is a simple circulation, we have that $|y^k_{ij}|$ is equal to some $\delta > 0$ on all arcs $(i,j)$ for which $y^k_{ij} \neq 0$. Therefore, by dividing the preceding condition by $\delta$ we obtain

$$\sum_{(i,j)\in Y^+} a_{ij} - \sum_{(i,j)\in Y^-} a_{ij} < 0, \tag{3.4}$$

$$f_{ij} < c_{ij}, \qquad \forall\, (i,j) \in Y^+,$$

$$b_{ij} < f_{ij}, \qquad \forall\, (i,j) \in Y^-.$$

By $\epsilon$–CS [cf. Eq. (3.3)], we have

$$p_i \leq p_j + a_{ij} + \epsilon, \qquad \forall\, (i,j) \in Y^+,$$

$$p_j \leq p_i - a_{ij} + \epsilon, \qquad \forall\, (i,j) \in Y^-,$$

which, by adding and using the hypothesis $\epsilon < 1/|N|$, yields

$$\sum_{(i,j)\in Y^+} a_{ij} - \sum_{(i,j)\in Y^-} a_{ij} \geq -|N|\epsilon > -1.$$

Since the $a_{ij}$ are integer, we obtain a contradiction of Eq. (3.4). **Q.E.D.**

A strengthened form of Prop. 3.1 which remains valid even if the arc cost coefficients and flow bounds are not integer, is obtained by replacing the condition $\epsilon < 1/|N|$ with the condition

$$\epsilon < \min_{\text{All cycles } Y} \left\{ -\frac{\text{Length of cycle } Y}{\text{Number of arcs of } Y} \mid \text{Length of } Y < 0 \right\},$$

where

$$\text{Length of cycle } Y = \sum_{(i,j)\in Y^+} a_{ij} - \sum_{(i,j)\in Y^-} a_{ij}.$$

The proof is obtained by suitably modifying the last relation in the proof of Prop. 3.1.

The *$\epsilon$–relaxation method* uses a fixed value of $\epsilon > 0$, and starts with a pair $(f, p)$ such that $\epsilon$–CS is satisfied and $f_{ij}$ are all integer. The algorithm preserves the $\epsilon$–CS and flow integrality properties throughout. A possible starting procedure is to arbitrarily choose the vector $p$, and to set $f_{ij} = b_{ij}$ if $(i, j)$ is inactive or balanced, and $f_{ij} = c_{ij}$ otherwise. At the start of each iteration, a node $i$ with positive surplus $g_i$ is chosen. (If all nodes have zero surplus the algorithm terminates; then $f$ is primal feasible and, together with $p$, satisfies $\epsilon$–CS, so Prop. 3.1 applies.) At the end of the iteration, the surplus $g_i$ is driven to zero, while another pair $(f, p)$ satisfying $\epsilon$–CS is obtained. During an iteration, all node prices stay unchanged except possibly for the price of the chosen node $i$. Similarly, all arc flows stay unchanged except for the flows of some of the arcs incident to node $i$. As a result of these flow changes, the surplus of some of the nodes adjacent to $i$ is increased. In order for the flow of an arc $(i, j)$ to change, the arc must be $\epsilon^+$–balanced and $f_{ij} < c_{ij}$ (such an arc is called *$\epsilon^+$–unblocked*). Similarly, in order for the flow of an arc $(j, i)$ to change, the arc must be $\epsilon^-$–balanced and $b_{ji} < f_{ji}$ (such an arc is called *$\epsilon^-$–unblocked*). The price of node $i$ at the end of the iteration is usually $\bar{p}_i + \epsilon$, where $\bar{p}_i$ is one of the maximizing points of $Q_i^p(\xi)$, the dual function along the $i$th price coordinate. This price level is reached through possibly several price increases at Step 4 below (see the subsequent discussion and Exercise 3.1). Each price increase may be preceded and followed by flow changes of incident arcs of node $i$ to maintain $\epsilon$–CS and to reduce the surplus $g_i$ to zero (Steps 2 and 3 below).

### Positive Surplus Node Iteration (or Up Iteration):

Let $(f, p)$ satisfy $\epsilon$–CS, and let $i$ be a node with $g_i > 0$.

**Step 1:** (*Scan incident arc*) Select a node $j$ such that $(i, j)$ is an $\epsilon^+$–unblocked arc and go to Step 2, or select a node $j$ such that $(j, i)$ is an $\epsilon^-$–unblocked arc and go to Step 3. If no such node can be found go to Step 4.

**Step 2:** (*Decrease surplus by increasing $f_{ij}$*) Let $\delta = \min\{g_i, c_{ij} - f_{ij}\}$. Update $f_{ij}$, $g_i$, and $g_j$ according to

$$f_{ij} := f_{ij} + \delta,$$
$$g_i := g_i - \delta, \qquad g_j := g_j + \delta.$$

If the updated values $g_i$ and $f_{ij}$ satisfy $g_i = 0$ and $f_{ij} < c_{ij}$, stop; else go to Step 1.

**Step 3:** (*Decrease surplus by reducing* $f_{ji}$) Let $\delta = \min\{g_i, f_{ji} - b_{ji}\}$. Update $f_{ji}$, $g_i$, and $g_j$ according to

$$f_{ji} := f_{ji} - \delta,$$
$$g_i := g_i - \delta, \qquad g_j := g_j + \delta.$$

If the updated values $g_i$ and $f_{ji}$ satisfy $g_i = 0$ and $b_{ji} < f_{ji}$, stop; else go to Step 1.

**Step 4:** (*Increase price of node* $i$) Set

$$p_i := \min_{\xi \in R^+ \cup R^-} \xi, \tag{3.5}$$

where

$$R_i^+ = \{p_j + a_{ij} + \epsilon \mid (i,j) \in A \text{ and } f_{ij} < c_{ij}\},$$
$$R_i^- = \{p_j - a_{ji} + \epsilon \mid (j,i) \in A \text{ and } b_{ji} < f_{ji}\}.$$

Go to Step 1. (Note: If $g_i > 0$ and the set $R_i^+ \cup R_i^-$ over which the minimum in Eq. (3.5) is taken is empty, the problem is infeasible and the algorithm terminates; see the comments that follow. If this set is empty and $g_i = 0$, we leave $p_i$ unchanged and stop.)

To see that Eq. (3.5) leads to a price increase, note that when Step 4 is entered, we have $f_{ij} = c_{ij}$ for all $(i,j)$ such that $p_i \geq p_j + a_{ij} + \epsilon$, and we have $b_{ji} = f_{ji}$ for all $(j,i)$ such that $p_i \geq p_j - a_{ji} + \epsilon$. Therefore, when Step 4 is entered, we have

$$p_i < \min R_i^+ = \min\{p_j + a_{ij} + \epsilon \mid (i,j) \in A \text{ and } f_{ij} < c_{ij}\},$$
$$p_i < \min R_i^- = \min\{p_j - a_{ji} + \epsilon \mid (j,i) \in A \text{ and } b_{ji} < f_{ji}\}.$$

It follows that $p_i$ must be increased via Eq. (3.5) when the set $R_i^+ \cup R_i^-$ over which the minimum is taken is nonempty. In the case where this set is empty, we have $f_{ij} = c_{ij}$ for all $(i,j)$ outgoing from $i$ and $b_{ji} = f_{ji}$ for all $(j,i)$ incoming to $i$, so maximum flow is going out of $i$ while minimal flow is coming in. Therefore, if $g_i > 0$ and $R_i^+ \cup R_i^-$ is empty, we can terminate the algorithm with the assurance that the problem is infeasible.

Figure 5.3.3 illustrates an up iteration. It is seen that each time Step 2 or 3 is executed, flow is pushed away from $i$ along an $\epsilon^+$–unblocked or an $\epsilon^-$–unblocked arc,

respectively. If no more flow can be pushed and $g_i > 0$, the price of $i$ is increased in Step 4.

Figures 5.3.3 and 5.3.4 illustrate the sequence of price changes of an up iteration in the cases where the dual cost has one and multiple maximizing points, respectively, with respect to $p_i$. It is seen in these figures that at the end of the iteration, the price of the node $i$ equals $\epsilon$ plus some value that maximizes the dual cost with respect to $p_i$ with all other prices kept fixed (this property can be shown for the case where $p_i + \epsilon$ is less than the minimal maximizing point of the dual cost; see Exercise 3.1). We thus obtain an interpretation of the algorithm as a relaxation (or coordinate ascent) method, although "approximate relaxation" may be a more appropriate term.

Consider now the case where there is a bounded interval $[\underline{p}_i, \bar{p}_i]$ of maximizing points with $\underline{p}_i < \bar{p}_i$ [see Fig. 5.3.4(a)]. A careful examination of the steps of the algorithm shows that it has a tendency to set the price $p_i$ close to the largest maximizing point $\bar{p}_i$. This is due to the fact that the iteration does not stop when $g_i = 0$ and $f_{ij} = c_{ij}$ (in Step 2) or $f_{ji} = b_{ji}$ (in Step 3). As a result Step 4 may be entered with $g_i = 0$ with an additional price increase resulting over the version of the iteration that always stops when $g_i = 0$ in Step 2 or 3. The latter version tends to set the price $p_i$ near the smallest maximizing point $\underline{p}_i$ and seems to work worse in practice. The reasons for this are not entirely clear, but the complexity analysis of the next section provides some justification since it suggests that the algorithm terminates faster when the price changes are as large as possible.

Note that a symmetric iteration can be used for nodes with negative surplus (called *down iteration*). One can construct an example (see Exercise 3.2) showing that the algorithm may not terminate if up and down iterations are mixed arbitrarily. It is therefore necessary to impose some assumptions either on the problem structure or on the method by which up and down iterations are interleaved. We henceforth assume that the algorithm consists of up iterations only.

**Proposition 3.2.** If problem (LNF) is feasible, the algorithm terminates with $(f, p)$ satisfying $\epsilon$–CS, and with $f$ being integer and primal feasible.

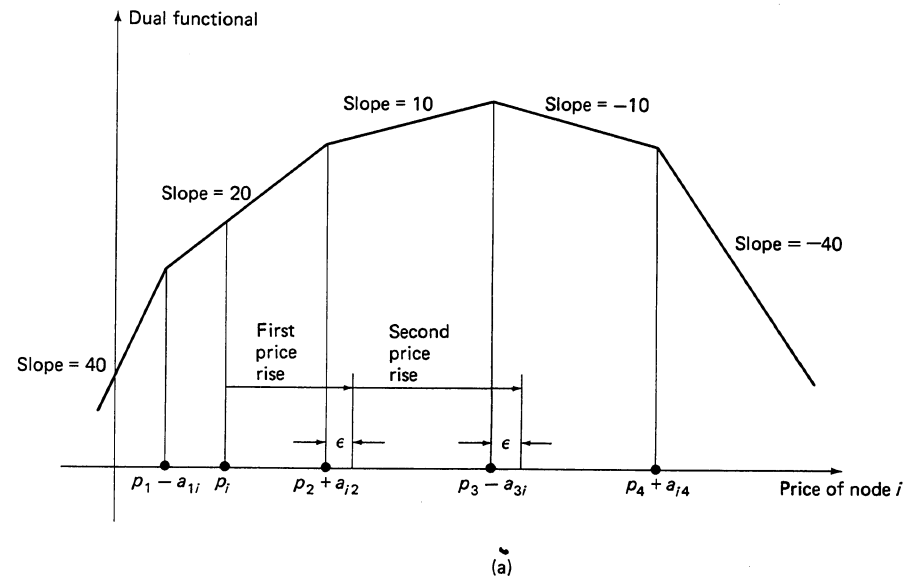
***Proof.*** The following facts can be verified based on the construction of the up iteration:

**(1)** The integrality of $f$ and the $\epsilon$–CS property of $(f, p)$ are preserved throughout the algorithm.

**(2)** The prices of all nodes are monotonically nondecreasing during the algorithm.

**(3)** Once a node has nonnegative surplus, its surplus stays nonnegative thereafter. (This follows from the fact that an up iteration at some node $i$ cannot drive the surplus of $i$ below zero, and can only increase the surplus of its adjacent nodes.)

**(4)** If at some time a node has negative surplus, it must have never been iterated on up to that time, and therefore its price must be equal to its initial price. [This is a consequence of (3) above and the fact that only nodes with positive surplus are iterated on by up iterations.]



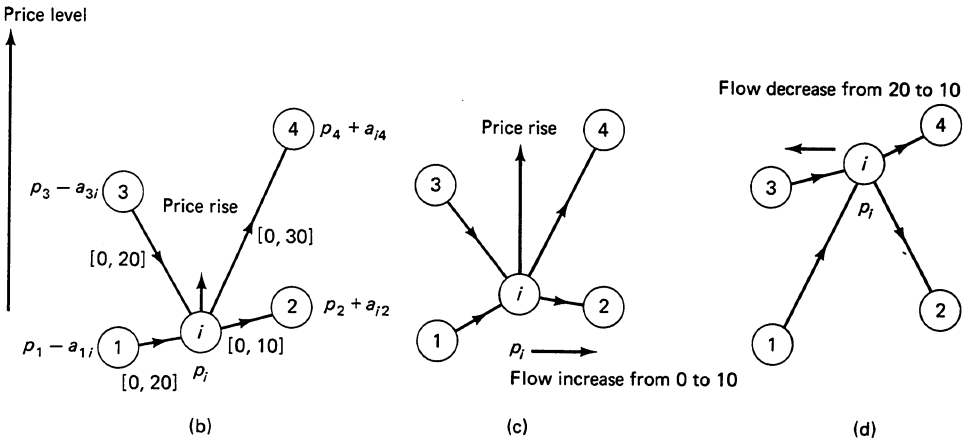




**Figure 5.3.3** Illustration of an up iteration involving a single node $i$ and the arcs $(1,i)$, $(3,i)$, $(i,2)$, and $(i,4)$ with feasible arc flow ranges [0,20], [0,20], [0,10], and [0,30], respectively, and $s_i = 0$. (a) Form of the dual functional along $p_i$ for given values of $p_1$, $p_2$, $p_3$, and $p_4$. The breakpoints correspond to the levels of $p_i$ for which the corresponding arcs become balanced. For values of $p_i$ between two successive breakpoints, there are no balanced arcs incident to node $i$. The corresponding slope of the dual cost is equal to the surplus $g_i$ resulting when all active arc flows are set to their upper bounds and all inactive arc flows are set to their lower bounds; compare with Eq. (2.1). (b) Illustration of a price rise of $p_i$ from a value between the first two breakpoints to a value $\epsilon$ above the breakpoint at which $(i,2)$ becomes balanced (Step 4). (c) Price rise of $p_i$ to a value $\epsilon$ above the breakpoint at which arc $(3,i)$ becomes balanced. When this is done, arc $(i,2)$ has changed from $\epsilon^+$–balanced to $\epsilon$–active, and its flow has increased from 0 to 10, maintaining $\epsilon$–CS. (d) Step 3 of the algorithm reduces the flow of arc $(3,i)$ from 20 to 10, driving the surplus of node $i$ to zero.

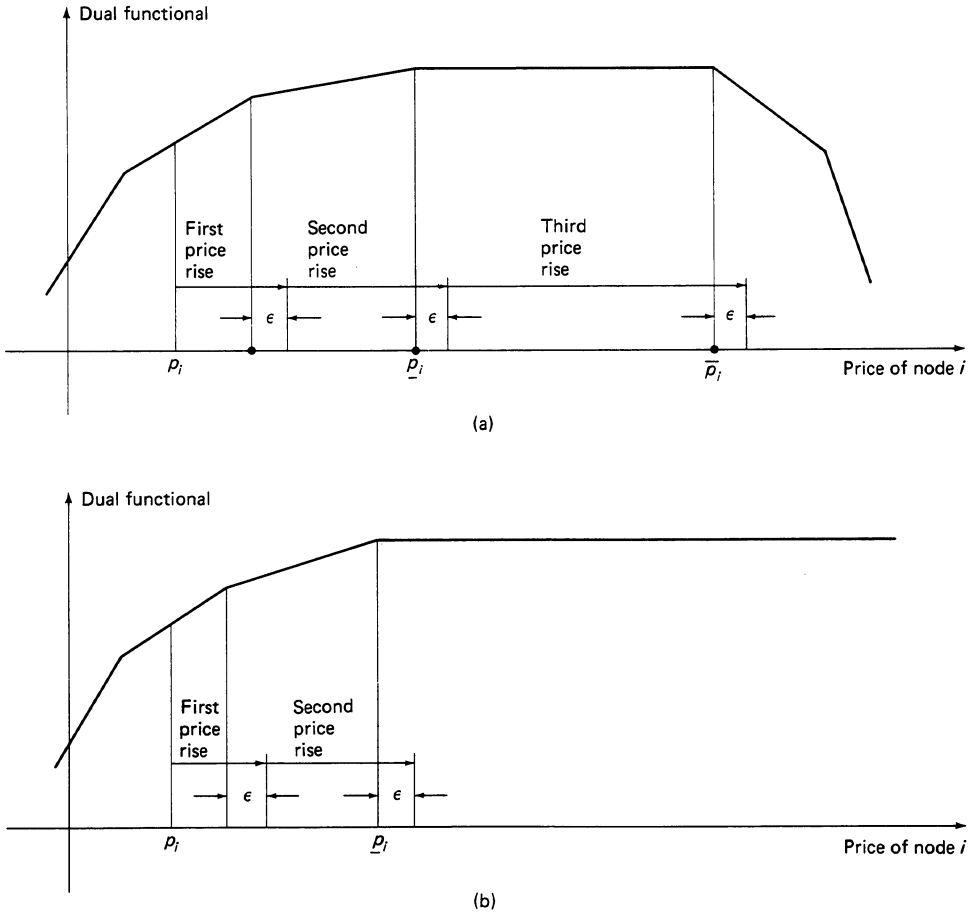



**Figure 5.3.4** Illustration of an up iteration in the case where there are multiple maximizing points of the dual cost with respect to $p_i$. In case (a), the set of maximizing points is bounded, and at the end of the iteration, $p_i$ is set at $\epsilon$ plus the largest maximizing point $\bar{p}_i$. In case (b), the set of maximizing points is unbounded, and at the end of the iteration, $p_i$ is set at $\epsilon$ plus the smallest maximizing point $\underline{p}_i$.

Based on (2) there are two possibilities: either (a) the prices of a nonempty subset $N^\infty$ of $N$ diverge to $+\infty$ or else (b) the prices of all nodes in $N$ stay bounded from above.

Suppose that case (a) holds. Then, since $N^\infty$ is nonempty, it follows that the algorithm never terminates, implying that at all times there must exist a node with negative surplus which, by (4), must have a constant price. It follows that $N^\infty$ is a proper subset of $N$. To preserve $\epsilon$–CS, we must have, after a sufficient number of iterations,

$$f_{ij} = c_{ij} \qquad \text{for all } (i,j) \in A \text{ with } i \in N^\infty, j \notin N^\infty,$$
$$f_{ji} = b_{ij} \qquad \text{for all } (j,i) \in A \text{ with } i \in N^\infty, j \notin N^\infty,$$

while the sum of surpluses of the nodes in $N^\infty$ must be positive. This means that even with as much flow as arc capacities allow coming out of $N^\infty$ to nodes $j \notin N^\infty$ and as little flow as arc capacities allow coming into $N^\infty$ from nodes $j \notin N^\infty$, the total surplus of nodes in $N^\infty$ is positive. It follows that there is no feasible flow vector contradicting the hypothesis. Therefore, case (b) holds (all prices of nodes in $N$ stay bounded).

We now show by contradiction that the algorithm terminates. If that is not so, then there must exist a node $i \in N$ at which an infinite number of iterations are executed. There must also exist an adjacent $\epsilon^-$–balanced arc $(j,i)$ or $\epsilon^+$–balanced arc $(i,j)$ whose flow is decreased or increased, respectively, by an integer amount during an infinite number of iterations. For this to happen, the flow of $(j,i)$ or $(i,j)$ must be increased or decreased, respectively, an infinite number of times due to iterations at the adjacent node $j$. This implies that the arc $(j,i)$ or $(i,j)$ must become $\epsilon^+$–balanced or $\epsilon^-$–balanced from $\epsilon^-$–balanced or $\epsilon^+$–balanced, respectively, an infinite number of times. For this to happen, the price of the adjacent node $j$ must be increased an infinite number of times by at least $2\epsilon$. It follows that $p_j \to \infty$, which contradicts the boundedness of all node prices shown earlier. Therefore the algorithm must terminate. **Q.E.D.**

Note that Prop. 3.2 holds for all $\epsilon > 0$. If $\epsilon < 1/|N|$, however, we see, by combining Props. 3.1 and 3.2, that the algorithm terminates with an *optimal* flow vector. Note also that the integrality of $a_{ij}$ was not needed for the proof of Prop. 3.2, while the integrality of $b_{ij}$, $c_{ij}$, $s_i$, and the starting flow vector were only needed to establish that the flow change increments are bounded from below during the course of the algorithm. The integrality assumptions are essential, however, for the complexity analysis of the next section.

Proposition 3.2 applies without modification to the variation of the algorithm, where up iterations are not necessarily carried to completion. In this variation, it is permissible to execute only *partial* up iterations, in which the algorithm can select a new node for iteration immediately following the completion of any Step 2, 3, or 4, even if the current node surplus is not yet zero.

While it is possible for a price change in Step 4 of the up iteration to degrade the dual cost, there is still an interesting interpretation of Step 4 as a dual cost improvement. It can be seen from Eq. (2.1) that the sign of the directional derivatives $g_i^+$ and $g_i^-$ can change if the cost coefficients of some of the $\epsilon$–balanced arcs incident to node $i$ are perturbed by a small $\epsilon$ amount. It follows that price changes in Step 4 yield a dual cost improvement of a *perturbed* problem where some of the arc cost coefficients are slightly changed.

A final issue has to do with detection of infeasibility (assuming it is not detected at Step 4 of some iteration). By using the argument of the proof of Prop. 3.2, it follows that for an infeasible problem, the prices of some nodes diverge to $+\infty$. In Section 5.4,

we derive a precomputable upper bound for the prices when the problem is feasible [Eq. (4.7)]. Once this bound is exceeded, we know that the problem is infeasible.

### 5.3.1 The Auction Algorithm for the Assignment Problem

The main idea of the $\epsilon$–relaxation iteration is to increase a single price coordinate so as to approximately optimize a dual cost with respect to that coordinate. This process can be applied to several other network duality formulations. As an illustration, we consider an alternative duality formulation of the assignment problem that leads to an effective computational method.

The method, called the *auction algorithm*, can be intuitively understood in terms of an economic process, and will be consequently described in those terms. Sources and sinks are viewed as persons and objects, respectively. The algorithm operates like an auction, whereby unassigned persons bid simultaneously for objects, thereby raising their prices. Once all bids are in, objects are awarded to the highest bidder. The serial or Gauss–Seidel version of the algorithm can be interpreted as a variation of the $\epsilon$–relaxation method (see Exercise 3.5 for the precise relation).

Consider $n$ persons wishing to divide among themselves $n$ objects. For each person $i$, there is a nonempty subset $A(i)$ of objects that can be assigned to $i$, and there is a given integer value $a_{ij}$ that person $i$ associates with each object $j$. An *assignment* $S$ is a (possibly empty) set of person–object pairs $(i,j)$ such that $j \in A(i)$ for all $(i,j) \in S$; for each person $i$, there is at most one pair $(i,j) \in S$; and for each object $j$, there is at most one pair $(i,j) \in S$. In the context of a given assignment $S$, we say that person $i$ is *assigned* if there exists an object $j$ such that $(i,j) \in S$; otherwise, we say that $i$ is *unassigned*. We use similar terminology for objects. A *complete assignment* is an assignment containing $n$ pairs (i.e., every person is assigned to a distinct object). We want to find a complete assignment that maximizes

$$\sum_{(i,j)\in S} a_{ij}$$

over all complete assignments $S$. This problem is equivalent to the linear programming problem

$$\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{n} \sum_{j\in A(i)} a_{ij} f_{ij} \\
\text{subject to} \quad & \sum_{j\in A(i)} f_{ij} = 1, && \forall\, i = 1,\ldots,n, \\
& \sum_{\{i \mid j\in A(i)\}}^{n} f_{ij} = 1, && \forall\, j = 1,\ldots,n, \\
& 0 \le f_{ij}, && \forall\, i = 1,\ldots,n, \quad j \in A(i).
\end{aligned} \tag{3.6}$$

A dual problem is given by (cf. Appendix C)

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{n} r_i + \sum_{j=1}^{n} p_j \\ &\text{subject to} \quad r_i + p_j \geq a_{ij}, \qquad \forall\, i,\ j \in A(i). \end{aligned} \tag{3.7}$$

We are considering a maximization as in Eq. (3.6) rather than a minimization problem in order to make the economic interpretation of the algorithm more transparent. It is also convenient for our purposes to use the constraint $0 \leq f_{ij}$ rather than $0 \leq f_{ij} \leq B$, with $B \geq 1$. None of these changes are of consequence, and problem (3.6) is essentially identical with the assignment problem considered earlier (cf. Fig. 5.1.1). Therefore, there is an integer optimal solution that assigns each person $i$ to a distinct object $j_i \in A(i)$ so that

$$\sum_{i=1}^{n} a_{ij_i}$$

is maximized over all such assignments.

We see from Eq. (3.7) that the cost of the dual problem is minimized when $r_i$ equals the maximum value of $a_{ij} - p_j$ over $j \in A(i)$. Thus, an equivalent form of the dual problem is

$$\begin{aligned} &\text{minimize} \quad q(p) \\ &\text{subject to no constraints on } p, \end{aligned} \tag{3.8}$$

where $p$ is the vector of object prices $p_j$, and

$$q(p) = \sum_{i=1}^{n} \max_{j \in A(i)} \{a_{ij} - p_j\} + \sum_{j=1}^{n} p_j. \tag{3.9}$$

For a given price vector $p$, the scalar

$$\pi_i = \max_{j \in A(i)} \{a_{ij} - p_j\} \tag{3.10}$$

is called the *profit margin* of person $i$ corresponding to $p$. It is helpful to think of $p_j$ as the amount of money that a person must pay when assigned to $j$. Therefore, for a given price vector $p$, $a_{ij} - p_j$ can be thought of as the benefit person $i$ associates with being assigned to object $j$. In this context, the name "profit margin" for $\pi_i$ as given by Eq. (3.10) becomes meaningful.

It is straightforward to verify that the complementary slackness conditions for an assignment $S$ (not necessarily complete) and a price vector $p$ can be written as

$$a_{ij_i} - p_{j_i} = \max_{j \in A(i)} \{a_{ij} - p_j\}, \qquad \forall\ (i, j_i) \in S.$$

A necessary and sufficient condition for $S$ and $p$ to be primal and dual optimal is that $S$ is complete and that $S$ and $p$ satisfy complementary slackness. Thus, at an optimal assignment, each person is assigned to an object attaining the maximum in the profit margin definition (3.10).

A relaxation of the complementary slackness condition is to allow persons to be assigned to objects that come within $\epsilon$ of attaining the maximum in Eq. (3.10). This can be seen to be equivalent to the $\epsilon$–CS condition (3.2) specialized to the assignment problem. Formally, we say that an assignment $S$ and a price vector $p$ satisfy $\epsilon$–CS if

$$\pi_i - \epsilon = \max_{k \in A(i)} \{a_{ik} - p_k\} - \epsilon \leq a_{ij} - p_j, \qquad \forall\ (i, j) \in S, \tag{3.11}$$

where $\pi_i$ is given by Eq. (3.10), and $\epsilon$ is a nonnegative constant.

We now describe formally the auction algorithm. We fix $\epsilon > 0$, and we start with some assignment (possibly empty) and price vector satisfying $\epsilon$–CS. The algorithm proceeds iteratively and terminates when a complete assignment is obtained. At the start of the generic iteration we have an assignment $S$ and a price vector $p$ satisfying $\epsilon$–CS. The iteration preserves the $\epsilon$–CS condition and consists of two phases: the *bidding phase* and the *assignment phase* described in the following.

**Bidding Phase:**

For each person $i$ that is unassigned under the assignment $S$:

1. Compute the "current value" of each object $j \in A(i)$ given by

$$v_{ij} = a_{ij} - p_j. \tag{3.12}$$

2. Find a "best" object $j^*$ having maximum value

$$v_{ij^*} = \max_{j \in A(i)} v_{ij},$$

and find the best value offered by objects other than $j^*$

$$w_{ij^*} = \max_{j \in A(i), j \neq j^*} v_{ij}. \tag{3.13}$$

[If $j^*$ is the only object in $A(i)$, we define $w_{ij^*}$ to be $-\infty$ or, for computational purposes, a number that is much smaller than $v_{ij^*}$.]

3. Compute the "bid" of person $i$ given by

$$b_{ij^*} = p_{j^*} + v_{ij^*} - w_{ij^*} + \epsilon = a_{ij^*} - w_{ij^*} + \epsilon. \tag{3.14}$$

[We characterize this situation by saying that person $i$ bid for object $j^*$, and that object $j^*$ received a bid from person $i$. The algorithm works if the bid has any value between $p_{j^*} + \epsilon$ and $p_{j^*} + v_{ij^*} - w_{ij^*} + \epsilon$, but it tends to work fastest for the maximal choice of Eq. (3.14).]

**Assignment Phase:**

For each object $j$:

Let $P(j)$ be the set of persons from which $j$ received a bid in the bidding phase of the iteration. If $P(j)$ is nonempty, increase $p_j$ to the highest bid:

$$p_j := \max_{i \in P(j)} b_{ij}, \tag{3.15}$$

remove from the assignment $S$ any pair $(i, j)$ (if $j$ was assigned to some $i$ under $S$), and add to S the pair $(i^*, j)$, where $i^*$ is a person in $P(j)$ attaining the maximum above.

It is seen that during an iteration, the objects whose prices are changed are the ones that received a bid during the iteration. Each price change involves an increace of at least $\epsilon$. To see this, note that from Eqs. (3.12) to (3.14) we have $b_{ij^*} = a_{ij^*} - w_{ij^*} + \epsilon \geq a_{ij^*} - v_{ij^*} + \epsilon = p_{j^*} + \epsilon$, and the conclusion follows from Eq. (3.15). At the end of the iteration, we have a new assignment that differs from the preceding one in that each object that received a bid is now assigned to some person that was unassigned at the start of the iteration. However, the assignment at the end of the iteration need not have more pairs than the one at the start of the iteration, because it is possible that all objects that received a bid were assigned at the start of the iteration.

A first important fact is that the algorithm preserves $\epsilon$–CS throughout its execution, that is, if the assignment and price vector available at the start of an iteration satisfy $\epsilon$–CS, the same is true for the assignment and price vector obtained at the end of the iteration. To see this, suppose that object $j^*$ received a bid from person $i$ and was assigned to $i$ during the iteration. Let $p_j$ and $p'_j$ be the object prices before and after the assignment phases, respectively. Then we have [cf. Eqs. (3.14) and (3.15)]

$$p'_{j^*} = b_{ij^*} = a_{ij^*} - w_{ij^*} + \epsilon. \tag{3.16}$$

Using this equation and the fact $p'_j \geq p_j$ for all $j$, it follows that

$$a_{ij^*} - p'_{j^*} = a_{ij^*} - b_{ij^*} = w_{ij^*} - \epsilon = \max_{j \in A(i), j \neq j^*} \{a_{ij} - p'_j\} - \epsilon. \tag{3.17}$$

This equation implies that

$$a_{ij^*} - p'_{j^*} \geq \max_{j \in A(i)} \{a_{ij} - p'_j\} - \epsilon, \tag{3.18}$$

which shows that the $\epsilon$–CS condition (3.11) continues to hold after the assignment phase of an iteration for a pair $(i, j^*)$ that entered the assignment during the iteration. Consider also any pair $(i, j^*)$ that belonged to the assigment just before an iteration, and also belongs to the assignment after the iteration. Then $j^*$ must not have received a bid during the iteration, so $p'_{j^*} = p_{j^*}$. Therefore, Eq. (3.18) holds in view of the $\epsilon$–CS condition that holds prior to the iteration and the fact $p'_j \geq p_j$ for all $j$.

Figure 5.3.5 indicates how each bidding and subsequent assignment phase can be interpreted as a Jacobi–like relaxation step for minimizing the dual function $q(p)$ of Eq. (3.9). In particular, *the price $p_j$ of each object $j$ that received a bid during the assignment phase is increased to either a value that minimizes $q(p)$ when all other prices are kept constant or else exceeds the largest such value by no more than $\epsilon$.* To see this, suppose that at some iteration, there is a bid for object $j$, raising its price from $p_j$ to $p'_j$. Then

$$p'_j = \max\{a_{ij} - w_{ij} \mid i \text{ was unassigned, } j \in A(i), \text{ and } j \text{ received a bid from } i\} + \epsilon, \tag{3.19}$$

$$p_j \geq \max\{a_{ij} - w_{ij} \mid i \text{ was unassigned, } j \in A(i), \text{ and } j \text{ did not receive a bid from } i\}. \tag{3.20}$$

Since whenever an object receives a bid, its price increases by at least $\epsilon$, we have

$$p'_j \geq p_j + \epsilon,$$

so from Eqs. (3.19) and (3.20), we obtain

$$p'_j \geq \max\{a_{ij} - w_{ij} \mid i \text{ was unassigned and } j \in A(i)\} + \epsilon. \tag{3.21}$$

Since the algorithm maintains the $\epsilon$–CS condition (3.11) throughout, we have that if at the start of the iteration person $i$ was assigned to some $k \neq j$ and $j \in A(i)$, then

$$a_{ij} - p_j \leq \pi_i \leq a_{ik} - p_k + \epsilon \leq w_{ij} + \epsilon.$$

Using this relation and the fact $p'_j \geq p_j + \epsilon$, we obtain

$$p'_j \geq p_j + \epsilon \geq a_{ij} - w_{ij}$$

and

$$p'_j \geq \max\{a_{ij} - w_{ij} \mid i \text{ was assigned to some } k \neq j, \text{ and } j \in A(i)\}. \tag{3.22}$$

Combining Eqs. (3.21) and (3.22), we obtain

$$p'_j \geq \max\{a_{ij} - w_{ij} \mid i \text{ was not assigned to } j, \text{ and } j \in A(i)\}.$$

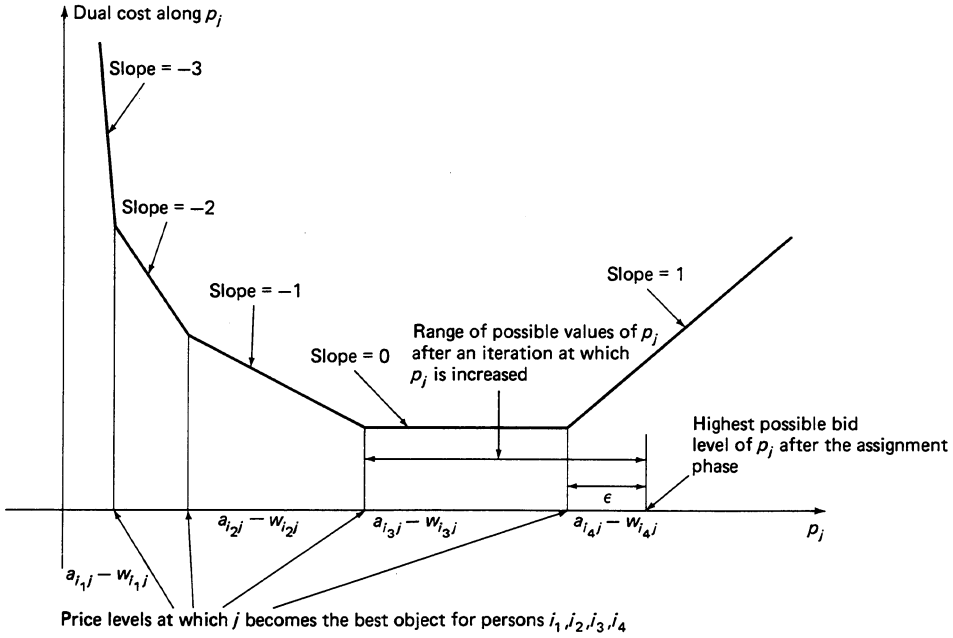



**Figure 5.3.5** Form of the dual cost along the price coordinate $p_j$. From Eq. (3.9), the right directional derivative of $q$ along $p_j$ is

$$d_j^+ = 1 - (\text{number of persons } i \text{ with } j \in A(i) \text{ and } p_j < a_{ij} - w_{ij})$$

where $w_{ij}$ is given by Eq. (3.13). The break points are $a_{ij} - w_{ij}$ for all $i$ such that $j \in A(i)$. If $p_j < a_{ij} - w_{ij}$, then an unassigned person $i$ bids for object $j$ the amount $a_{ij} - w_{ij} + \epsilon$. The price $p_j$ after the assignment phase is increased to $\epsilon$ plus the highest level $a_{ij} - w_{ij}$ over all unassigned persons $i$ with $j \in A(i)$.

Since there can be at most one person assigned to $j$, it follows from the form of the dual cost shown in Fig. 5.3.5, that $p_j'$ is no less than the smallest value of $p_j$ that minimizes $q(p)$. Combining this fact with Eq. (3.19), we see that $p_j'$ has the property stated in the beginning of this paragraph.

Note that the dual cost (3.9) can deteriorate after a price increase. However, the cost deterioration is at most $\epsilon$. Similarly with the $\epsilon$–relaxation method, for $\epsilon$ small enough, an optimal solution can still be obtained thanks to the rounding introduced by the integer nature of the problem data and the fact that $\epsilon$–CS holds at termination [cf. Eq. (3.11)].

Figure 5.3.5 suggests a variation of the algorithm, whereby, in addition to all unassigned persons, each *assigned* person $i$ bids for its *own* assigned object $j$ the amount $a_{ij} - w_{ij} + \epsilon$. This variation can be useful under some circumstances.

The above algorithm can be viewed as a *Jacobi* version of the relaxation idea since the bids of all unassigned persons bid are calculated simultaneously and the prices of objects that receive a bid are raised simultaneously. An alternative is a *Gauss–Seidel*

version, whereby a single unassigned person bids for an object and the price rise of the object is taken into account when the next bid by an unassigned person takes place. This version is just as valid as the Jacobi version and in fact tends to converge somewhat faster in practice, but is generally less parallelizable because the corresponding dependency graph can be quite dense (cf. the discussion of Subsection 1.2.4).

Suppose now that the algorithm terminates with the final (complete) assignment $\{(i, j_i) \mid i = 1, \ldots, n\}$, the object prices $p_j$, and the profit margins $\pi_i$ given by Eq. (3.10). Then, by adding the $\epsilon$–CS condition (3.11) over $i$, it is seen that

$$\sum_{i=1}^{n} a_{ij_i} \geq \sum_{i=1}^{n} (\pi_i + p_{j_i}) - n\epsilon.$$

If A* is the optimal primal value and the (equal) optimal dual value, we have, using the relation above,

$$A^* \geq \sum_{i=1}^{n} a_{ij_i} \geq \sum_{i=1}^{n} (\pi_i + p_{j_i}) - n\epsilon \geq A^* - n\epsilon,$$

where the last step follows from the feasibility of the scalars $\pi_i$ and $p_{j_i}$ for the dual problem (3.7). Therefore, the assignment $\{(i, j_i) \mid i = 1, \ldots, n\}$ is within $n\epsilon$ of being optimal. Since $a_{ij}$ are integer, an optimal assignment is obtained when $\epsilon < 1/n$. Thus, we have shown the following:

**Proposition 3.3.** An assignment $\{(i, j_i) \mid i = 1, \ldots, n\}$ obtained upon termination of the auction algorithm is within $n\epsilon$ of being optimal, and is optimal if $\epsilon < 1/n$.

The next result asserts that the algorithm terminates assuming existence of at least one feasible assignment. The proof relies on the following facts:

**(a)** Once an object is assigned, it remains assigned throughout the remainder of the algorithm's duration. Furthermore, except at termination, there will always exist at least one object that has never been assigned, and has a price equal to its initial price. This is because a bidding and assignment phase can result in a reassignment of an already assigned object to a different person, but cannot result in the object becoming unassigned.

**(b)** Each time an object receives a bid, its price increases by at least $\epsilon$ [cf. Eqs. (3.14) and (3.15)]. Therefore if the object receives a bid an infinite number of times, its price increases to $\infty$.

**(c)** For every $|A(i)|$ bids by person $i$, where $|A(i)|$ is the number of objects in the set $A(i)$, the profit margin $\pi_i$ as defined by Eq. (3.10) decreases by at least $\epsilon$. This is because a bid by person $i$ either decreases $\pi_i$ by at least $\epsilon$, or else leaves $\pi_i$ unchanged because there is more than one object $j$ attaining the maximum in Eq. (3.10). However, in the latter case, the price of the object $j^*$ receiving the bid

will increase by at least $\epsilon$, and object $j^*$ will not receive another bid by person $i$ until $\pi_i$ decreases by at least $\epsilon$. The conclusion is that if a person $i$ bids an infinite number of times, $\pi_i$ must decrease to $-\infty$.

**Proposition 3.4.** If at least one complete assignment exists, the algorithm terminates in a finite number of steps.

***Proof.*** If the algorithm continues indefinitely, the prices of a proper [cf. (a) above] subset $J^\infty$ of objects increases to $\infty$, while the profit margins $\pi_i$ of a subset $I^\infty$ of persons decrease to $-\infty$, [cf. (c) above]. Furthermore, eventually, in view of Eq. (3.11), at any given time, each object in $J^\infty$ can only be assigned to a person from $I^\infty$ , and a person from $I^\infty$ will either be assigned to an object in $J^\infty$ or be unassigned. Also, in view of (c) above, eventually only persons from $I^\infty$ will be unassigned. Therefore, the cardinality of $I^\infty$ is greater than the cardinality of $J^\infty$, while, in view of Eq. (3.11), we have $J^\infty \supset A(i)$ for all $i$ in $I^\infty$. This contradicts the existence of a complete assignment. **Q.E.D.**

Practical experience with the serial version of the auction algorithm has shown that it is at least competitive with the best alternative serial algorithms for the assignment problem, particularly for sparse problems [Ber88], [BeE88]. It is sometimes important, however, to combine the algorithm with the scaling technique described in the next subsection.

### 5.3.2 Parallel Versions of the $\epsilon$–Relaxation and the Auction Algorithms

In this subsection, we discuss the parallel implementation aspects of the auction and $\epsilon$–relaxation algorithms. It is clear that both the bidding and the assignment phases of the auction algorithm are highly parallelizable. In the extreme case of a fine grain parallel computing environment, where there is a processor associated with each person and a processor associated with each object, all unassigned persons/processors can compute their bids simultaneously and communicate them to the relevant objects/processors. Those object/processors that receive at least one bid can determine the highest bidder simultaneously and communicate to the relevant persons/processors the changes in the current assignment and price vector. A similar implementation is possible in systems where there are relatively few processors communicating via an interconnection network such as a hypercube. Each processor is given the responsibility of computing the bids of several persons, and of updating the assignments and prices of several objects. At the end of the bidding phase, the bids are transmitted to the appropriate processors using some form of total exchange algorithm that depends on the interconnection network and on the sparsity structure of the assignment problem graph (cf. Subsection 1.3.4). At the end of the assignment phase, the updated object assignments and prices are transmitted to all processors via a multinode broadcast.

The auction algorithm is also well suited for implementation in a shared memory machine. Here the processors of the system perform tasks such as bid calculations,

object assignments, and price updates. A synchronization mechanism is required for strict separation of the bidding and the assignment phases. In particular, it is necessary that the bids of all unassigned persons are calculated before the price or assignment of any object is updated. This separation is not necessary in an asynchronous implementation of the type to be discussed in the next chapter.

We now discuss how the $\epsilon$–relaxation algorithm can be implemented in a message–passing system that assigns a separate processor to each node. This processor is charged with the responsibility of carrying out up iterations at the node and communicating the results to the adjacent processor/nodes. Our discussion applies with minor modifications to systems with few processors, where several nodes are assigned to each processor.

There are three basic parallel implementation modes for the $\epsilon$–relaxation method. The first two, discussed here, are synchronous and will be referred to as the Gauss–Seidel and Jacobi versions in view with their similarity with the Gauss–Seidel and Jacobi relaxation methods discussed in Chapter 3. The third is a totally asynchronous version, and will be discussed in the next chapter (Section 6.5).

The synchronous algorithms are operated in phases, as discussed in Section 1.4. Some nodes/processors $i$ with $g_i > 0$ at the start of the phase execute a complete or partial up iteration during the phase, and the results of the iteration are communicated to all adjacent nodes. A node cannot proceed to the next phase before it knows the results of the computation (if any) at all adjacent nodes during the preceding phase.

In the synchronous Gauss–Seidel version, the set of nodes is partitioned into subsets. Each subset should not contain a pair of nodes joined by an arc. In each phase, a single subset is selected and the positive surplus nodes of this subset only execute an up iteration. Because no two adjacent nodes execute an up iteration concurrently, it is seen that the Gauss–Seidel version is mathematically equivalent to a sequential version with a specific order for choosing nodes to execute an up iteration [see the discussion in Subsection 1.2.4; the dependency graph here contains the bidirectional arc $(i, j)$ if either $(i, j)$ or $(j, i)$ is an arc of the graph of the problem]. Note that for transportation problems, we can use just two subsets; the subset of all sources and the subset of all sinks.

The Gauss–Seidel version has the drawback that some positive surplus nodes may be idle during some phases. This motivates the synchronous Jacobi version, whereby all positive surplus nodes execute an up iteration at every phase based on the prices and flows of adjacent nodes and arcs at the start of the phase. At the end of the phase, the price of each node and the flows of all its incident arcs are communicated to its corresponding adjacent nodes. There is an issue here on how two adjacent nodes $i$ and $j$ agree on a common value of the flow of the arc $(i, j)$ joining them. The problem is that $f_{ij}$ may be simultaneously modified by both $i$ and $j$ during a phase. When only one of the nodes $i$ and $j$ increases its price during a phase, we require that the value of $f_{ij}$ as set by the node that increased its price is accepted by the other node. We resolve situations where both nodes $i$ and $j$ increase their prices during a phase as follows: if at the end of phase $k$, the prices $p_i$ and $p_j$ have become such that $(i, j)$ is $\epsilon$–active (or $\epsilon$–inactive), then $f_{ij}$ is set to $c_{ij}$ (or $b_{ij}$) by both nodes $i$ and $j$; otherwise, both nodes $i$ and $j$ set $f_{ij}$ at the value of $f_{ij}$ that prevailed at the start of phase $k$.

It can be seen that with this rule, $\epsilon$–CS is preserved, and that at the end of each phase $k$, we have $g_i \geq 0$ for all nodes $i$ having $g_i \geq 0$ at the start of phase $k$. With these observations, the proof of Prop. 3.2 goes through almost verbatim, thereby showing that the Jacobi version of the algorithm terminates in a finite number of phases.

An interesting question relates to the speedup that can be attained with a parallel implementation of the $\epsilon$–relaxation method. We first note here that the maximum number of processors that can be executing up iterations in parallel at any one time cannot exceed the number of nodes with positive surplus. Computational experimentation shows that this number is typically quite large in the early stages of the computation. Near termination of the algorithm, however, most nodes have zero surplus, so the number of processors actively engaged in iterations is quite small. In fact, we give an example at the end of the next subsection where a parallel implementation of the algorithm leads to no appreciable speedup over a serial implementation. This example represents worst case behavior. Limited experience with parallel implementations of the $\epsilon$–relaxation method and with the auction algorithm indicates that a speedup of the order of 10 should be attainable in many computing systems. This is only a rough estimate. Further experimentation and research is needed to establish the potential of parallel implementations of the $\epsilon$–relaxation method and to provide a comparison with parallel implementations of the multiple node relaxation method of the previous section.

## EXERCISES

**3.1.** Assume that for some price vector $p$, the dual cost along the $i$th price coordinate, $q(p_1, \ldots, p_{i-1}, \xi, p_{i+1}, \ldots, p_{|N|})$, attains a maximum over $\xi$ in the interval $[\underline{p}_i, \bar{p}_i]$. Show that an up iteration at node $i$ starting at a price value $p_i < \underline{p}_i - \epsilon$ sets $p_i$ to $\bar{p}_i + \epsilon$, [cf. Fig. 5.3.4(a)]. Assume instead that the maximum is attained in the interval $[\underline{p}_i, \infty)$. Show that an up iteration at node $i$ starting at a price value $p_i < \underline{p}_i - \epsilon$ sets $p_i$ to $\underline{p}_i + \epsilon$, [cf. Fig. 5.3.4(b)]. Show that under either one of the preceding assumptions, $p_i$ is set to a value that is within $\epsilon$ of some maximizing point of the dual cost along the $i$th price coordinate.

**3.2. (Mixing of Up and Down Iterations, [Tse86] and [Eck87].)** Define a *down iteration*, which is similar to an up iteration, but only applies to nodes $i$ with $g_i < 0$, increases $g_i$, and decreases $p_i$.

**(a)** Discuss briefly how to modify the proof of Prop. 3.2 to show the finiteness of the following algorithm: while there exists a node $i \in N$ with $g_i < 0$, select such an $i$ and perform a down iteration upon it.

**(b)** Consider the following algorithm: while there exists an $i \in N$ with $g_i \neq 0$, select such an $i$ and perform an up iteration if $g_i > 0$ or a down iteration if $g_i < 0$. Consider also the following network problem: $N = \{1, 2, 3, 4\}$, $A = \{(1,2), (1,3), (2,4), (3,4)\}$, $s_1 = 1$, $s_2 = s_3 = 0$, $s_4 = -1$, $a_{ij} = 0$, $b_{ij} = 0$, and $c_{ij} = 2$ for all $(i,j) \in A$. Show that for this problem the algorithm is not guaranteed to be finite for $\epsilon = 1$ and the initial price vector $p = 0$. *Hint:* Give a sequence of price and flow modifications meeting the specifications of the algorithm, in which the state $p_1 = 1$, $p_2 = p_3 = 0$, $p_4 = -1$, $f_{12} = 1$, $f_{13} = f_{24} = 0$, $f_{34} = 1$ recurs infinitely many times.

**3.3.** Consider the multiple node relaxation iteration of Section 5.2, and also the primal–dual methods of Exercise 2.3. Show that if the terms "balanced", "active", and "inactive" are replaced by "$\epsilon$–balanced", "$\epsilon$–active", and "$\epsilon$–inactive", respectively, then the resulting methods terminate in a finite number of iterations and that the final pairs $(f, p)$ obtained satisfy $\epsilon$–CS.

**3.4.** In this exercise, we consider a variation of an up iteration that involves *degenerate price increases*. A degenerate price increase raises the price of a node that currently has zero surplus to the maximum possible value that does not violate $\epsilon$–CS with respect to the current flow vector (assuming there exists such a maximum value). One example of such a price increase occurs when Step 4 of the up iteration is executed with $g_i = 0$. Show that Prop. 3.2 holds even if degenerate price rises are allowed in the up iteration.

**3.5. (Relation of the $\epsilon$–Relaxation Method and the Auction Algorithm.)** Consider the assignment problem of Fig. 5.1.1 having $n$ sources, $n$ sinks, and an arbitrary set $A$ of source–to–sink arcs. We say that source $i$ is assigned to sink $j$ if $(i, j)$ has positive flow. We consider a version of the $\epsilon$–relaxation algorithm in which up iterations are organized as follows: between iterations (and also at initialization), only source nodes $i$ can have positive surplus. Each iteration does the following: (1) finds any unassigned source $i$ (i.e., one with positive surplus), and performs an up iteration at $i$; and (2) takes the sink $j$ to which $i$ was consequently assigned, and performs an up iteration at $j$, even if $j$ has zero surplus. (If $j$ has zero surplus, such an up iteration will consist of just a degenerate price rise; see Exercise 3.4.)

More specifically, an iteration by an unassigned source $i$ works as follows: (a) Source node $i$ sets its price to $p_j + a_{ij} + \epsilon$, where $j$ minimizes $p_k + a_{ik} + \epsilon$ over all $k$ for which $(i, k) \in A$. It then sets $f_{ij} = 1$, assigning itself to $j$. (b) Node $i$ then raises its price to $p_{j'} + a_{ij'} + \epsilon$, where $j'$ minimizes $p_k + a_{ik} + \epsilon$ for $k \neq j$, $(i, k) \in A$. (c) If sink $j$ had a previous assignment $f_{i'j} = 1$, it breaks the assignment by setting $f_{i'j} := 0$ (one can show inductively that if this occurs, $p_j = p_{i'} - a_{i'j} + \epsilon$). (d) Sink $j$ then raises its price $p_j$ to

$$p_i - a_{ij} + \epsilon = p_{j'} + a_{ij'} - a_{ij} + 2\epsilon.$$

Show that the corresponding algorithm is equivalent to the sequential (Gauss–Seidel) version of the auction algorithm.

**3.6. (The Auction Algorithm with Similar Objects [BeC87].)** Given the assignment problem of Subsection 5.3.1, we say that two objects $j$ and $j'$ are *similar*, and write $j \sim j'$, if for all persons $i = 1, \ldots, n$, we have

$$j \in A(i) \quad \Rightarrow \quad j' \in A(i) \quad \text{and} \quad a_{ij} = a_{ij'}.$$

For each object $j$, the set of all objects similar to $j$ is called the similarity class of $j$ and is denoted $M(j)$. Consider a variation of the auction algorithm that is the same as the one of Subsection 5.3.1 except for one difference: in the bidding phase, $w_{ij*}$ is defined now as

$$w_{ij*} = \max_{j \in A(i), j \notin M(j*)} v_{ij}$$

(instead of $w_{ij*} = \max_{j \in A(i), j \neq j*} v_{ij}$). Show that, assuming the initial assignment $S$ satisfies $\epsilon$–CS together with the initial vector $\hat{p}$ defined by

$$\hat{p}_j = \min_{k \in M(j)} p_k, \qquad j = 1, \ldots, n,$$

that is,

$$\max_{k \in A(i)} \{a_{ik} - \hat{p}_k\} - \epsilon \leq a_{ij} - \hat{p}_j, \qquad \forall\, (i,j) \in S,$$

the same is true of the assignment and the vector $\hat{p}$ obtained at the end of each assignment phase. Show also that the algorithm terminates finitely with an optimal assignment if $\epsilon < 1/n$.

**3.7. (The Auction Algorithm for Network Problems with Unit Capacity Bounds.)** Consider the linear network flow problem (LNF) for the case where the feasible flow range of each arc $(i,j)$ is $0 \leq f_{ij} \leq 1$. Convert this problem into a transportation problem, as in Fig. 5.1.4 of Section 5.1, and describe the application of the auction algorithm of Exercise 3.6.

**3.8. (The Auction Algorithm for Transportation Problems [BeC87].)** Consider the assignment problem. We say that two persons $i$ and $i'$ are *similar*, and write $i \sim i'$, if for all objects $j = 1, \ldots, N$, we have

$$j \in A(i) \quad \Rightarrow \quad j \in A(i') \quad \text{and} \quad a_{ij} = a_{i'j}.$$

The set of all persons similar to $i$ is called the similarity class of $i$.

**(a)** Generalize the auction algorithm with similar objects given in Exercise 3.6 so that it takes into account both similar persons and similar objects. *Hint:* Consider simultaneous bids by all persons in the same similarity class.

**(b)** Show how the algorithm of part (a) can be applied to transportation problems.

**3.9. (The Auction Algorithm for Incomplete Assignment Problems.)**

**(a)** Derive a variation of the auction algorithm for an assignment problem where the number of persons $m$ is greater than the number of objects $n$. All objects must be assigned to distinct persons. *Hint:* Introduce $m - n$ additional objects connected to all persons with zero cost arcs. Use the auction algorithm with similar objects of Exercise 3.7.

**(b)** Repeat part (a) for the case where $m < n$ and all persons must be assigned to distinct objects. *Hint:* Introduce $n - m$ additional persons which are similar (cf. Exercise 3.8).

**(c)** Repeat part (b) for the case where a person need not be assigned to an object, that is, the constraints of the problem are

$$\sum_{j \in A(i)} f_{ij} \leq 1, \qquad \sum_{\{i \mid j \in A(i)\}} f_{ij} \leq 1, \qquad f_{ij} \geq 0.$$

**3.10. (An Auction–Like Algorithm Based on the Alternating Direction Method.)** Consider a transportation problem of the form

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

subject to

$$\sum_{j\in O(i)} f_{ij} = \alpha_i, \qquad \forall\, i = 1,\ldots,m$$

$$\sum_{i\in I(j)} f_{ij} = \beta_j, \qquad \forall\, j = 1,\ldots,n$$

$$0 \le f_{ij}, \qquad \forall\, (i,j) \in A,$$

where for all $i$ and $j$,

$$O(i) = \{j \mid (i,j) \in A\}, \qquad I(j) = \{i \mid (i,j) \in A\}.$$

Consider an iteration where we first calculate, for all sources $i$ in parallel,

$$\{f_{ij}(t+1) \mid j \in O(i)\}$$

$$= \arg \min_{\substack{\sum_{j\in O(i)} f_{ij} = \alpha_i \\ f_{ij} \ge 0}} \left\{ \sum_{j\in O(i)} \left[ \big(a_{ij} + p_j(t)\big) f_{ij} + \frac{c}{2} \big(f_{ij} - f_{ij}(t) + \bar{g}_j(t)\big)^2 \right] \right\},$$

and then we calculate, for all sinks $j$ in parallel,

$$p_j(t+1) = p_j(t) + c\bar{g}_j(t+1),$$

where for all $j$ and $t$,

$$\bar{g}_j(t) = \frac{1}{d_j} \left( \sum_{i\in I(j)} f_{ij}(t) - \beta_j \right)$$

and $d_j$ is the number of sources in $I(j)$. The initial flows $f_{ij}(0)$ and prices $p_j(0)$ are arbitrary, and $c$ is a positive constant. Show that this method is a special case of the alternating direction method of multipliers of Subsection 3.4.4.

## 5.4 COMPLEXITY ANALYSIS OF THE $\epsilon$–RELAXATION METHOD AND ITS SCALED VERSION

In this section, we derive a bound on the order of time taken by the $\epsilon$–relaxation algorithm. We then introduce a scaled version of the method with a particularly favorable time bound. Our analysis assumes the following:

**Assumption 4.1.** There exists at least one feasible solution of problem (LNF).

**Assumption 4.2.** All arc cost coefficients are integer multiples of $\epsilon$.

**Assumption 4.3.** All starting prices are integer multiples of $\epsilon$, all starting flows are integer, and together they satisfy $\epsilon$–CS. Furthermore, initially there are no $\epsilon^+$–unblocked or $\epsilon^-$–unblocked arcs.

To achieve the last property required in Assumption 4.3 we can simply take initially $f_{ij} = c_{ij}$ for all $\epsilon^+$–balanced arcs $(i,j)$ and $f_{ij} = b_{ij}$ for all $\epsilon^-$–balanced arcs $(i,j)$, but better choices may be possible in particular situations.

A notion that is central in the subsequent complexity analysis is the so called *admissible graph*, which consists of the $\epsilon^+$–unblocked arcs and of the $\epsilon^-$–unblocked arcs with their directions reversed (i.e., the arcs along which flow is allowed to change according to the rules of the algorithm, with each arc oriented in the direction of the flow change). Formally, the admissible graph is defined as $G^* = (N, A^*)$, where an arc $(i,j)$ belongs to $A^*$ if and only if it is possible to "push" flow from $i$ to $j$ without an intervening price change according to the rules of the algorithm in Step 2 or 3. In other words, $A^*$ contains an arc $(i,j)$ if either $(i,j)$ is an $\epsilon^+$–unblocked arc of $A$ or $(j,i)$ is an $\epsilon^-$–unblocked arc of $A$. Note that the admissible graph depends on the current pair $(f,p)$ that satisfies $\epsilon$–CS and changes as the pair $(f,p)$ changes during the course of the algorithm. In particular, when flow is increased (or decreased) to the upper bound of an $\epsilon^+$–unblocked arc in Step 2 (or the lower bound of an $\epsilon^-$–unblocked arc in Step 3, respectively) of the up iteration, the arc is removed from $A^*$. Furthermore, when there is a price increase of a node $i$ during Step 4 of the up iteration, all the incident arcs of node $i$ that belonged to $A^*$ prior to the price increase are removed from $A^*$, and the incident arcs $(i,j)$ or $(j,i)$ that become $\epsilon^+$–unblocked or $\epsilon^-$–unblocked, respectively, following the price increase [i.e., those for which the minimum in Eq. (3.5) is attained] are added to $A^*$.

To organize the computation efficiently, it is necessary to maintain the admissible graph in a data structure. In particular, the incident arcs of each node $i$ that belong to the admissible graph are maintained in a list $L_i$, which is updated as necessary after each execution of a Step 2, 3, or 4 of an up iteration. We assume that the list $L_i$ is organized in a way that the addition and deletion of a single arc takes $O(1)$ computation; this is true, for example, if $L_i$ is a doubly linked list [Kru87]. Then it is seen that updating $L_i$ at the end of Steps 2, 3, and 4 of the up iteration takes $O(1)$, $O(1)$, and $O(d_i)$ time, respectively, where $d_i$ is the number of incident arcs of node $i$. As a result, updating the list $L_i$ does not affect the order of time needed for these steps.

The role of the admissible graph can be understood in terms of the example of Fig. 5.4.1. Here the admissible graph contains the cycle 2–3–2. As a result, arc flows change along the cycle by small increments for a number of times that can be as large as the arc flow bounds. We call this phenomenon *flow looping*. It can occur only when the admissible graph has cycles. It does not arise when $\epsilon < 1/|N|$, because then the admissible graph is always acyclic. To see this, note that if there existed a cycle, then by adding the condition for an $\epsilon^+$–balanced or $\epsilon^-$–balanced arc along the cycle, we obtain that the sum of the arc costs of the $\epsilon^-$–balanced arcs minus the sum of the arc costs of

the $\epsilon^+$–balanced arcs on the cycle equals $\epsilon$ times the number of arcs on the cycle. This is impossible when the coefficients $a_{ij}$ are integer and $\epsilon < 1/|N|$. When $\epsilon \geq 1/|N|$, it is necessary to choose the initial flows and prices so that the admissible graph is initially acyclic. Then all up iterations maintain the acyclicity of this graph, as will be shown in the proof of Prop. 4.1. Assumption 4.3 guarantees that the admissible graph initially has no arcs and is therefore trivially acyclic.

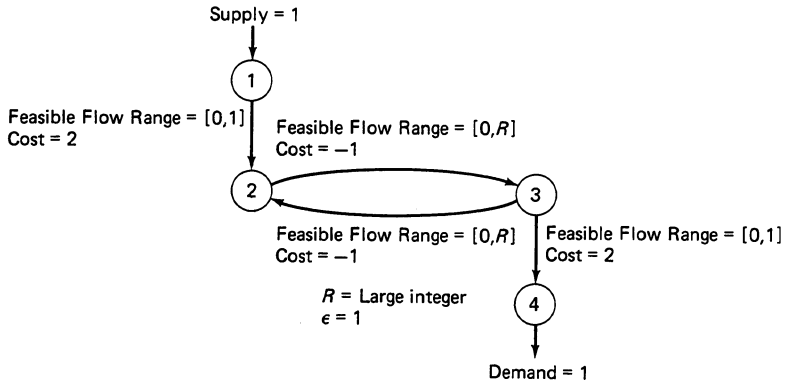



**Figure 5.4.1** An example demonstrating the role of Assumption 4.3 on the initial conditions. Initially, we choose $f = 0$ and $p = 0$, which do satisfy 1–CS, but not Assumption 4.3. The algorithm will push one unit of flow $R$ times around the cycle 2–3–2, implying $\Omega(R)$ solution time.

In order to maintain the acyclicity of the admissible graph, we need a special data structure and a restriction in the way the algorithm is operated. We introduce an order for choosing nodes in iterations. A *cycle* is a set of iterations whereby all nodes are chosen once in a given order, and an up iteration is executed at each node having positive surplus at the time its turn comes. The order in which nodes are taken up in a cycle can change from one cycle to the next. This node order is maintained in a linked list that is traversed from the first to the last element in each cycle. Each time a node $i$ changes price as a result of its up iteration within a cycle, node $i$ is removed from its present list position and is placed in the first list position. (This does not change the order in which the remaining nodes are taken up in the current cycle; only the order for the subsequent cycle is affected.) The initial list is arbitrary. It will be shown as part of the proof of the subsequent Prop. 4.1 that with the above method for operating the algorithm, the admissible graph is always acyclic.

**Assumption 4.4.** The algorithm is operated in cycles as described above.

The motivation for the linked list data structure is illustrated in Fig. 5.4.2, and is based on the admissible graph, which is acyclic at all times and defines a partial order on the nodes. Within any one cycle, and up to the point where a price change occurs, flow can only be pushed from a higher ranking (first in the list) to a lower ranking node

according to the order at the beginning of the cycle. Therefore, it is most efficient to iterate on higher ranking nodes first. Choosing the lower ranking node first may be wasteful since its surplus will be set to zero through an up iteration and can become positive again within the same cycle through an up iteration at a higher ranking node; this cannot happen if nodes are chosen according to the partial order induced by the admissible graph. This order will be shown to be consistent with the order of nodes in the linked list previously described. As a result, during a cycle, high ranking nodes are taken up for iteration before low ranking nodes. This effect can be described as sweeping the positive surplus along the admissible graph from top to bottom, so we call the corresponding implementation of the $\epsilon$–relaxation method *sweep implementation.*

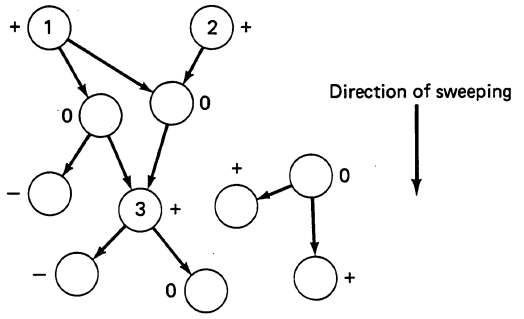



**Figure 5.4.2** Illustration of the admissible graph consisting of the $\epsilon^+$–unblocked arcs and the $\epsilon^-$–unblocked arcs with their directions reversed. These arcs specify the direction along which flow can be changed according to the rules of the algorithm. A "+" (or "−" or "0") indicates a node with positive (or negative or zero) surplus. The algorithm is operated so that the admissible graph is acyclic at all times. The sweep implementation, based on the linked list data structure, requires that the high ranking nodes (e.g., nodes 1 and 2 in the graph) are chosen for iteration before the low ranking nodes (e.g., node 3 in the graph).

We begin the complexity analysis by introducing some notation and terminology. For any path $H$, we denote by $s(H)$ and $t(H)$ the start and end nodes of $H$, respectively, and by $H^+$ and $H^-$ the sets of forward and backward arcs of $H$, respectively, as the path is traversed in the direction from $s(H)$ to $t(H)$. For any price vector $p$ and simple path $H$, we define

$$\begin{aligned} d_H(p) &= \max\left\{0, \sum_{(i,j)\in H^+}(p_i - p_j - a_{ij}) - \sum_{(i,j)\in H^-}(p_i - p_j - a_{ij})\right\} \\ &= \max\left\{0,\ p_{s(H)} - p_{t(H)} - \sum_{(i,j)\in H^+} a_{ij} + \sum_{(i,j)\in H^-} a_{ij}\right\}. \end{aligned} \tag{4.1}$$

Note that the second term in the maximum can be viewed as a "reduced cost length of $H$", being the sum of the reduced costs $p_i - p_j - a_{ij}$ over all arcs $(i,j) \in H^+$ less the sum of $p_i - p_j - a_{ij}$ over all arcs $(i,j) \in H^-$. For any flow vector $f$ satisfying the capacity constraints (1.2) we say that a simple path $H$ is *unblocked with respect to* $f$ if we have $f_{ij} < c_{ij}$ for all arcs $(i,j) \in H^+$ and we have $f_{ij} > b_{ij}$ for all arcs $(i,j) \in H^-$. In words, $H$ is unblocked with respect to $f$ if there is margin for sending positive flow along $H$ (in addition to $f$) from $s(H)$ to $t(H)$ without violating the capacity constraints.

For any price vector $p$ and flow vector $f$ satisfying both the conservation of flow and the capacity constraints (1.1) and (1.2), denote

$$D(p, f) = \max\{d_H(p) \mid H \text{ is a simple unblocked path with respect to } f\}. \quad (4.2)$$

In the exceptional case where there is no simple unblocked path with respect to $f$, we define $D(p, f) = 0$. In this case, we must have $b_{ij} = c_{ij}$ for all $(i, j)$ since any arc $(i, j)$ with $b_{ij} < c_{ij}$ gives rise to a one–arc unblocked path with respect to $f$.

Let

$$\beta(p) = \min\{D(p, f) \mid f \text{ satisfies constraints (1.1) and (1.2)}\}. \quad (4.3)$$

Since for a given $p$, there is only a finite number of values that $D(p, f)$ can take, it follows that the minimum in Eq. (4.3) is attained by some $f$. The following lemma shows that $\beta(p)$ provides a measure of suboptimality of the price vector $p$. The solution time estimate for the algorithm to be obtained shortly is proportional to $\beta(p^0)$, where $p^0$ is the initial price vector:

**Lemma 4.1.**

**(a)** If there exists a flow vector $f$ satisfying the conservation of flow and the capacity constraints (1.1) and (1.2), and satisfying $\gamma$–CS together with $p$ for some $\gamma \geq 0$, then

$$0 \leq \beta(p) \leq (|N| - 1)\gamma. \quad (4.4)$$

**(b)** $p$ is dual optimal if and only if $\beta(p) = 0$.

***Proof.***

**(a)** For each simple path $H$ that is unblocked with respect to $f$ and has $|H|$ arcs, we have, by adding the $\gamma$–CS condition along $H$ and using Eq. (4.1), $d_H(p) \leq |H|\gamma \leq (|N| - 1)\gamma$ and the result follows from Eqs. (4.2) and (4.3).

**(b)** If $p$ is optimal, then it satisfies complementary slackness together with some primal optimal vector $f$, so from Eq. (4.4) (with $\gamma = 0$), we obtain $\beta(p) = 0$. Conversely, if $\beta(p) = 0$, then from Eq. (4.3), we see that there must exist a primal feasible $f$ such that $D(p, f) = 0$. Hence, $d_H(p) = 0$ for all unblocked simple paths $H$ with respect to $f$. Applying this fact to single–arc paths $H$ and using the definition (4.1), we obtain that $f$ together with $p$ satisfy complementary slackness. It follows that $p$ and $f$ satisfy all the optimality conditions (1.8)–(1.11), and $p$ is optimal. **Q.E.D.**

**Proposition 4.1.** Under Assumptions 4.1 – 4.4, the $\epsilon$–relaxation algorithm terminates in $O\big(|N|^3 + |N|^2\beta(p^0)/\epsilon\big)$ time, where $p^0$ is the initial price vector.

***Proof.*** To economize on notation, we write $\beta$ in place of $\beta(p^0)$. We first show the following:

**Lemma 4.2.** The number of price increases at each node is $O(|N| + \beta/\epsilon)$.

***Proof.*** Let $f^0$ be a flow vector attaining the minimum in the definition (4.3) of $\beta(p^0)$. To explain the main argument better, we assume that $f^0$ is the zero vector. This can be done without loss of generality because we can transform the problem by replacing $c_{ij}$, $b_{ij}$, $f_{ij}$ and $s_i$ by $c_{ij} - f^0_{ij}$, $b_{ij} - f^0_{ij}$, $f_{ij} - f^0_{ij}$ and 0, respectively. The transformation does not change the surplus of any node, and does not change the prices generated by the algorithm. Let $(f, p)$ be a vector pair generated by the algorithm. If $g_t > 0$ for some node $t$, there must exist a node $s$ with $g_s < 0$ and a simple path $H$ with $s(H) = s$, $t(H) = t$, and such that $f_{ij} > 0$ for all $(i,j) \in H^+$ and $f_{ij} < 0$ for all $(i,j) \in H^-$. [This follows from the Conformal Realization Theorem (Appendix B). It can also be shown quickly from first principles: take $T_0 = \{t\}$, and given $T_k$, define

$$T_{k+1} = T_k \cup \big\{ j \notin T_k \mid \text{there is a node } i \in T_k,$$
$$\text{and either an arc } (i,j) \text{ such that } f_{ij} < 0, \text{ or an arc } (j,i) \text{ such that } f_{ji} > 0 \big\}.$$

If none of the negative surplus nodes belongs to any of the $T_k$, then the total surplus of the nodes in $\cup T_k$ is positive, while the forward arcs of the arc set separating $\cup T_k$ and its complement have nonnegative flow and the backward arcs have nonpositive flow. This is a contradiction, showing that a node $s$ with $g_s < 0$ and the aforementioned properties can be found.]

We thus conclude that the path $H$ is unblocked with respect to $f^0$. Hence, from Eq. (4.2), we must have $d_H(p^0) \leq D(p^0, f^0) = \beta$, and by using Eq. (4.1),

$$p^0_s - p^0_t - \sum_{(i,j)\in H^+} a_{ij} + \sum_{(i,j)\in H^-} a_{ij} \leq \beta. \tag{4.5}$$

Also, using $\epsilon$–CS, we have $p_j + a_{ij} \leq p_i + \epsilon$ for all $(i,j) \in H^+$ and $p_i \leq p_j + a_{ij} + \epsilon$ for all $(i,j) \in H^-$. By adding these conditions along $H$, we obtain

$$-p_s + p_t + \sum_{(i,j)\in H^+} a_{ij} - \sum_{(i,j)\in H^-} a_{ij} \leq |H|\epsilon \leq (|N| - 1)\epsilon \tag{4.6}$$

where $|H|$ is the number of arcs of $H$. We have $p^0_s = p_s$, since the condition $g_s < 0$ implies that the price of $s$ has not yet changed. Therefore, by adding Eqs. (4.5) and (4.6), we obtain

$$p_t - p^0_t < (|N| - 1)\epsilon + \beta \tag{4.7}$$

throughout the algorithm for all nodes $t$ with $g_t > 0$. Since all the starting prices and arc cost coefficients are integer multiples of $\epsilon$, it follows that the size of a price increase

is a positive integer multiple of $\epsilon$, and we see from Eq. (4.7) that the number of price increases of each node is $O(|N| + \beta/\epsilon)$. **Q.E.D.**

Note that the price bound (4.7) ensures that an infeasible problem instance can be detected by checking whether the total price rise of any node exceeds a known upper bound to $(|N| - 1)\epsilon + \beta$.

We now proceed with the proof of Prop. 4.1. The dominant computational requirements are:

**(1)** The computation required for price increases in Step 4.

**(2)** The computation required for Steps 2 or 3 for which the flow of the corresponding arc is set to its upper or its lower bound.

**(3)** The computation required for Steps 2 or 3 for which the flow of the corresponding arc is set to a value strictly between its upper and its lower bound.

We first make the general observation that by using the lists $L_i$ that hold the arcs of the admissible graph, the computation time to examine any one arc in the algorithm is $O(1)$. In particular, the time to execute Steps 1, 2, 3, and 4 (including the updating of the lists $L_i$) is $O(1)$, $O(1)$, $O(1)$, and $O(d_i)$, respectively, where $d_i$ is the number of incident arcs of the node $i$ iterated on. Since there are $O(|N| + \beta/\epsilon)$ price increases for each node, the requirements in (1) above are $O\big(|A|(|N| + \beta/\epsilon)\big)$ operations. Whenever an arc flow is set to either the upper or the lower bound due to an iteration at one of the end nodes, it takes a price increase of at least $2\epsilon$ by the opposite end node before the arc flow can change again. Therefore, there are $O(|N| + \beta/\epsilon)$ Steps 2 or 3 per arc for which the flow of the arc is set to its upper or lower bound. The computation time for each of these steps is $O(1)$, so the total requirements for (2) above are $O\big(|A|(|N| + \beta/\epsilon)\big)$ operations.

There remains to estimate the computational requirements for (3) above. At this point, we will use the fact that the algorithm is operated in cycles with the node order in each cycle determined by a linked list that is restructured in the course of the algorithm, as described earlier. We will demonstrate that the number of cycles up to termination is $O\big(|N|(|N| + \beta/\epsilon)\big)$. Given this, the proof of Prop. 4.2 can be completed as follows: for each cycle, there can only be one arc flow per node set to a value strictly between the upper and lower arc flow bound in Step 2 or 3. Therefore the total number of operations required for these steps [cf. (3) above] is $O\big(|N|^2(|N| + \beta/\epsilon)\big)$. Adding the computational requirements for (1) and (2) calculated earlier, we obtain an $O\big(|N|^2(|N| + \beta/\epsilon)\big) + O\big(|A|(|N| + \beta/\epsilon)\big)$ or $O\big(|N|^2(|N| + \beta/\epsilon)\big)$ time bound.

To show that the number of cycles up to termination is $O\big(|N|(|N| + \beta/\epsilon)\big)$, we use the admissible graph $G^* = (N, A^*)$ and we argue as follows: a node $i$ is called a *predecessor* of a node $j$ if a directed path exists from $i$ to $j$ in $G^*$. First, we claim that immediately following a price rise at node $j$, there are no arcs $(i, j)$ in $A^*$ that are incoming to $j$, and hence $j$ has no predecessors. To see this, note that if $(i, j) \in A$ is $\epsilon^+$–balanced after the price change, it must have been $\epsilon$–active beforehand, and,

hence, $f_{ij} = c_{ij}$, implying that $(i,j)$ is not in $A^*$. The $\epsilon^-$–balanced case is similar, establishing the claim. We next claim that $G^*$ is always acyclic. This is true initially because Assumption 4.3 implies that $A^*$ is empty. Flow change operations (Steps 2 and 3) can only remove arcs from $A^*$, so $G^*$ can acquire a cycle only immediately after a price rise at some node $j$, and the cycle must include that node. But since $j$ must then have no incoming incident arcs in the admissible graph, no such cycle is possible. This establishes the second claim. Finally, we claim that the node list maintained by the algorithm will always be compatible with the partial order induced by $G^*$, in the sense that every node will always appear in the list after all its predecessors. Again this is initially true because $A^*$ starts out empty. Furthermore a flow change operation does not create new predecessor relationships, while after the price of some node $i$ rises, $i$ can have no predecessors and is moved to the head of the list before any possible descendants. This establishes the claim.

Let $N^+$ be the set of nodes with positive surplus that have no predecessor with positive surplus, and let $N^0$ be the set of nodes with nonpositive surplus that have no predecessor with positive surplus. Then, as long as no price increase takes place, all nodes in $N^0$ remain in $N^0$, and execution of a complete up iteration at a node $i \in N^+$ moves $i$ from $N^+$ to $N^0$. If no node changed price during a cycle, then all nodes of $N^+$ will be added to $N^0$ by the end of the cycle, which implies that the algorithm terminates. Therefore, there will be a node price change during every cycle except possibly for the last one. Since the number of price increases per node is $O(|N| + \beta/\epsilon)$, there can be only $O\big(|N|(|N| + \beta/\epsilon)\big)$ cycles, leading to an $O\big(|N|^2(|N| + \beta/\epsilon)\big)$ overall time bound based on the argument given earlier. **Q.E.D.**

An upper bound for $\beta(p^0)$ is given by $(|N| - 1)C + p^+ - p^-$, where

$$p^+ = \max_{i \in N} p_i^0,$$

$$p^- = \min_{i \in N} p_i^0,$$

and $C$ is the arc cost range:

$$C = \max_{(i,j) \in A} |a_{ij}|. \tag{4.8}$$

Assuming that $p^+ - p^- = O(1)$, we obtain the time bound $O\big(|N|^3 C/\epsilon\big)$. The algorithm is indeed sensitive to $C$, as shown in the example of Fig. 5.4.3.

### Application to the Max–Flow Problem

For classes of problems with special structure, a better estimate of $\beta(p^0)$ may be possible. As an example, consider the max–flow problem formulation shown in Fig. 5.1.2. The artificial arc $(t,s)$ connecting the sink $t$ with the source $s$ has cost coefficient $-1$, and flow bounds $\beta_{ts} = 0$ and $c_{ts} = \sum_i c_{si}$. We assume that $a_{ij} = 0$ and $b_{ij} = 0 < c_{ij}$ for all other arcs $(i,j)$, and that $s_i = 0$ for all $i$. We apply the $\epsilon$–relaxation algorithm with initial prices

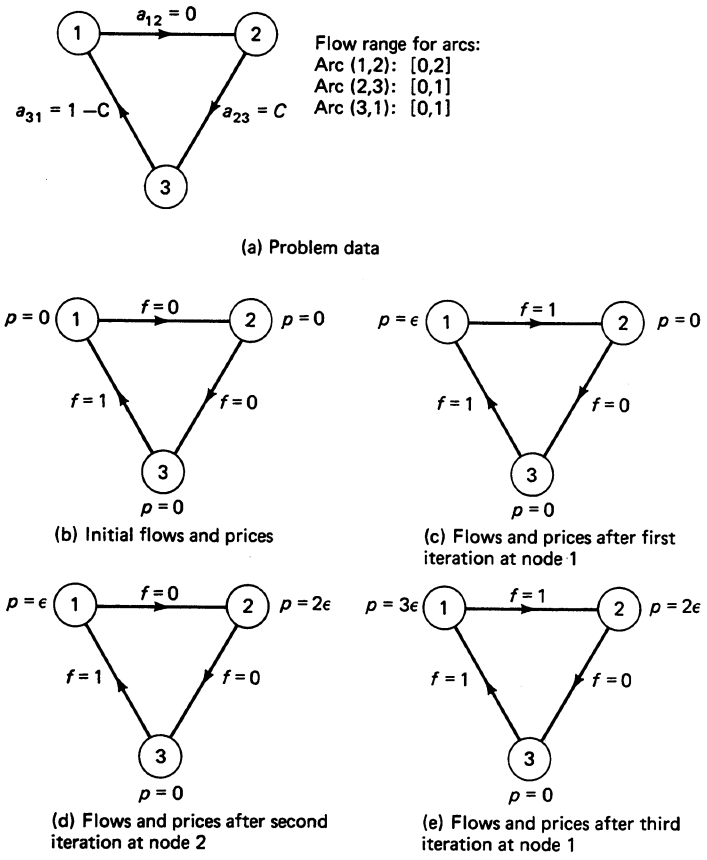



**Figure 5.4.3** Example showing that the pure form of the algorithm can take time that is proportional to the cost–dependent factor $C$. Here up iterations at node 1 alternate with up iterations at node 2 until the time when $p_1$ rises to the level $C - 1 + \epsilon$ and arc (3,1) becomes $\epsilon^-$–balanced, so that a unit of flow can be pushed back along that arc. At this time, the optimal solution is obtained. Since prices rise by increments of no more than $2\epsilon$, the number of up iterations is $\Omega(C/\epsilon)$.

and arc flows satisfying $\epsilon$–CS, where $\epsilon = 1/(|N| + 1)$ and $p^+ - p^- = O(1)$. Because there is only one arc that has nonzero $(-1)$ cost coefficient, we obtain $d_H(p^0) = O(1)$ for all paths $H$. Therefore, $\beta(p^0) = O(1)$ and Prop. 4.1 yields an $O(|N|^3)$ time bound. This bound is competitive with that of other max–flow algorithms [PaS82], and can only be improved through the use of sophisticated data structures [GoT86], [AhO86].

### 5.4.1 The Scaled Version of the Algorithm

Since the algorithm is sensitive to the arc cost range $C$, it is natural to consider cost scaling procedures involving solution of a sequence of approximations to the original problem, gradually increasing the accuracy of the cost coefficient data.

Consider the problem obtained from (LNF) by multiplying all arc cost coefficients by $|N|+1$, that is, the problem with arc cost coefficients

$$a'_{ij} = (|N|+1)a_{ij}, \qquad \forall\ (i,j) \in A.$$

We refer to this problem as (SNLF). If a pair $(f', p')$ satisfies 1–CS (namely, $\epsilon$–CS with $\epsilon = 1$) with respect to (SLNF), then clearly the pair

$$(f,p) = \left(f', \frac{p'}{|N|+1}\right)$$

satifies $(|N|+1)^{-1}$–CS with respect to (LNF), and hence $f'$ is optimal for (LNF) by Prop. 4.1. In the scaled algorithm, we seek a 1–CS solution to (SLNF).

Let

$$M = \lfloor \log_2(|N|+1)C \rfloor + 1 = O\big(\log(|N|C)\big), \tag{4.9}$$

where $C = \max_{(i,j)\in A} |a_{ij}|$. In the scaled algorithm, we solve $M$ subproblems. The $m$th subproblem is a minimum cost flow problem, where the cost coefficient of each arc $(i,j)$ is

$$a_{ij}(m) = \text{Trunc}\left(\frac{a'_{ij}}{2^{M-m}}\right), \tag{4.10}$$

where Trunc$(\cdot)$ denotes integer rounding in the direction of 0, that is, down for positive and up for negative numbers. Note that $|a_{ij}(m)|$ is the integer consisting of the $m$ most significant bits in the $M$–bit binary representation of $|a'_{ij}|$. In particular each $a_{ij}(1)$ is 0, $+1$, or $-1$, while $a_{ij}(m+1)$ is obtained by doubling $a_{ij}(m)$ and adding (subtracting) 1 if the $(m+1)$st bit of the $M$–bit representation of $|a_{ij}|$ is a 1 and $a_{ij}$ is positive (negative). Note also that

$$a_{ij}(M) = a'_{ij},$$

so the last problem of the sequence is (SLNF).

All problems in the sequence are solved by applying the $\epsilon$–relaxation algorithm using $\epsilon = 1$, yielding upon termination a pair $\big(f^t(m), p^t(m)\big)$ satisfying 1–CS with respect to the cost coefficients $a_{ij}(m)$. The algorithm is operated in cycles as per Assumption 4.4.

The starting pair $\big(f^0(1), p^0(1)\big)$ for the first problem must be integer and must satisfy 1–CS. The starting price vector for the $(m+1)$st problem $(m = 1, 2, ..., M-1)$ is

$$p^0(m+1) = 2p^t(m), \tag{4.11}$$

where $p^t(m)$ is the final price vector obtained from solution of the $m$th problem. Doubling $p^t(m)$ as above roughly maintains complementary slackness since $a_{ij}(m)$ is roughly

doubled when passing to the $(m+1)$st problem. Indeed, it can be seen that every arc that was 1–balanced (1–active, 1–inactive) upon termination of the algorithm for the $m$th problem will be 3–balanced (1–active, 1–inactive, respectively) at the start of the $(m+1)$st problem.

The starting flow vector $f^0(m+1)$ for the $(m+1)$st problem is obtained from the final flow vector $f^t(m)$ of the preceding problem by setting

$$f_{ij}^0(m+1) = f_{ij}^t(m) \qquad \text{for all balanced arcs } (i,j),$$

$$f_{ij}^0(m+1) = c_{ij} \qquad \text{for all active arcs } (i,j),$$

$$f_{ij}^0(m+1) = b_{ij} \qquad \text{for all inactive arcs } (i,j).$$

(The definitions of balanced, active, and inactive arcs used above are those given in Section 5.2, that is, they correspond to $\epsilon = 0$.) Note that this initialization method implies that the starting price and flow vector will be integer, and that there will be no $1^+$–unblocked and $1^-$–unblocked arcs initially for the $(m+1)$st problem. These facts guarantee that Assumptions 4.2 and 4.3 are satisfied for the subproblems. A more refined initial flow vector choice for the subproblems is given in Exercise 4.1.

Based on Prop. 4.1, the scaled form of the algorithm solves the problem in $O(|N|^3 + |N|^2 B)$ time, where

$$B = \sum_{m=1}^{M} \beta_m \left[p^0(m)\right], \tag{4.12}$$

and $\beta_m(\cdot)$ is defined by Eqs. (4.1)–(4.3) but with the modified cost coefficients $a_{ij}(m)$ replacing $a_{ij}$ in the definition (4.1). We will show that $\beta_m[p^0(m)] = O(|N|)$ for every $m$, thereby obtaining the following proposition:

**Proposition 4.2.** Assume that for the initial subproblem, Assumptions 4.1-4.3 are satisfied and that $p_i^0 - p_j^0 = O(1)$ for all arcs $(i,j)$. The scaled form of the algorithm solves the problem in $O\left(|N|^3 \log(|N|C)\right)$ time, where $C = \max_{(i,j)\in A} |a_{ij}|$.

***Proof.*** Since initially we have $p_i - p_j = O(1)$ and $a_{ij}(1) = O(1)$ for all arcs $(i,j)$, we obtain $d_H\left(p^0(1)\right) = O(|N|)$ for all $H$, and $\beta_1\left[p^0(1)\right] = O(|N|)$. We also have that the final flow vector $f^t(m)$ obtained from the $m$th problem satisfies constraints (1.1) and (1.2), and together with $p^0(m+1)$ it can be seen to satisfy 3–CS. It follows from Lemma 4.1(a) that $\beta_{m+1}\left[p^0(m+1)\right] \leq 3(|N|-1) = O(|N|)$ and the result follows from Eq. (4.12) as discussed above. **Q.E.D.**

### 5.4.2 Application to the Assignment Problem

Consider the special case of the assignment problem shown in Fig. 5.1.1, with the feasible flow range for all arcs taken to be [0,1]. Since all flows generated by the algorithm are integer, it follows that at no Step 2 or 3 of the $\epsilon$–relaxation method, an arc flow

is set strictly between the corresponding upper and the lower bound. Therefore, the computation required for such steps can be eliminated from the accounting of the proof of Prop. 4.1 leading to a time bound $O\big(|A|(|N| + \beta(p^0)/\epsilon)\big)$ for the $\epsilon$–relaxation method and $O\big(|A||N| \log(|N|C)\big)$ for the scaled version.

The auction algorithm described in the previous section can also be shown to have an $O\big(|A|\big(|N| + \beta(p^0)/\epsilon\big)\big)$ time bound in its pure form and an $O\big(|A||N| \log(|N|C)\big)$ bound in its scaled version. This can be done by first modifying the proof of Lemma 4.2 to show that the number of times the profit margin of each person decreases or the price of each object increases is $O\big(|N| + \beta(p^0)/\epsilon\big)$. Then, one uses the argument of the proof of Prop. 4.2 while excluding from the time accounting the computation for steps for which an arc flow is set strictly between the upper and lower bounds. Note here that to achieve the stated time bound, it is still necessary to maintain the arcs of the admissible graph in appropriate lists, one per person and one per object. For an unassigned person $i$, this list consists of the arcs $(i, j)$ for which $\pi_i = \max_{k \in A(i)}\{a_{ik} - p_k\} = a_{ij} - p_j$, and person $i$ can detect at the start of an iteration if its list contains more than one arc. The iterations in which this happens are the ones for which the profit margin $\pi_i$ does not change. In such an iteration, it can be seen that the bid $b_{ij^*}$ of Eq. (3.14) is simply $p_{j^*} + \epsilon$ and can be calculated in $O(1)$ time. With this observation, the complexity analysis of the auction algorithm closely parallels the one of the $\epsilon$–relaxation method (see also [BeE88]). Note, however, that maintaining the lists of arcs of the admissible graph, while useful for theoretical analysis purposes, can actually slow down the algorithm in practice.

Computational experience has shown that the serial scaled version of the auction algorithm performs very well, outperforming some highly efficient implementations of alternative methods. There is also some experience with parallel implementations of the auction algorithm in shared memory machines showing a modest speedup of up to about ten over the serial version of the algorithm. This is probably the maximum speedup that can be expected from a parallel version of the auction algorithm, because near termination there are typically very few persons that are unassigned thereby diminishing the method's potential for concurrency.

We finally mention that it is possible to combine the auction algorithm with the primal–dual method (Exercises 2.4 and 3.8) with the purpose of improving its serial time bound. This method is described in Exercise 4.5 and takes time $O\big(|A||N|^{\frac{1}{2}} \log(|N|C)\big)$. Its practical performance, however, does not appear to be better than the performance of the pure auction algorithm with scaling.

### An Example of Poor Performance of the $\epsilon$–Relaxation Method

In spite of the good theoretical time bound obtained for the scaled $\epsilon$–relaxation method, there are problems where it can perform poorly compared with alternative methods. In particular, there is sometimes a tendency for each node to make a large number of small price rises, and the actual amount of work involved in price rises is then of the same order as its theoretical bound. Figure 5.4.4 presents an example of such behavior. It is an assignment problem with $2n$ nodes, nodes $s_1, ..., s_n$ being persons and $t_1, ..., t_n$ being objects. The arcs are $(s_k, t_k)$ for $k = 1, ..., n$, and $(s_k, t_{k+1})$ for $k = 1, ..., n-1$. All arcs have unit capacity and zero cost. The problem may also be viewed as a max–flow

problem by adjoining a "super source" node $s$ and arcs $(s, s_k)$, along with a "super sink" node $t$ and arcs $(t_k, t)$. Suppose that the (scaled or pure) $\epsilon$–relaxation algorithm is applied to the assignment version of this example, with $\epsilon = 1$, initial node order $1, 2, ..., n$, and the rule that whenever it is possible to push flow away from a node on more than one arc, the one that is uppermost in Fig. 5.4.4 is selected. After the first $n$ price rises, the prices and flows will be as shown in Fig. 5.4.5(a).

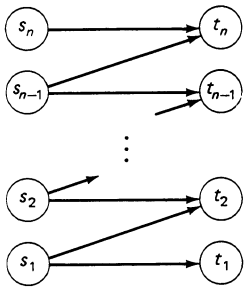



**Figure 5.4.4** An assignment example in which the number of price changes required by the $\epsilon$–relaxation method is proportional to $|N|^2$. Note that the only feasible solution has each $s_k$ assigned to the corresponding $t_k$.

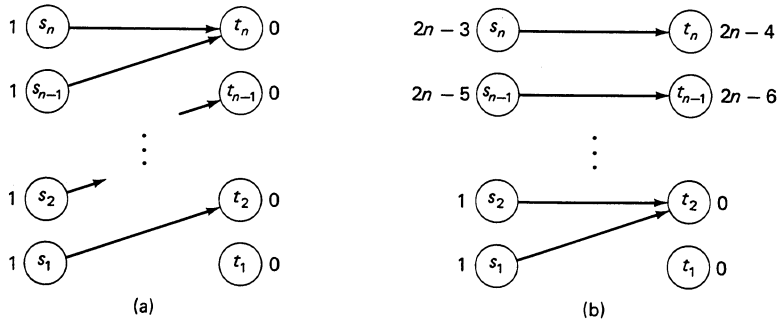



**Figure 5.4.5** (a) The assignment example after $n$ price rises. Prices are shown next to the corresponding node. Only arcs with positive flow are depicted. (b) The intermediate result after $(n-1)^2 + 1$ price rises.

**Claim.** The $\epsilon$–relaxation algorithm as applied to the example of Fig. 5.4.4 requires $n^2$ price rises. The final price of node $s_k$ is $2k - 1$, and that of $t_k$ is $2k - 2$.

***Proof.*** By induction. When $n = 1$, a single price rise at $s_1$ and the ensuing flow adjustment yield a solution in which $s_1$ has price 1, $t_1$ has price 0, and $s_1$ is assigned to $t_1$. This establishes the base case of the induction. Now assume the claim is true for the problem of size $n - 1$; we establish it for the problem of size $n$. After $n$ price rises, the configuration of Fig. 5.4.5(a) will be attained. This leaves nodes $s_2, ..., s_n$ and $t_2, ..., t_n$ in precisely the same state as after $n - 1$ price changes in a problem of size $n - 1$. By induction, after another

$$(n-1)^2 - (n-1) = n^2 - 3n + 2$$

price changes, the algorithm reaches the configuration of Fig. 5.4.5(b). Following the rules of $\epsilon$–relaxation, the reader can confirm that the sequence of nodes now iterated on is $t_2, s_2, t_3, s_3, \ldots, t_n, s_n$, and the promised prices are obtained after $2(n-1)$ further price rises. Following this, the nodes are processed in the opposite order, and a primal feasible solution is obtained in $2n$ additional iterations (but no further price rises). The total number of price changes is

$$n + (n^2 - 3n + 2) + 2(n-1) = n^2.$$

This establishes the induction. **Q.E.D.**

The total number of nodes in the example is $|N| = 2n$; hence the number of price changes is $(|N|/2)^2 = |N|^2/4 = \Omega(|N|^2)$, and increases with $|N|$ at the same rate as its theoretical bound. Since all arc costs are zero, scaling cannot be of any help in this situation. Note also that there is little opportunity for parallelism here, because there is never more than one node with a positive surplus at any time after the first $n$ iterations.

It can be verified that the auction algorithm performs much better than the $\epsilon$–relaxation method for the preceding example, requiring only $O(n)$ iterations in its Gauss–Seidel version (Exercise 4.3).

## EXERCISES

**4.1.** Consider a variation of the initial flow vector choice for the subproblems of the scaled method of Subsection 5.4.1, whereby the flow of each arc $(i, j)$ that belonged to the admissible graph at the end of the previous subproblem, and which is 1–balanced at the start of the current subproblem, is left unchanged, that is, $f_{ij}^0(m+1) = f_{ij}^t(m)$. Show that the estimate of Prop. 4.2 remains unchanged.

**4.2. (Time Bound for the Multiple Node Relaxation Method.)** Consider the multiple node relaxation method of Section 5.2, where iterations start only from nodes with positive surplus.

**(a)** Use an argument similar to the one of Lemma 4.2 to show that the number of iterations that result in a price increase is $O\big(|N|\beta(p^0)\big)$, and, therefore, the total amount of time to solve the problem is $O\big(|N|\beta(p^0)F\big)$, where $F$ is the amount of time for flow augmentations between two successive price increases. [If nodes are scanned in the order they are labeled, it is possible to show that $F = O\big(|N|^5\big)$; see [PaS82] and [Law76]. With a more sophisticated implementation, $F$ can be reduced further considerably.]

**(b)** Show that if cost scaling is used as described in this section, the time estimate becomes $O(|N|^2 F \log C)$.

**4.3.** Apply the Gauss–Seidel version of the auction algorithm to the example of Fig. 5.4.4 and show that it requires $O(n)$ price increases.

**4.4. ($\epsilon$–Scaling.)** Consider the assignment problem. An alternative to the cost scaling procedure given in this section is to apply the auction algorithm for a decreasing sequence $\{\epsilon_k\}$ of values of $\epsilon$, while transferring price and assignment information from one application to

the next. The general form of this procedure is as follows: initially, we replace all arc cost coefficients $a_{ij}$ with $a'_{ij} = (n+1)a_{ij}$, we select a scalar $\theta \in (0,1)$, we set $\epsilon_0 = \max_{(i,j)\in A} |a'_{ij}|$, and for $k = 1, 2, \ldots$ we set $\epsilon_k = \lceil \theta \epsilon_{k-1} \rceil$. We apply the auction algorithm for $k = 0, 1, \ldots, \bar{k}$ with $\epsilon = \epsilon_k$, where $\bar{k}$ is the first integer $k$ for which $\epsilon_k = 1$. (Note here that the admissible graph should be maintained in appropriate lists, as discussed in Subsection 5.4.2.) Let $p^k$ and $S^k$ be the price vector and complete assignment, respectively, obtained at the end of the $k$th application of the algorithm. The starting price vector and assignment for the $(k+1)$st application of the algorithm are $p^k$ and $\left\{(i,j) \in S^k \mid \pi_i^k - \epsilon_{k+1} \leq a_{ij} - p_j^k\right\}$, respectively, where $\pi_i^k$ is the profit margin of $i$ corresponding to $p^k$. The starting price vector and assignment for the initial application of the algorithm must satisfy $\epsilon^0$–CS. Show that the algorithm finds an optimal assignment in $O\big(n|A|\log(nC)\big)$ time.

**4.5. (Hybrid Auction Algorithm with Running Time $O\left(n^{1/2}|A|\log(nC)\right)$ [AhO87].)** This exercise shows how the auction algorithm can be combined with a more traditional primal–dual method to obtain an algorithm with an improved running time bound. The auction algorithm is used to assign the first $n - O\left(n^{1/2}\right)$ persons and the primal–dual method is used to assign the rest. Consider the solution of the assignment problem by the Gauss–Seidel variant of the scaled auction algorithm ($\epsilon = 1$ throughout).

**(a)** Extend Lemma 4.2 to show that in any subproblem of the scaled auction algorithm we have $\sum_{i\in I}(\pi_i^0 - \pi_i) \leq 8\epsilon n$, where $I$ is the set of unassigned persons, $\pi_i^0 = \max_{j\in A(i)}\left\{a_{ij} - p_j^0\right\}$, and $p^0$ is the vector of prices prevailing at the outset of the subproblem.

**(b)** Suppose that at the outset of each subproblem we use a modified Gauss–Seidel auction procedure in which only persons $i$ with profit margins $\pi_i$ greater than or equal to $\pi_i^0 - (8n)^{1/2}\epsilon$ are allowed to place bids. Show that this procedure can be implemented so that at most $(8n)^{1/2}+1$ iterations are performed at each person node $i$, and that it terminates in $O(n^{1/2}|A|)$ time. Furthermore the number of unassigned persons after termination is at most $(8n)^{1/2}$.

**(c)** Assume that there exists some algorithm $X$ which, given an incomplete assignment $S$ and a price vector $p$ obeying $\epsilon$–CS, produces a new pair $(S', p')$ obeying $\epsilon$–CS in $O(|A|)$ time, with $S'$ containing one more assignment than $S$ (Exercise 3.3 indicates how such an algorithm may be constructed). Outline how one would construct an $O\left(n^{1/2}|A|\log(nC)\right)$ assignment algorithm.

## 5.5 NETWORK FLOW PROBLEMS WITH STRICTLY CONVEX COST

We now consider the nonlinear cost version of the problem of the past four sections:

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

subject to

$$\sum_{\{j|(i,j)\in A\}} f_{ij} - \sum_{\{j|(j,i)\in A\}} f_{ji} = s_i, \qquad \forall\, i \in N, \tag{5.1}$$

$$b_{ij} \leq f_{ij} \leq c_{ij}, \qquad \forall\, (i,j) \in A. \tag{5.2}$$

The difference from the linear cost problem (LNF) is that in place of the linear arc cost functions $a_{ij}f_{ij}$, we now have the nonlinear cost functions $a_{ij}(f_{ij})$. We make the following assumption:

**Assumption 5.1.** (*Strict Convexity*) The functions $a_{ij}(\cdot)$, $(i,j) \in A$, are strictly convex real–valued functions, and problem (CNF) has at least one feasible solution.

Assumption 5.1 implies that (CNF) has a unique optimal solution. To see this note that existence of a solution follows from the Weierstrass theorem (Prop. A.8 in Appendix A), since the cost function is continuous (being real–valued and convex; Prop. A.36 in Appendix A), and the constraint set of Eqs. (5.1) and (5.2) is compact. Uniqueness follows from the strict convexity of the cost function [Prop. A.35(g) in Appendix A].

As in the linear cost case, we will develop an unconstrained dual problem involving a dual variable or price for each node. There is a fundamental difference, however, because the strict convexity of the arc cost functions implies differentiability of the dual cost as demonstrated at the end of Appendix C, and as will also be shown shortly. As a result, the relaxation method for the dual problem encounters none of the difficulties with corner points that were addressed in the previous sections.

Problem (CNF) and the analysis outlined above admit considerable extension. One possibility is to eliminate the upper bounds $c_{ij}$ and/or the lower bounds $b_{ij}$, and replace them by growth conditions on the arc cost functions (see the analysis later in this section). A more substantial extension is to allow the cost function to be nonseparable and/or to replace the network conservation of flow constraints (5.1) with more general linear constraints (see Exercise 5.1). Still, as long as the corresponding dual cost is differentiable, the application of relaxation methods is straightforward. The degree of parallelism afforded by these methods, however, depends on the dependency graph associated with the relaxation method for the dual problem (Subsection 1.2.4), and is greatly affected by the structure of the linear constraints.

### The Dual Problem

The formulation of the dual problem for (CNF) is similar to the formulation of Section 5.1 (see also Appendix C). The Lagrangian function is given by

$$\begin{aligned} L(f,p) &= \sum_{(i,j)\in A} a_{ij}(f_{ij}) + \sum_{i\in N} p_i \left( \sum_{\{j|(j,i)\in A\}} f_{ji} - \sum_{\{j|(i,j)\in A\}} f_{ij} + s_i \right) \\ &= \sum_{(i,j)\in A} \bigl( a_{ij}(f_{ij}) + (p_j - p_i) f_{ij} \bigr) + \sum_{i\in N} s_i p_i. \end{aligned} \tag{5.3}$$

The dual function value $q(p)$ at a price vector $p$ is obtained by minimizing $L(f,p)$ over all $f$ satisfying the capacity constraints (5.2). This leads to the dual problem

$$\begin{aligned} &\text{maximize} \quad q(p) \\ &\text{subject to no constraint on } p, \end{aligned} \tag{5.4}$$

with the dual functional $q$ given by

$$q(p) = \min_{\substack{b_{ij} \le f_{ij} \le c_{ij} \\ (i,j) \in A}} L(f,p) = \sum_{(i,j) \in A} q_{ij}(p_i - p_j) + \sum_{i \in N} s_i p_i, \tag{5.5a}$$

where

$$q_{ij}(p_i - p_j) = \min_{b_{ij} \le f_{ij} \le c_{ij}} \{a_{ij}(f_{ij}) - (p_i - p_j) f_{ij}\}. \tag{5.5b}$$

The minimization in the definition (5.5b) of $q_{ij}$ involves a continuous function and a compact constraint set. Therefore, the minimum is attained (Weierstrass' theorem given as Prop. A.8 in Appendix A), and it follows that $q_{ij}$ is real-valued. Note also that $q_{ij}$ is concave as it is the pointwise minimum of linear functions [Prop. A.35(d) in Appendix A]. The form of $q_{ij}$ is illustrated in Fig. 5.5.1.

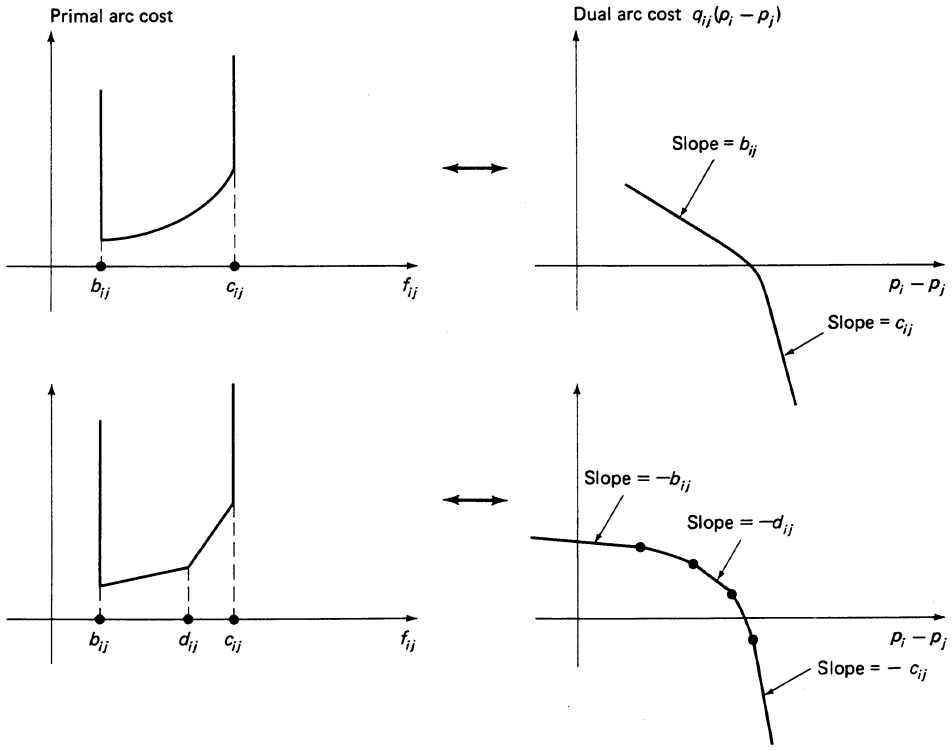



**Figure 5.5.1** Illustration of primal and dual arc cost function pairs. Points where the primal function is nondifferentiable correspond to linear segments of the dual function.

The necessary and sufficient conditions for a pair $(f,p)$ to be primal and dual optimal are given by the Duality Theorem in Appendix C. When specialized to our problem, these conditions are similar to (and indeed generalize) the conditions of Prop.

1.1 for the linear cost problem (LNF). There is the primal feasibility condition that the surplus of each node $i$ should be zero:

$$g_i = \sum_{\{j|(j,i)\in A\}} f_{ji} - \sum_{\{j|(i,j)\in A\}} f_{ij} + s_i = 0, \qquad \forall\, i \in N. \tag{5.6}$$

The role of the cost coefficients $a_{ij}$ of the linear cost problem is played by the right and left derivatives of the arc cost functions defined by

$$a_{ij}^+(f_{ij}) = \lim_{\delta \downarrow 0} \frac{a_{ij}(f_{ij}+\delta) - a_{ij}(f_{ij})}{\delta},$$

$$a_{ij}^-(f_{ij}) = \lim_{\delta \downarrow 0} \frac{a_{ij}(f_{ij}) - a_{ij}(f_{ij}-\delta)}{\delta}.$$

The generalization of the complementary slackness (CS) conditions for a flow–price vector pair $(f,p)$ is that $f_{ij}$ attains the minimum in the definition (5.5b) of $q_{ij}$ for all arcs $(i,j)$ (see the Duality Theorem in Appendix C). Thus, denoting

$$t_{ij} = p_i - p_j,$$

these conditions take the following form, which generalizes the corresponding CS conditions (1.9)–(1.11) for the linear cost problem:

$$t_{ij} \le a_{ij}^+(b_{ij}) \quad \Rightarrow \quad f_{ij} = b_{ij}, \tag{5.7}$$

$$t_{ij} \ge a_{ij}^-(c_{ij}) \quad \Rightarrow \quad f_{ij} = c_{ij}, \tag{5.8}$$

$$a_{ij}^+(b_{ij}) < t_{ij} < a_{ij}^-(c_{ij}) \quad \Rightarrow \quad b_{ij} < f_{ij} < c_{ij}, \quad \text{and} \quad a_{ij}^-(f_{ij}) \le t_{ij} \le a_{ij}^+(f_{ij}), \tag{5.9}$$

as illustrated in Fig. 5.5.2. A key observation is that because of the strict convexity of $a_{ij}$, for each price differential $t_{ij} = p_i - p_j$ there is a *unique* flow $f_{ij}$ attaining the minimum in the definition (5.5b) of $q_{ij}$, and satisfying the CS conditions. It will be shown shortly [Lemma 5.1(d) to follow] that this unique arc flow is equal to minus the derivative of the dual arc cost at $t_{ij}$, that is,

$$f_{ij} \text{ and } t_{ij} \text{ satisfy the CS conditions (5.7)–(5.9)} \iff f_{ij} = -\nabla q_{ij}(t_{ij}). \tag{5.10}$$

This fact will be of central importance in the relaxation algorithm to be described shortly.

By using the Lagrange Multiplier and Duality Theorems of Appendix C, we obtain:

**Proposition 5.1.** Let Assumption 5.1 hold:

**(a)** A flow vector $f$ is primal optimal if and only if $f$ is primal feasible and there exists a price vector $p$ satisfying together with $f$ the CS conditions (5.7)–(5.9).

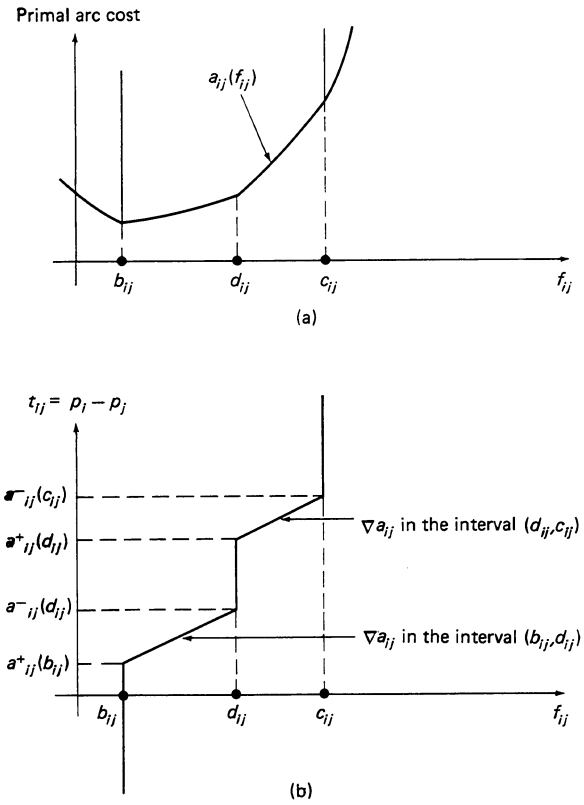



**Figure 5.5.2** Illustration of the CS conditions for the case where $b_{ij} < c_{ij}$. (a) A primal cost function with a single point of nondifferentiability in the interval $(b_{ij}, c_{ij})$. (b) For CS to be satisfied, the pair of $f_{ij}$ and $t_{ij}$ should lie on the diagram. Note that, because $a_{ij}$ is strictly convex, the gradient $\nabla a_{ij}$ is monotonically increasing and there is a unique $f_{ij}$ corresponding to each $t_{ij}$.

**(b)** For every dual optimal solution $p^*$, we have

$$q(p^*) = \sum_{(i,j)\in A} a_{ij}(f^*_{ij}),$$

where $f^* = \{f^*_{ij} \mid (i,j) \in A\}$ is the unique optimal flow vector. Furthermore, there exists at least one dual optimal solution.

**(c)** A flow vector $f$ and a price vector $p$ are primal and dual optimal, respectively, for (CNF) if and only if they satisfy primal feasibility and the CS conditions (5.7)–(5.9).

The proofs of the theorems of Appendix C require differentiability of the primal cost function. For the problem of this section, however, a fairly simple proof that does not require differentiability is possible (see Exercise 5.6).

From the definition of the dual function (5.5), it is seen (using the fact $\sum_{i\in N} s_i = 0$, which is a requirement for feasibility) that if $\{p^*_i \mid i \in N\}$ is an optimal set of prices, the same is true for $\{p^*_i + c \mid i \in N\}$, where $c$ is any constant. Therefore, Prop. 5.1(b) shows that the dual problem has an infinite number of solutions. This implies in

particular that the dual function $q$ is not strictly concave. In fact, it can be seen through simple examples (compare with Fig. 5.5.1) that $q$ need not be strictly concave along any single coordinate direction, so the convergence analysis of the Jacobi and Gauss–Seidel methods for maximizing $q$ (cf. Section 3.2) does not apply.

### The Gradient of the Dual Function

According to the Dual Function Differentiability Theorem of Appendix C, the strict convexity of the primal cost implies differentiability of the dual function $q$. We provide a simple independent proof of this property and we derive the form of the gradient $\nabla q$.

The relation between the primal and dual arc cost functions $a_{ij}$ and $q_{ij}$ is a special case of a conjugacy relation that is central in the theory of convex functions ([Roc70], [Roc84], and [StW70]). More precisely, $-q_{ij}(t_{ij})$ is the *conjugate convex function* of the extended real–valued function $\tilde{a}_{ij}(f_{ij})$ given by

$$\tilde{a}_{ij}(f_{ij}) = \begin{cases} a_{ij}(f_{ij}), & \text{if } b_{ij} \le f_{ij} \le c_{ij} \\ +\infty, & \text{otherwise.} \end{cases} \tag{5.11}$$

Several interesting facts regarding the relation of $\tilde{a}_{ij}$ and $q_{ij}$, including the differentiability of $q_{ij}$, can be obtained by appealing to the theory of conjugate convex functions. To keep the presentation simple, however, we prove only the facts needed for our analysis in the following lemma:

**Lemma 5.1.** Let $a_{ij}(\cdot)$ be a strictly convex real–valued function. Then $a_{ij}(\cdot)$ and the function

$$q_{ij}(t_{ij}) = \min_{b_{ij} \le f_{ij} \le c_{ij}} \{a_{ij}(f_{ij}) - t_{ij} f_{ij}\} \tag{5.12}$$

[cf. Eq. (5.5b)] are related by

$$\sup_{t_{ij}} \{q_{ij}(t_{ij}) + t_{ij} f_{ij}\} = \begin{cases} a_{ij}(f_{ij}), & \text{if } b_{ij} \le f_{ij} \le c_{ij} \\ +\infty, & \text{otherwise.} \end{cases} \tag{5.13}$$

Furthermore, $q_{ij}$ is differentiable, and the following statements are equivalent for two scalars $t_{ij}$ and $f_{ij} \in [b_{ij}, c_{ij}]$:

**(a)** $t_{ij} f_{ij} = a_{ij}(f_{ij}) - q_{ij}(t_{ij})$.
**(b)** $f_{ij}$ attains the minimum in Eq. (5.12).
**(c)** $t_{ij}$ attains the supremum in Eq. (5.13).
**(d)** $\nabla q_{ij}(t_{ij}) = -f_{ij}$.
**(e)** $f_{ij}$ and $t_{ij}$ satisfy the CS conditions (5.7)–(5.9).

***Proof.*** Figure 5.5.3 sketches a proof of Eq. (5.13); it is left for the reader to complete the details. From Eq. (5.12) we see that (b) is equivalent with (a). Similarly, from

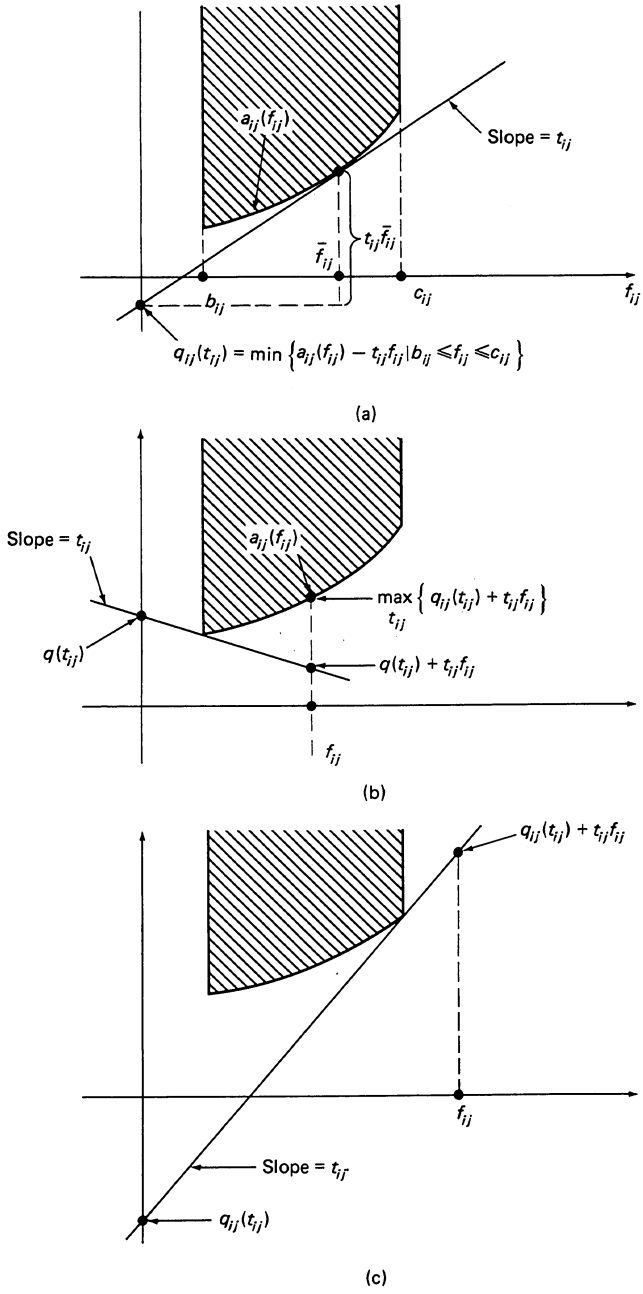



**Figure 5.5.3** Sketch of the proof that $\sup_{t_{ij}}\{q_{ij}(t_{ij}) + t_{ij}f_{ij}\}$ is equal to $a_{ij}(f_{ij})$ if $b_{ij} \leq f_{ij} \leq c_{ij}$ and is equal to $\infty$ otherwise [cf. Eq. (5.13)]:
(a) For any $t_{ij}$, the value of $q_{ij}(t_{ij})$ is obtained by constructing a supporting line with slope $t_{ij}$ to the shaded convex set

$$\{(f_{ij}, \mu) \mid b_{ij} \leq f_{ij} \leq c_{ij},\ \mu \geq a_{ij}(f_{ij})\},$$

and by obtaining the point where this line intercepts the vertical axis.
(b) For a given $f_{ij} \in [b_{ij}, c_{ij}]$, the value of $q_{ij}(t_{ij}) + t_{ij}f_{ij}$ is obtained by intercepting the vertical line passing through $\big(f_{ij}, a_{ij}(f_{ij})\big)$ with the line of slope $t_{ij}$ that supports the shaded set. This point of intercept cannot lie higher than $a_{ij}(f_{ij})$, and with proper choice of $t_{ij}$ lies exactly at $a_{ij}(f_{ij})$. This proves that $\sup_{t_{ij}}\{q_{ij}(t_{ij}) + t_{ij}f_{ij}\} = a_{ij}(f_{ij})$ for $f_{ij} \in [b_{ij}, c_{ij}]$.
(c) For $f_{ij} \notin [b_{ij}, c_{ij}]$, the construction given shows that $\sup_{t_{ij}}\{q_{ij}(t_{ij}) + t_{ij}f_{ij}\} = \infty$.

Eq. (5.13) we see that (c) is equivalent with (a). Therefore, (b) and (c) are equivalent. To prove differentiability of $q_{ij}$, let us fix $t_{ij}$, and let $q_{ij}^{+}(t_{ij})$ and $q_{ij}^{-}(t_{ij})$ be the right and left directional derivatives of $q_{ij}$, respectively, at $t_{ij}$. A scalar $y$ satisfies

$$q_{ij}^+(t_{ij}) \leq y \leq q_{ij}^-(t_{ij}), \tag{5.14}$$

where $q_{ij}^+$ and $q_{ij}^-$ are the right and left derivatives, respectively, of $q_{ij}$, if and only if $t_{ij}$ maximizes $q_{ij}(\xi) - \xi y$ over all $\xi$, which is true [by the equivalence of (b) and (c)] if and only if $-y$ attains the minimum in Eq. (5.12). Since $a_{ij}$ is strictly convex there is only one minimizing scalar in Eq. (5.12), call it $f_{ij}$, and $-f_{ij}$ is the only scalar $y$ satisfying Eq. (5.14). Therefore, $q_{ij}$ is differentiable, and (b) implies (d). Conversely, if $\nabla q_{ij}(t_{ij}) = -f_{ij}$, then $t_{ij}$ attains the supremum in Eq. (5.13), and [since (c) implies (b)] $f_{ij}$ attains the minimum in Eq. (5.12). The equivalence of (b) and (d) follows. Finally, the CS conditions (5.7)–(5.9) are equivalent to (b), so the proof is complete. **Q.E.D.**

We can now derive the gradient of the dual cost given a price vector $p$. We have for all $i \in N$

$$\begin{aligned} \frac{\partial q(p)}{\partial p_i} &= \sum_{(m,n)\in A} \frac{\partial q_{mn}(p_m - p_n)}{\partial p_i} + s_i \\ &= -\sum_{\{j|(j,i)\in A\}} \nabla q_{ji}(p_j - p_i) + \sum_{\{j|(i,j)\in A\}} \nabla q_{ij}(p_i - p_j) + s_i. \end{aligned} \tag{5.15}$$

By Lemma 5.1, the derivatives above are equal to minus the unique arc flows satisfying the CS conditions (5.7)–(5.9) together with $p$. Therefore, comparing the preceding relation with the definition of surplus (5.6), we obtain

$$\frac{\partial q(p)}{\partial p_i} = g_i(p) \tag{5.16a}$$

where

$$\begin{aligned} g_i(p) = {} & \text{Surplus of node } i \text{ corresponding to the unique } f \\ & \text{satisfying the CS conditions (5.7)–(5.9) together with } p. \end{aligned} \tag{5.16b}$$

### 5.5.1 The Relaxation Method

The relaxation method is simply the coordinate ascent (or nonlinear Gauss–Seidel) method of Subsection 3.2.4 applied to the maximization of the dual function. We generalize the method somewhat by allowing the maximization along each coordinate to be inexact to some extent, and to be controlled by a given scalar $\delta \in [0, 1)$.

At the start of the typical iteration we have a price vector $p$. If the corresponding surplus $g_i(p)$ of Eq. (5.16) is zero for all nodes $i$, then $p$ and the unique vector $f$ satisfying CS together with $p$ are dual and primal optimal, respectively, and the algorithm terminates. Otherwise:

**Relaxation Iteration.** Choose any node $i$. If the surplus $g_i(p)$ is zero do nothing. Otherwise change the $i$th coordinate of $p$, obtaining a vector $\bar{p}$ which is such that

$$0 \leq g_i(\bar{p}) \leq \delta g_i(p) \qquad \text{if } g_i(p) > 0,$$
$$\delta g_i(p) \leq g_i(\bar{p}) \leq 0 \qquad \text{if } g_i(p) < 0.$$

Figure 5.5.4 illustrates the relaxation iteration when $\delta = 0$, in which case we have $g_i(\bar{p}) = \partial q(\bar{p})/\partial p_i = 0$, and the coordinate maximization is exact [cf. Eq. (5.16)]. There is a great deal of flexibility regarding the order in which nodes are taken up for relaxation. The only assumption we make is the following:

**Assumption 5.2.** Every node is chosen as the node $i$ in the relaxation iteration an infinite number of times.

The relaxation iteration is well defined, in the sense that it is always possible to adjust the price $p_i$ as required. To see this, suppose that $g_i(p) > 0$ and that there does not exist a $\gamma > 0$ such that $g_i(p + \gamma e_i) \leq \delta g_i(p)$, where $e_i$ denotes the $i$th coordinate vector. Consider the price differentials $t_{ij}(\gamma)$, $(i,j) \in A$ and $t_{ji}(\gamma)$, $(j,i) \in A$ corresponding to the price vector $p + \gamma e_i$. We have $t_{ij}(\gamma) = p_i - p_j + \gamma \to \infty$ and $t_{ji}(\gamma) = p_j - p_i - \gamma \to -\infty$ as $\gamma \to \infty$. Therefore, the corresponding unique arc flows satisfying the CS conditions (5.7)–(5.9) become $f_{ij} = c_{ij}$ and $f_{ji} = b_{ji}$ as $\gamma \to \infty$, and using the definition (5.16) of $g_i(\cdot)$, it is seen that

$$\lim_{\gamma \to \infty} g_i(p + \gamma e_i) = \sum_{\{j \mid (j,i) \in A\}} b_{ji} - \sum_{\{j \mid (i,j) \in A\}} c_{ij} + s_i \geq \delta g_i(p) > 0.$$

This implies that the surplus of node $i$ is positive for any flow vector $f$ satisfying the capacity constraints (5.2), and contradicts the existence of a feasible flow (Assumption 5.1). An analogous argument can be made for the case where $g_i(p) < 0$.

It is evident that the relaxation algorithm admits synchronous parallel Gauss–Seidel or Jacobi implementations similar to the $\epsilon$–relaxation method of Section 5.3. We now show that the Gauss–Seidel version of the algorithm is convergent. Simple examples show that the Jacobi version is not convergent (see Fig. 5.5.5), but in the next two chapters, it will be seen that when suitably modified, the algorithm is convergent under reasonable conditions when implemented in either the Jacobi or an asynchronous mode.

### 5.5.2 Convergence Analysis

In order to obtain our convergence result, we must show that the sequence of flow vectors generated by the relaxation algorithm approaches the linear manifold defined by the conservation of flow constraint (5.1). The line of argument that we will use is as follows: we will bound from below the improvement in the dual functional $q$ per iteration by a positive quantity. We will then show that if the sequence of flow vectors does not approach the constraint manifold, the quantity itself can be lower bounded by

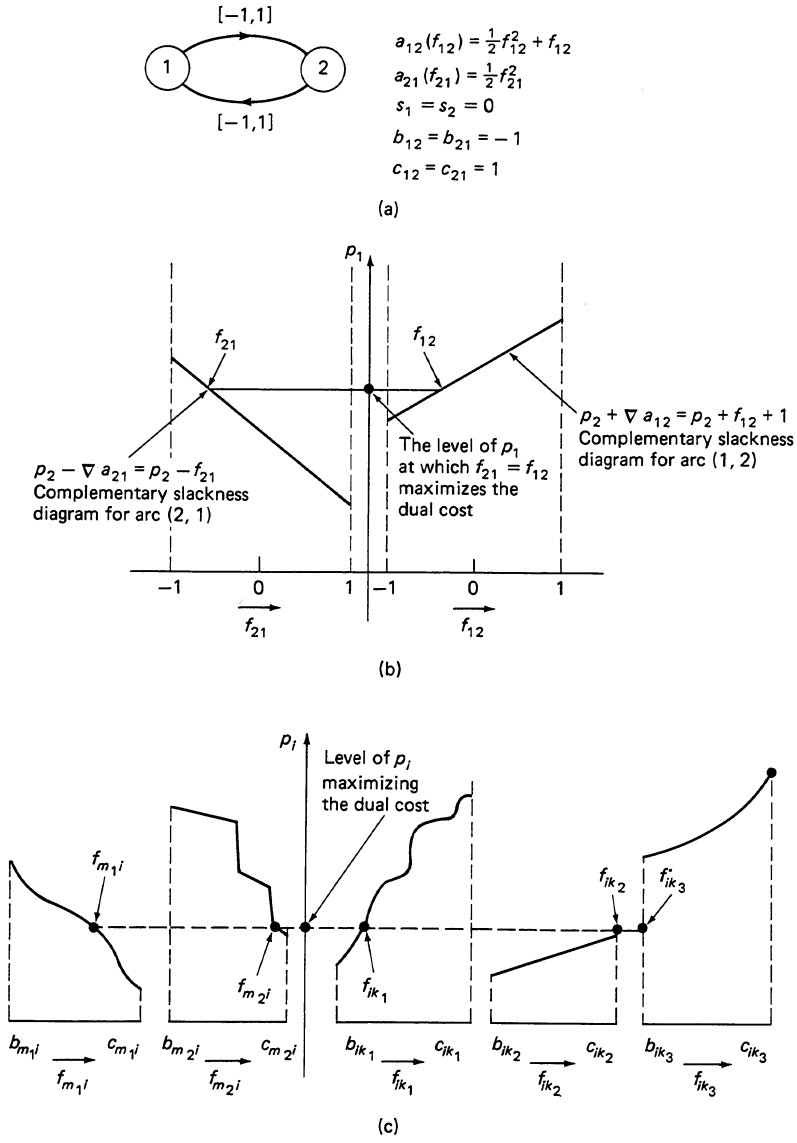



**Figure 5.5.4** Illustration of the relaxation iteration. (a) An example problem involving two nodes with zero supplies, and two arcs with the arc costs and flow bounds shown. (b) Adjustment of the price $p_1$ so as to maximize the dual function along the first coordinate. The CS diagrams for arcs (1,2) and (2,1) are superimposed, and $p_1$ is set at the level at which the flows $f_{12}$ and $f_{21}$ obtained from these diagrams are equalized. (c) Illustration of the same process for the case where the node $i$ chosen for relaxation has more than two incident arcs. The price $p_i$ is set at the level at which the sum of outgoing arc flows (obtained from the CS diagrams) equals the sum of the corresponding incoming arc flows plus $s_i$.

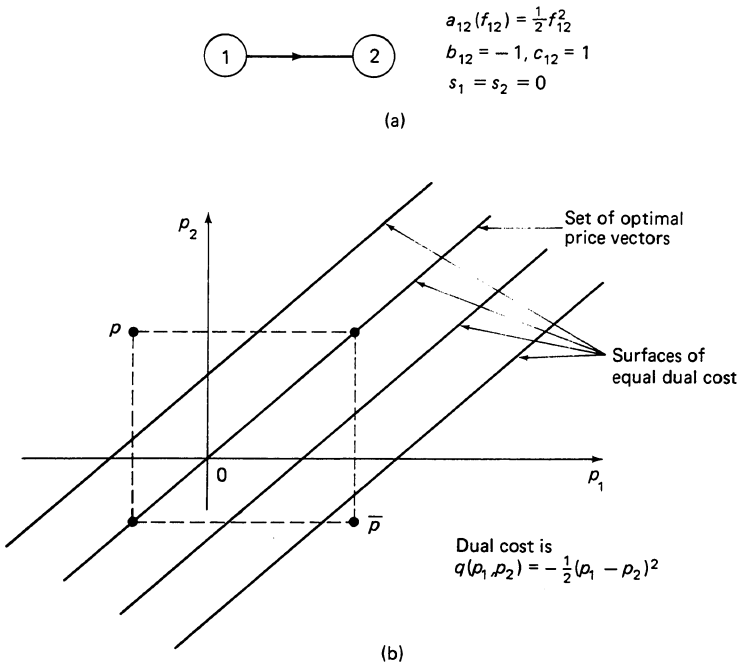



**Figure 5.5.5** (a) An example problem for which the Jacobi version of the relaxation algorithm does not converge to an optimal price vector. (b) The Jacobi version of the relaxation method (with exact minimization along each price coordinate), starting from $p$ yields $\bar{p}$, and starting from $\bar{p}$ yields $p$. By contrast, the Gauss–Seidel version finds an optimal price vector in a single iteration (assuming again exact maximization along each price coordinate).

a positive constant, which implies that the optimal dual cost is $\infty$. This will contradict the finiteness of the optimal primal cost (Prop. 5.1 and Assumption 5.1). We will denote the price vector generated at the $k$th iteration by $p^k$, $k = 0, 1, \ldots$, and the node operated on at the $k$th iteration by $i^k$, $k = 0, 1, \ldots$. We will also use the notation

$$t_{ij}^k = p_i^k - p_j^k, \qquad f_{ij}^k = -\nabla q_{ij}(p^k).$$

Note that by Lemma 5.1, $f_{ij}^k$ is the unique flow satisfying, together with $t_{ij}^k = p_i^k - p_j^k$ the CS conditions (5.7)–(5.9). For any directed cycle $Y$ of the network, we denote

$$Y^+ = \{(i,j) \mid (i,j) \text{ is a forward arc of } Y\},$$
$$Y^- = \{(i,j) \mid (i,j) \text{ is a backward arc of } Y\}.$$

We also denote [cf. Eq. (5.11)]

$$\tilde{a}_{ij}^{+}(f_{ij}) = \begin{cases} a_{ij}^{+}(f_{ij}), & \text{if } b_{ij} \le f_{ij} < c_{ij} \\ +\infty, & \text{if } f_{ij} = c_{ij}, \end{cases} \tag{5.17a}$$

$$\tilde{a}_{ij}^{-}(f_{ij}) = \begin{cases} a_{ij}^{-}(f_{ij}), & \text{if } b_{ij} < f_{ij} \le c_{ij} \\ -\infty, & \text{if } b_{ij} = f_{ij}, \end{cases} \tag{5.17b}$$

that is, $\tilde{a}_{ij}^{-}$ and $\tilde{a}_{ij}^{+}$ denote the left and right derivatives of the function $\tilde{a}_{ij}$ of Eq. (5.11). Note that the conditions for $f_{ij}$ and $t_{ij}$ to satisfy CS can be written as

$$b_{ij} \le f_{ij} \le c_{ij}, \qquad \tilde{a}_{ij}^{-}(f_{ij}) \le t_{ij} \le \tilde{a}_{ij}^{+}(f_{ij}). \tag{5.18}$$

We first show a few preliminary results:

**Lemma 5.2.** We have for all $k$ such that $p^{k+1} \neq p^k$ [i.e., $g_{i^k}(p^k) \neq 0$]

$$q(p^{k+1}) - q(p^k) \ge \sum_{(i,j)\in A} \left[a_{ij}(f_{ij}^{k+1}) - a_{ij}(f_{ij}^{k}) - (f_{ij}^{k+1} - f_{ij}^{k})t_{ij}^{k}\right] > 0, \tag{5.19}$$

with equality holding in the left inequality if $g_{i^k}(p^{k+1}) = 0$ (i.e., the exact maximum of the dual cost along the $i^k$th coordinate is found at the $k$th iteration).

***Proof.*** Fix an index $k \ge 0$. Denote $\gamma = p_{i^k}^{k+1} - p_{i^k}^{k}$. From the definition of the dual function (5.5), Lemma 5.1, and the definitions of $t_{ij}^k$ and $f_{ij}^k$, we have

$$q(p^k) = \sum_{(i,j)\in A} \left[a_{ij}(f_{ij}^{k}) - t_{ij}^{k} f_{ij}^{k}\right] + \sum_{i\in N} s_i p_i^k, \qquad \forall\, k \ge 0.$$

Therefore, denoting $e_{ij} = 1$ if $i = i^k$, $e_{ij} = -1$ if $j = i^k$, and $e_{ij} = 0$ otherwise, we have

$$\begin{aligned} q(p^{k+1}) - q(p^k) &= \sum_{(i,j)\in A} \left[a_{ij}(f_{ij}^{k+1}) - t_{ij}^{k+1} f_{ij}^{k+1}\right] - \sum_{(i,j)\in A} \left[a_{ij}(f_{ij}^{k}) - t_{ij}^{k} f_{ij}^{k}\right] \\ &\qquad + \sum_{i\in N} \left(p_i^{k+1} - p_i^k\right) s_i \\ &= \sum_{(i,j)\in A} \left[a_{ij}(f_{ij}^{k+1}) - (t_{ij}^{k} + e_{ij}\gamma) f_{ij}^{k+1}\right] - \sum_{(i,j)\in A} \left[a_{ij}(f_{ij}^{k}) - t_{ij}^{k} f_{ij}^{k}\right] + \gamma s_{i^k} \\ &= \sum_{(i,j)\in A} \left[a_{ij}(f_{ij}^{k+1}) - a_{ij}(f_{ij}^{k}) - (f_{ij}^{k+1} - f_{ij}^{k})t_{ij}^{k}\right] + \gamma \left(s_{i^k} - \sum_{(i,j)\in A} e_{ij} f_{ij}^{k+1}\right) \\ &= \sum_{(i,j)\in A} \left[a_{ij}(f_{ij}^{k+1}) - a_{ij}(f_{ij}^{k}) - (f_{ij}^{k+1} - f_{ij}^{k})t_{ij}^{k}\right] + \gamma g_{i^k}(p^{k+1}). \end{aligned}$$

When $\gamma > 0$ (that is, $p_{i^k}^k$ is increased), the new price $p_{i^k}^{k+1}$ of $i^k$ either maximizes the cost with respect to $p_{i^k}$, in which case $g_{i^k}(p^{k+1}) = 0$ or $p_{i^k}^{k+1}$ is smaller than all prices that maximize the dual cost, in which case $g_{i^k}(p^{k+1}) > 0$. Thus, we have $\gamma g_{i^k}(p^{k+1}) \geq 0$ when $\gamma > 0$. A similar argument shows that $\gamma g_{i^k}(p^{k+1}) \geq 0$ when $\gamma < 0$, and from the above equation, we obtain the left side of Eq. (5.19). The right side of Eq. (5.19) follows from the strict convexity of $a_{ij}$, the fact $f^{k+1} \neq f^k$ (since the surplus of $i^k$ has changed), and the fact that $f_{ij}^k$, by Lemma 5.1, minimizes $a_{ij}(f_{ij}) - t_{ij}^k f_{ij}$ over $f_{ij} \in [b_{ij}, c_{ij}]$. **Q.E.D.**

The next result is remarkable in that it shows that under a mild restriction on the way the relaxation iteration is carried out (which is typically very easy to satisfy in practice), the sequence of price vectors approaches the dual optimal set in an unusual manner. In particular, the distance of the current iterate from *any* optimal price vector with respect to the maximum norm never increases.

**Lemma 5.3.** Let $p$ be a price vector, $i$ be a node, and $\bar{p}$ be a dual price vector obtained by applying the relaxation iteration to $p$ using node $i$. Assume in addition that $\bar{p}$ is chosen so that

$$g_i(p) > 0 \quad \Rightarrow \quad g_i\big(\bar{p} + \gamma(p - \bar{p})\big) > 0, \qquad \forall\, \gamma > 0, \tag{5.20a}$$

$$g_i(p) < 0 \quad \Rightarrow \quad g_i\big(\bar{p} + \gamma(p - \bar{p})\big) < 0, \qquad \forall\, \gamma > 0. \tag{5.20b}$$

Then for all optimal dual price vectors $p^*$, we have

$$\min_{m \in N}\{p_m - p_m^*\} \leq \min_{m \in N}\{\bar{p}_m - p_m^*\} \leq \max_{m \in N}\{\bar{p}_m - p_m^*\} \leq \max_{m \in N}\{p_m - p_m^*\}. \tag{5.21}$$

*Note:* When $g_i(p) > 0$ [or $g_i(p) < 0$], the assumption (5.20) is equivalent to assuming that $\bar{p}_i$ is chosen less (greater) than or equal to the smallest (largest) maximizing point of the dual cost along the $i$th coordinate starting from $p$. It is automatically satisfied if there is a unique maximizing point along the $i$th coordinate.

***Proof.*** The desired assertion (5.21) holds if $g_i(p) = 0$ since then we have $\bar{p} = p$. Assume that $g_i(p) > 0$, and fix an optimal dual price vector $p^*$. Consider the vector $\hat{p}$ defined by

$$\hat{p}_j = \begin{cases} p_j, & \text{if } j \neq i \\ p_i^* + \max_{m \in N}\{p_m - p_m^*\}, & \text{if } j = i. \end{cases}$$

We have

$$\hat{p}_i - \hat{p}_m \geq p_i^* - p_m^*, \qquad \forall\, (i, m) \in A,$$

$$\hat{p}_m - \hat{p}_i \leq p_m^* - p_i^*, \qquad \forall\, (m, i) \in A.$$

Since each $\nabla q_{ij}$ is a nonincreasing function (being the gradient of a concave function), we have

$$\nabla q_{im}(\hat{p}_i - \hat{p}_m) \leq \nabla q_{im}(p_i^* - p_m^*), \qquad \forall\ (i,m) \in A,$$
$$\nabla q_{mi}(\hat{p}_m - \hat{p}_i) \geq \nabla q_{mi}(p_m^* - p_i^*), \qquad \forall\ (m,i) \in A.$$

By adding over all $m$ and by using Eqs. (5.15) and (5.16), we obtain $g_i(\hat{p}) \leq g_i(p^*) = 0$. Therefore, using assumption (5.20), we have $\bar{p}_i \leq \hat{p}_i$, while at the same time $p_i < \bar{p}_i$, and $p_m = \bar{p}_m$ for all $m \neq i$. The assertion (5.21) follows. The proof is similar when $g_i(p) < 0$. **Q.E.D.**

Lemma 5.3 implies, among other things, that if care is taken so that the condition (5.20) is satisfied at all iterations, the sequence $\{p^k\}$ generated by the relaxation method is bounded. Furthermore, the lemma shows that if $\{p^k\}$ has a limit point that is an optimal price vector, then $\{p^k\}$ must converge to that vector. We are now ready to show our main result.

**Proposition 5.2.** Let $\{p^k, f^k\}$ be a sequence generated by the relaxation method for strictly convex arc costs. Then:

**(a)**
$$\lim_{k\to\infty} g_i(p^k) = 0, \qquad \forall\ i \in N. \tag{5.22}$$

**(b)**
$$\lim_{k\to\infty} f^k = f^*, \tag{5.23}$$

where $f^*$ is the unique optimal flow vector.

**(c)**
$$\lim_{k\to\infty} q(p^k) = \max_p q(p).$$

**(d)** If each iteration is carried out so that the condition (5.20) is satisfied, then

$$\lim_{k\to\infty} p^k \to p^*, \tag{5.24}$$

where $p^*$ is some optimal price vector.

***Proof.***

**(a)** We first show that

$$\lim_{k\to\infty} g_{i^k}(p^k) = 0. \tag{5.25}$$

Indeed, if this were not so, there must exist an $\epsilon > 0$ and a subsequence $K$ such that $|g_{i^k}(p^k)| \geq \epsilon$ for all $k \in K$. Without loss of generality, we assume that $g_{i^k}(p^k) \geq \epsilon$ for all $k \in K$. Since $\delta|g_{i^k}(p^k)| \geq |g_{i^k}(p^{k+1})|$, we have that at the $k$th iteration, some arc incident to node $i^k$ must change its flow by at least $\Delta$, where $\Delta = (1-\delta)\epsilon/|A|$. By passing to a subsequence, if necessary, we assume that this happens for the same

arc, say $(i, j)$, for all $k \in K$, and that $f_{ij}^{k+1} - f_{ij}^k \geq \Delta$, and $i^k = i$ for all $k \in K$ (the case $f_{ij}^{k+1} - f_{ij}^k < \Delta$ and $i^k = j$ for all $k \in K$ can be treated analogously). Using the boundedness of $\{f^k\}$ we can also assume that the subsequence $\{f_{ij}^k\}_{k \in K}$ converges to some $f_{ij}$. We note that all terms $a_{ij}(f_{ij}^{k+1}) - a_{ij}(f_{ij}^k) - (f_{ij}^{k+1} - f_{ij}^k)t_{ij}^k$ in the sum of Eq. (5.19) in Lemma 5.2 are nonnegative. Therefore, we have from Eq. (5.19)

$$\begin{aligned} q(p^{k+1}) - q(p^k) &\geq a_{ij}(f_{ij}^{k+1}) - a_{ij}(f_{ij}^k) - (f_{ij}^{k+1} - f_{ij}^k)t_{ij}^k \\ &\geq a_{ij}(f_{ij}^k + \Delta) - a_{ij}(f_{ij}^k) - \Delta t_{ij}^k \\ &\geq a_{ij}(f_{ij}^k + \Delta) - a_{ij}(f_{ij}^k) - \Delta a_{ij}^+(f_{ij}^k), \end{aligned}$$

where the second inequality follows from the fact that $a_{ij}(f_{ij}^k + \delta) - a_{ij}(f_{ij}^k) - \delta t_{ij}^k$ is a monotonically increasing function of $\delta$ (since $t_{ij}^k$ is a subgradient of $a_{ij}$ at $f_{ij}^k$), and the last inequality follows from the fact $\Delta > 0$ (implying $f_{ij}^k < c_{ij}$) and the CS condition (5.18). Taking the limit as $k \to \infty$, $k \in K$, and using the facts $f_{ij}^k \to f_{ij}$ and $\liminf_{k \to \infty} a_{ij}^+(f_{ij}^k) \leq a_{ij}^+(f_{ij})$ (in view of the fact that $a_{ij}^+$ is monotonically nondecreasing and right–continuous), we obtain

$$\liminf_{k \in K, k \to \infty} \left[q(p^{k+1}) - q(p^k)\right] \geq a_{ij}(f_{ij} + \Delta) - a_{ij}(f_{ij}) - \Delta a_{ij}^+(f_{ij}) > 0.$$

This implies that $\lim_{k \to \infty} q(p^k) = \infty$, contradicting the equality of the optimal dual value and the optimal primal value (which is finite), and proving Eq. (5.25).

We now show that $\lim_{k \to \infty} g_i(p^k) = 0$ [cf. Eq. (5.22)]. Choose any $i \in N$. Take any $\epsilon > 0$ and let $K$ be the set of indices $k$ such that $g_i(p^k) > 2\epsilon$. Assume without loss of generality that $g_i(p^k) < \epsilon$ for all $k$ with $i = i^k$ [cf. Eq. (5.25)]. For every $k \in K$, let $k'$ be the first index with $k' > k$ such that $i = i^{k'}$. Then during iterations $k, k+1, \ldots, k'-1$ node $i$ is not chosen for relaxation while its surplus decreases from greater than $2\epsilon$ to lower than $\epsilon$. We claim that during these iterations the total absolute surplus $\sum_{j \in N} |g_j(p)|$ is decreased by a total of more than $2\epsilon$. To see this, we first note that the total absolute surplus cannot increase due to an iteration because a flow change on an arc reflects itself in a change of the surplus of its start node and an opposite change in the surplus of its end node; furthermore, the surplus of the node chosen for relaxation at an iteration cannot increase in absolute value or change sign during that iteration. Next observe that for any of the iterations $k, k+1, \ldots, k'-1$, say $\bar{k}$, for which the surplus of $i$ is decreased by an amount $\xi > 0$ from a positive value $g_i(p^{\bar{k}}) > 0$, the node $s = i^{\bar{k}}$ chosen for relaxation must be adjacent to $i$ and must have a negative surplus $g_s(p^{\bar{k}}) < 0$. Since all of the increase in $g_s(p^{\bar{k}})$ during the iteration must be matched by decreases of the surpluses of the neighbor nodes of $s$, and, furthermore, the surplus of $s$ will remain nonpositive after the iteration, it follows that the total absolute surplus will be decreased by at least $2\min\{\xi,\ g_i(p^{\bar{k}})\}$ during the iteration. This shows that during iterations $k, k+1, \ldots, k'-1$, the total absolute surplus must decrease by more than $2\epsilon$. Therefore, the set $K$ of indices $k$ for which $g_i(p^k) > 2\epsilon$ cannot be infinite.

Since $\epsilon > 0$ is arbitrary, we obtain $\limsup_{k\to\infty} g_i(p^k) \leq 0$. Similarly, we can show that $\liminf_{k\to\infty} g_i(p^k) \geq 0$ and therefore $g_i(p^k) \to 0$.

**(b)–(c)** Since $\{f_{ij}^k\}$ is bounded, it has a subsequence converging to some $f^*$ which is primal feasible in view of part (a). We will show that $f^*$ is the unique optimal flow vector, and, therefore, the entire sequence $\{f_{ij}^k\}$ actually converges to $f^*$. For every arc $(i,j)$ for which $b_{ij} < c_{ij}$, there are three possibilities for the corresponding subsequence of $\{t_{ij}^k\}$:

**(1)** $\{t_{ij}^k\}$ is bounded.

**(2)** $f_{ij}^* = c_{ij}$, and $-\infty < \liminf_{k\to\infty} t_{ij}^k \leq \limsup_{k\to\infty} t_{ij}^k = \infty$.

**(3)** $f_{ij}^* = b_{ij}$, and $-\infty = \liminf_{k\to\infty} t_{ij}^k \leq \limsup_{k\to\infty} t_{ij}^k < \infty$.

For an arc $(i,j)$ with $b_{ij} = c_{ij}$ we must have $f_{ij}^* = f_{ij}^k$ for all $k$. From these facts, it is seen that we can construct a subsequence $K$ such that

$$\sum_{(i,j)\in A} t_{ij}^k (f_{ij}^k - f_{ij}^*) \leq \sum_{(i,j)\in B} t_{ij}^k (f_{ij}^k - f_{ij}^*), \qquad \forall\, k \in K,$$

where $B$ is a set of arcs $(i,j)$ such that $\{t_{ij}^k\}_K$ is bounded. We have, using the definition of the dual functional (5.5), Lemma 5.1 and the fact $\sum_{(i,j)\in A} t_{ij}^k f_{ij}^* = \sum_{i\in N} p_i s_i$,

$$\begin{aligned}\sum_{(i,j)\in A} a_{ij}(f_{ij}^k) - q(p^k) &= \sum_{(i,j)\in A} t_{ij}^k f_{ij}^k - \sum_{i\in N} p_i s_i \\ &= \sum_{(i,j)\in A} t_{ij}^k (f_{ij}^k - f_{ij}^*) \leq \sum_{(i,j)\in B} t_{ij}^k (f_{ij}^k - f_{ij}^*).\end{aligned}$$

Since $f_{ij}^k \to f_{ij}^*$ and $\{t_{ij}^k\}_K$ is bounded for $(i,j) \in B$, we obtain by taking the limit above

$$\sum_{(i,j)\in A} a_{ij}(f_{ij}^*) \leq \lim_{k\to\infty,\ k\in K} q(p^k).$$

Since $f^*$ is primal feasible and $q(p)$ is less than or equal to the optimal primal cost, the optimality of $f^*$ follows. Since $q(p^k)$ is monotonically nondecreasing, and therefore converges, it also follows from the relation above that it converges to the dual optimal value.

**(d)** Under the condition (5.20), Lemma 5.3 implies that $\{p^k\}$ is bounded. Let $\{p^k\}_{k\in K}$ be a subsequence converging to a vector $p^*$, and let $t^*$ be the vector with elements $t_{ij}^* = p_i^* - p_j^*$. We have for all $(i,j) \in A$,

$$\tilde{a}_{ij}^-(f_{ij}^k) \leq t_{ij}^k \leq \tilde{a}_{ij}^+(f_{ij}^k), \qquad \forall\, k \in K.$$

It follows, using part (b) and the fact that $\tilde{a}_{ij}^-$ (or $\tilde{a}_{ij}^+$) is nondecreasing and left continuous (or nondecreasing and right–continuous, respectively) (Prop. A.38 in Appendix A), that for all $(i,j) \in A$,

$$\tilde{a}_{ij}^-(f_{ij}^*) \leq t_{ij}^* \leq \tilde{a}_{ij}^+(f_{ij}^*),$$

where $f^*$ is the optimal flow vector. Therefore, $t^*$ satisfies together with $f^*$ the CS condition (5.18) for all $(i,j) \in A$, and must be dual optimal. Lemma 5.3 shows that $\{p^k\}$ cannot have two different dual optimal price vectors as limit points and the conclusion follows. **Q.E.D.**

### 5.5.3 The Problem without Arc Flow Bounds

Consider the variation of the convex cost network flow problem (CNF), where there are no arc flow bounds, that is, the problem

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

$$\text{subject to} \quad \sum_{\{j|(j,i)\in A\}} f_{ji} - \sum_{\{j|(i,j)\in A\}} f_{ij} = s_i, \qquad \forall\, i \in N. \qquad (5.26)$$

In place of Assumption 5.1, we assume that the arc cost functions $a_{ij}$ grow at an infinite rate as $f_{ij}$ approaches the boundaries of the domain of $a_{ij}$ (see Fig. 5.5.6):

**Assumption 5.2.** For each arc $(i,j)$, there is a nonempty open interval $(b_{ij}, c_{ij})$, where $b_{ij} \in [-\infty, \infty)$ and $c_{ij} \in (-\infty, \infty]$, and such that the real–valued function $a_{ij}$ is defined and is strictly convex over $(b_{ij}, c_{ij})$. Furthermore, $a_{ij}$ satisfies

$$\lim_{f_{ij}\downarrow b_{ij}} a_{ij}(f_{ij}) = \infty, \qquad \lim_{f_{ij}\uparrow c_{ij}} a_{ij}(f_{ij}) = \infty, \qquad (5.27a)$$

and its directional derivatives satisfy

$$\lim_{f_{ij}\downarrow b_{ij}} a_{ij}^-(f_{ij}) = -\infty, \qquad \lim_{f_{ij}\uparrow c_{ij}} a_{ij}^+(f_{ij}) = \infty. \qquad (5.27b)$$

In addition the problem (UCNF) has at least one feasible solution.

One consequence of the above assumption is that the primal problem (UCNF) has a unique optimal solution ($a_{ij}$ is seen to be continuous and the level sets of the cost function are compact, so Weierstrass' theorem, given as Prop. A.8 in Appendix A, applies). Similarly as earlier, the dual function is defined by [cf. Eq. (5.5)]

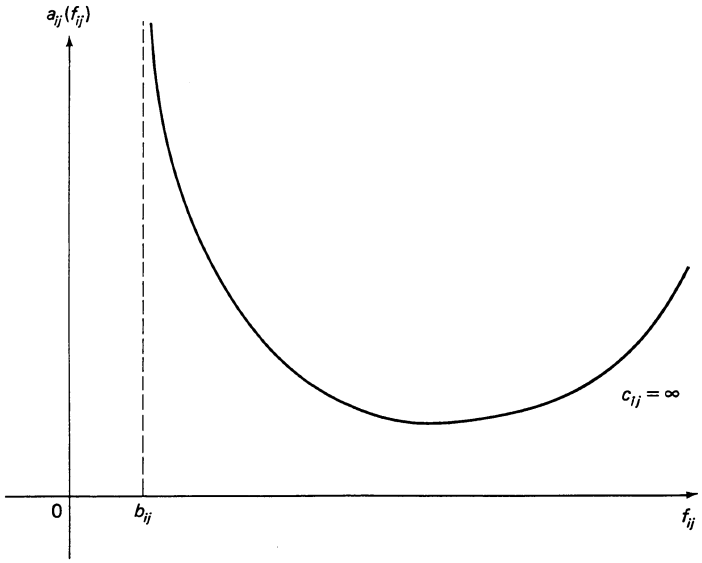



**Figure 5.5.6** Illustration of the conditions (5.27). The function $a_{ij}$ grows to $\infty$ as $f_{ij}$ approaches the endpoints of the (open) domain $(b_{ij}, c_{ij})$. The rate of increase also approaches infinity.

$$q(p) = \sum_{(i,j)\in A} q_{ij}(p_i - p_j) + \sum_{i\in N} s_i p_i, \tag{5.28a}$$

where

$$q_{ij}(p_i - p_j) = \min_{f_{ij}\in(b_{ij},c_{ij})} \{a_{ij}(f_{ij}) - (p_i - p_j)f_{ij}\}. \tag{5.28b}$$

A second consequence of Assumption 5.2 is that the minimum in the preceding equation is attained for all $p$, so the dual function value is finite. Using these facts, one can derive variations of Prop. 5.1 and Lemma 5.1 (Exercise 5.3), as well as a modified version of the relaxation algorithm. The CS conditions (5.7)–(5.9) are replaced by the condition

$$a_{ij}^-(f_{ij}) \le t_{ij} \le a_{ij}^+(f_{ij}).$$

It is seen that the strict convexity of $a_{ij}$ implies that, given $p$, there exists a unique flow vector satisfying the condition above together with $p$. Using this, one can verify the dual gradient expressions (5.15) and (5.16), and carry out a convergence analysis of the corresponding relaxation algorithm that is similar to the one given earlier.

Similar results hold for the case of hybrid problems where some of the arc flows are subject to upper and lower bounds, as in problem (CNF), while other arc flows are subject to no such constraints.

### 5.5.4 An Example: Constrained Matrix Problems

A problem that arises in a variety of contexts is to find an $n \times m$ matrix $F$ that has given row sums and column sums, and approximates a given $n \times m$ matrix $M$ in some optimal sense. It is often appropriate to formulate such a problem in terms of a bipartite graph consisting of $n$ sources and $m$ sinks. In this graph, the set of arcs $A$ consists of the pairs $(i, j)$ for which the corresponding entry $f_{ij}$ of the matrix $F$ is allowed to be nonzero. The given row sums $r_i$ and the given column sums $c_j$ imply the constraints

$$\sum_{\{j|(i,j)\in A\}} f_{ij} = r_i, \qquad i = 1, \ldots, n,$$

$$\sum_{\{i|(i,j)\in A\}} f_{ij} = c_j, \qquad j = 1, \ldots, m.$$

There may be also bounds for the entries $f_{ij}$ of $F$. Thus, the structure of this problem is similar to the structure of a transportation problem. The cost function to be optimized can take the form

$$\sum_{(i,j)\in A} a_{ij}(f_{ij}).$$

Commonly used examples expressing the objective of making the entries of $F$ close to the corresponding entries of $M$ are the quadratic function

$$a_{ij}(f_{ij}) = \tfrac{1}{2} \textstyle\sum_{(i,j)\in A} w_{ij}(f_{ij} - m_{ij})^2, \tag{5.29}$$

where $w_{ij}$ are given positive weights, and the logarithmic function

$$a_{ij}(f_{ij}) = f_{ij}\left[\ln\left(\frac{f_{ij}}{m_{ij}}\right) - 1\right], \tag{5.30}$$

where we assume that $m_{ij} > 0$ for all $(i, j) \in A$. Note that the logarithmic arc cost function (5.30) is not defined for $f_{ij} \leq 0$, but it can be seen that this does not cause difficulties either in the dual problem definition or in the corresponding relaxation algorithm, since the growth conditions (5.27) are satisfied for $b_{ij} = 0$ and $c_{ij} = \infty$.

Let us denote by $\pi_i$, $i = 1, \ldots, n$ the prices corresponding to the rows of $F$, and by $p_j$, $j = 1, \ldots, m$, the prices corresponding to the columns of $F$. For the quadratic cost function (5.29), the dual problem (assuming no bound constraints on the entries of $F$) is

$$\text{maximize} \qquad q(p) = \sum_{(i,j)\in A} \left(-\frac{(\pi_i - p_j)^2}{2w_{ij}} + m_{ij}(\pi_i - p_j)\right) - \sum_{i=1}^{n} r_i\pi_i + \sum_{j=1}^{m} c_j p_j$$

subject to no constraints on $\pi$ and $p$.

For every set of prices $(\pi, p)$, the unique flow vector that satisfies the CS conditions is

$$f_{ij} = m_{ij} - \frac{\pi_i - p_j}{w_{ij}}, \qquad (i,j) \in A.$$

The relaxation iteration with exact maximization along each price coordinate is given by

$$\pi_i := \frac{1}{\sum_{j \in O(i)} 1/w_{ij}} \left( \sum_{j \in O(i)} \left( \frac{p_j}{w_{ij}} + m_{ij} \right) - r_i \right), \qquad i = 1, \ldots, n,$$

$$p_j := \frac{1}{\sum_{i \in I(j)} 1/w_{ij}} \left( \sum_{i \in I(j)} \left( \frac{\pi_i}{w_{ij}} - m_{ij} \right) + c_j \right), \qquad j = 1, \ldots, m,$$

where

$$O(i) = \{ j \mid (i,j) \in A \}, \qquad I(j) = \{ i \mid (i,j) \in A \}.$$

Note that a synchronous parallel implementation of the method is possible, whereby in each iteration, the prices of all rows are updated simultaneously followed by simultaneous updating of the prices of all columns. Since no two rows (columns) are connected by an arc, this method is equivalent to the Gauss–Seidel method of this section with a particular order of choosing nodes for relaxation.

For the logarithmic cost function (5.30) the dual problem has the form

$$\text{maximize} \qquad q(p) = - \sum_{(i,j) \in A} m_{ij} e^{p_j - \pi_i} - \sum_{i=1}^{n} r_i \pi_i + \sum_{j=1}^{m} c_j p_j$$

subject to no constraints on $\pi$ and $p$.

The corresponding relaxation iteration sets $\pi_i$ to the value $\tilde{\pi}_i$, for which

$$\left( \sum_{j \in O(i)} m_{ij} e^{p_j} \right) e^{-\tilde{\pi}_i} = r_i, \qquad i = 1, \ldots, n,$$

and sets $p_j$ to the value $\tilde{p}_j$, for which

$$\left( \sum_{i \in I(j)} m_{ij} e^{-\pi_i} \right) e^{\tilde{p}_j} = c_j, \qquad j = 1, \ldots, m.$$

If we make the change of variables

$$y_i = e^{-\pi_i}, \qquad z_j = e^{p_j},$$

we obtain the iterations

$$y_i := \frac{r_i}{\sum_{j \in O(i)} m_{ij} z_j}, \qquad i = 1, \ldots, n, \tag{5.31}$$

$$z_j := \frac{c_j}{\sum_{i \in I(j)} m_{ij} y_i}, \qquad j = 1, \ldots, m. \tag{5.32}$$

The iterative method consisting of alternative updating of $y_i$ using Eq. (5.31) and updating of $z_j$ using Eq. (5.32) is known as *Kruithof's method* and dates to 1937 [Kru37]. It is used in telephony to predict traffic between $n$ origins and $m$ destinations based on predictions of the total traffic of origins and destinations. In this context, the known values $m_{ij}$ represent traffic intensity from origins $i$ to destinations $j$ as measured during some earlier period.

### 5.5.5 Parallel Implementations of the Relaxation Method

The parallel implementation aspects of the relaxation method of this subsection are similar to those of the $\epsilon$–relaxation method (Subsection 5.3.2). We assume that there is a processor assigned to each node, which has the responsibility of carrying out relaxation iterations at the node and communicating the results to adjacent processors/nodes. Minor modifications in the following are required in the case where several nodes are assigned to each processor. A possibility for further parallelization in executing approximately the coordinate minimization in the relaxation iteration is described in Exercise 5.7, and requires that a separate processor be used for each arc in each direction.

There are three main possibilities. The first is the synchronous Gauss–Seidel implementation, where the set of nodes is partitioned in subsets by means of a coloring scheme (cf. Subsection 1.2.4); there is no pair of nodes connected by an arc that is contained in the same subset. The algorithm is operated in phases. In each phase, a subset is selected, and a relaxation iteration is executed using each node of the subset. The iterations of the nodes in the same subset can be done in parallel. The method is then mathematically equivalent to a serial algorithm for which the convergence analysis of this section applies. Note that when the graph is bipartite as in the case of the constrained matrix problems of Subsection 5.5.4, it is possible to use just two subsets of nodes and execute a relaxation iteration at every node in just two phases.

The other possibilities are a synchronous Jacobi implementation and an asynchronous implementation. It was seen earlier (Fig. 5.5.5) that the synchronous Jacobi method need not converge to an optimal solution because adding the same constant to all prices leaves the dual cost unchanged. To overcome this difficulty, we consider in Section 6.5 a variation of the relaxation method, whereby the price of a single node, say node 1, is kept fixed, and we assume that there is a unique dual optimal vector $p$ with $p_1$ equal to the given fixed value. Under these circumstances, we show that the synchronous Jacobi method and a totally asynchronous relaxation method converge to that vector. A variation to be described in Chapter 7, that involves the use of a relaxation parameter and a partially asynchronous implementation, will be shown to converge as desired without

fixing the price of any node and without making any assumptions on the nature of the set of dual optimal price vectors.

The limited computational experience available with parallel implementations of the relaxation method of this section ([ZeM86] and [ZeL87]) indicates that the speedup obtained is substantial. This is due to the experimentally observed fact that there are many nodes with nonzero surplus throughout the algorithm. This observation is not supported by any analysis at present.

## EXERCISES

**5.1.** Consider the problem

$$\text{minimize} \quad \sum_{j=1}^{n} f_j(x_j)$$

$$\text{subject to} \quad Ax = s, \qquad b_j \leq x_j \leq c_j, \qquad j = 1, \ldots, n,$$

where $f_j : \Re \mapsto \Re$ are strictly convex functions, $A$ is a given $m \times n$ matrix, $s$ is a given vector in $\Re^m$, and $b_j$, $c_j$ are given scalars.

**(a)** Dualize the constraints $Ax = s$ and write the dual problem.

**(b)** Calculate the gradient of the dual function and describe the corresponding relaxation method. Under what circumstances is the method highly parallelizable?

**(c)** Repeat parts (a) and (b) for the case where the cost is a general strictly convex function $f : \Re^n \mapsto \Re$.

**5.2.** **(Necessary and Sufficient Optimality Condition.)** This exercise generalizes the necessary and sufficient optimality conditions for linear network flow problems (cf. Exercise 1.2). Show that a flow vector $f = \{f_{ij} \mid (i,j) \in A\}$ is optimal for the convex network flow problem (CNF) if and only if $f$ is feasible, and for every directed cycle $Y$ for which

$$f_{ij} < c_{ij}, \qquad \forall\, (i,j) \in Y^+,$$

$$b_{ij} < f_{ij}, \qquad \forall\, (i,j) \in Y^-,$$

we have

$$\sum_{(i,j) \in Y^+} a_{ij}^+(f_{ij}) - \sum_{(i,j) \in Y^-} a_{ij}^-(f_{ij}) \geq 0.$$

*Hint:* The rate of change of the primal cost at a feasible flow vector $f$ in the direction of a circulation $y$ is given by

$$d(f;y) = \sum_{\{(i,j) \mid y_{ij} > 0\}} a_{ij}^+(f_{ij}) y_{ij} + \sum_{\{(i,j) \mid y_{ij} < 0\}} a_{ij}^-(f_{ij}) y_{ij}.$$

If $f$ is feasible but not optimal, there must exist a circulation $y$ such that $d(f;y) < 0$. Use the Conformal Realization Theorem of Appendix B to show that the given optimality condition is then violated.

**5.3.** State and prove analogs of Lemma 5.1 and Prop. 5.1 for problem (UCNF) under Assumption 5.2.

**5.4.** **(A Problem of Agreement.)** The following problem will be reencountered in more general form in Chapter 7, where it will play a significant role in the analysis. Consider a connected undirected graph $(N, A)$. Each node $i$ has a scalar value $p_i(t)$ at time $t$. At each time $t$, a node $i$ is selected and its value is set to a convex combination of its value and a weighted average of the values of its neighbors, that is,

$$p_i(t+1) = \delta_i p_i(t) + (1-\delta_i) \sum_{j \in A(i)} a_{ij} p_j(t),$$

where $\delta_i \in [0,1)$ is a given scalar, $A(i)$ is the set of nodes connected to $i$ with an arc, and $a_{ij}$ are positive scalars such that $a_{ij} = a_{ji}$ and $\sum_{j \in A(i)} a_{ij} = 1$ for all $i$. Assuming each node is selected infinitely often, show that the nodes will asymptotically agree on a common value, that is, there exists some scalar $p^*$ such that

$$\lim_{t \to \infty} p_i(t) = p^*, \qquad \forall\, i \in N.$$

*Hint:* Replace each undirected arc $(i,j) \in A$ with two directed arcs $(i,j)$ and $(j,i)$. To each directed arc $(i,j)$ of the resulting network, assign the arc cost function $a_{ij}(f_{ij}) = (1/2a_{ij})f_{ij}^2$ and consider the problem of minimizing $\sum_{(i,j)} a_{ij}(f_{ij})$ subject to the constraint that $f$ is a circulation. Show that a price vector is optimal if and only if all its coordinates are equal. Apply the relaxation method.

**5.5.** **(Feasible Differential Theorem.)** Consider a directed graph $(N, A)$. For each arc $(i,j) \in A$, we are given two scalars $a_{ij}^- \in [-\infty, \infty)$ and $a_{ij}^+ \in (-\infty, \infty]$, with $a_{ij}^- \leq a_{ij}^+$. Show that there exists a price vector $p$ satisfying

$$a_{ij}^- \leq p_i - p_j \leq a_{ij}^+, \qquad \forall\, (i,j) \in A,$$

if and only if for every directed cycle $Y$, we have

$$0 \leq \sum_{(i,j) \in Y^+} a_{ij}^+ - \sum_{(i,j) \in Y^-} a_{ij}^-. \tag{5.33}$$

*Hint:* Consider the instance of (CNF) involving the piecewise linear arc cost functions

$$a_{ij}(f_{ij}) = \max\left\{a_{ij}^- f_{ij}, a_{ij}^+ f_{ij}\right\},$$

and the constraint that $f$ is a circulation (that is, $s_i = 0$ for all $i$, and there are no arc flow bounds). Show that if Eq. (5.33) holds for all $Y$, then $f = 0$ is an optimal solution of this piecewise linear cost problem, and otherwise there is no optimal solution. Convert this problem to a linear programming problem, and use the linear programming duality theory of Appendix C.

**5.6. (Proof of the Duality Theorem.)** Use Lemma 5.1, and the results of Exercises 5.2 and 5.5 to prove Prop. 5.1. *Hint:* We have

$$q(p') \leq \sum_{(i,j)\in A} a_{ij}(f'_{ij})$$

for all $p'$ and feasible $f'$ [cf. Eq. (C.12) in Appendix C]. If $f$ and $p$ satisfy primal feasibility and the CS conditions (5.7)–(5.9), argue that $f_{ij}$ attains the minimum in the definition (5.5b) of the dual arc function $q_{ij}$, and use Lemma 5.1 to show that equality is obtained above. Conversely, suppose that $f$ is optimal. Combine the results of Exercises 5.2 and 5.5 to show that there exists a price vector $p$ such that

$$a_{ij}^-(f_{ij}) \leq p_i - p_j \leq a_{ij}^+(f_{ij}), \qquad \forall\ (i,j) \in A.$$

Show that this price vector is dual optimal.

**5.7. (Parallelization of Stepsize Selection [Tse87].)** Consider a relaxation iteration where a node $i$ with $g_i(p) > 0$ is chosen, and we wish to find a stepsize $\bar{\gamma}$ such that for some $\delta \in [0,1)$, we have $0 \leq g_i(\bar{p}) \leq \delta g_i(p)$, where $\bar{p}_i = p_i + \bar{\gamma}$ and $\bar{p}_j = p_j$ for $j \neq i$. The following is a parallelizable procedure for computing a suitable stepsize $\bar{\gamma}$. Let $\mu$ be any scalar with $\mu \in (0,1)$, and define for all $\gamma \in \Re$ and all incident arcs to node $i$

$$f_{ij}(\gamma) = \arg \min_{b_{ij}\leq \xi \leq c_{ij}} \{a_{ij}(\xi) - (p_i - p_j + \gamma)\xi\},$$

$$f_{ji}(\gamma) = \arg \min_{b_{ji}\leq \xi \leq c_{ji}} \{a_{ji}(\xi) - (p_j - p_i - \gamma)\xi\}.$$

Let $n_i$ be the number of arcs $(i,j)$ that are outgoing from $i$ and satisfy $f_{ij} < c_{ij}$, plus the number of arcs $(j,i)$ that are incoming to $i$ and satisfy $f_{ji} > b_{ji}$. For each arc $(i,j)$ that is outgoing from $i$ and $f_{ij} < c_{ij}$, set

$$\gamma_{ij} = \infty, \qquad \text{if} \quad f_{ij}(\gamma) - f_{ij} < \frac{\mu}{n_i} g_i(p), \qquad \forall\ \gamma \geq 0,$$

and otherwise let $\gamma_{ij}$ be a scalar such that

$$\frac{\mu}{n_i} g_i(p) \leq f_{ij}(\gamma_{ij}) - f_{ij} \leq \frac{1}{n_i} g_i(p).$$

Similarly, for each arc $(j,i)$ that is incoming to $i$ and $f_{ji} > b_{ji}$ set

$$\gamma_{ji} = \infty, \qquad \text{if} \qquad f_{ji} - f_{ji}(\gamma) < \frac{\mu}{n_i} g_i(p), \qquad \forall\ \gamma \geq 0,$$

and otherwise let $\gamma_{ji}$ be a scalar such that

$$\frac{\mu}{n_i} g_i(p) \leq f_{ji} - f_{ji}(\gamma_{ji}) \leq \frac{1}{n_i} g_i(p).$$

(Note that all the scalars $\gamma_{ij}$ and $\gamma_{ji}$ can be computed in parallel in a system that assigns a separate processor to each incident arc to node $i$.) Let

$$\bar{\gamma} = \min \left\{ \min_{\{j \mid (i,j) \in A\}} \gamma_{ij}, \min_{\{j \mid (j,i) \in A\}} \gamma_{ji} \right\}.$$

**(a)** Show that $\bar{\gamma} > 0$, and that

$$0 \leq g_i(\bar{p}) \leq \left(1 - \frac{\mu}{n_i}\right) g_i(p).$$

**(b)** Provide a similarly parallelizable stepsize procedure for the case where $g_i(p) < 0$.

**5.8.** ($\epsilon$**–Complementary Slackness [BHT87].**) Consider the convex network problem (CNF). Given $\epsilon \geq 0$, we say that a flow–price vector pair $(f, p)$ satisfies $\epsilon$–complementary slackness ($\epsilon$–CS) if for all $(i, j) \in A$, we have $f_{ij} \in [b_{ij}, c_{ij}]$ and

$$\tilde{a}_{ij}^-(f_{ij}) - \epsilon \leq p_i - p_j \leq \tilde{a}_{ij}^+(f_{ij}) + \epsilon,$$

where $\tilde{a}_{ij}^+(f_{ij})$ and $\tilde{a}_{ij}^-(f_{ij})$ are defined by Eqs. (5.17a) and (5.17b), respectively. Verify that this definition contains as a special case the corresponding definition of Section 5.3, and show that if $(f, p)$ satisfies $\epsilon$–CS, then

$$0 \leq \sum_{(i,j) \in A} a_{ij}(f_{ij}) - q(p) \leq \epsilon \sum_{(i,j) \in A} (c_{ij} - b_{ij}).$$

## 5.6 NONLINEAR MULTICOMMODITY FLOW PROBLEMS–ROUTING APPLICATIONS

In this section, we consider a network flow routing model that is more complex than the models of the previous sections in that we distinguish between several independently constrained types of flow (commodities) that share the arcs of the given network. Typical applications of this model arise in routing of data in computer communication networks, and in equilibrium studies of transportation networks. The size of the problems encountered in the context of these applications is often very large, so it may be essential to speedup the solution method through parallelization in order to meet practical solution time constraints. In the case of routing in a data network, a distributed on–line implementation is often desirable; we refer to [BeG87] for a description of some of the practical issues arising in this context.

We are given a network and a set $W$ of ordered pairs $w$ of distinct nodes referred to as the origin and the destination of $w$. We refer to $w$ as an OD pair. For each $w$, we are given a scalar $r_w$ referred to as the input traffic of $w$. In the context of routing in a data network, $r_w$ (measured in data units/second) is the arrival rate of traffic entering and exiting the network at the origin and the destination of $w$, respectively. The routing

objective is to divide each $r_w$ among the many paths from origin to destination in a way that the resulting total arc flow pattern minimizes a suitable cost function. We denote:

$P_w$: A given set of simple positive paths that start at the origin and end at the destination of $w$.

$x_p$: The flow of path $p$.

The collection of all path flows $\{x_p \mid w \in W, p \in P_w\}$ must satisfy the constraints

$$\sum_{p \in P_w} x_p = r_w, \qquad \forall\, w \in W, \tag{6.1}$$

$$x_p \geq 0, \qquad \forall\, p \in P_w,\ w \in W, \tag{6.2}$$

as shown in Fig. 5.6.1. The total flow $F_{ij}$ of arc $(i,j)$ is the sum of all path flows traversing the arc:

$$F_{ij} = \sum_{\substack{\text{all paths } p \\ \text{containing } (i,j)}} x_p. \tag{6.3}$$

Consider a cost function of the form

$$\sum_{(i,j)} D_{ij}(F_{ij}). \tag{6.4}$$

The problem is to find a set of path flows $\{x_p\}$ that minimize this cost function subject to the constraints of Eqs. (6.1) to (6.3).

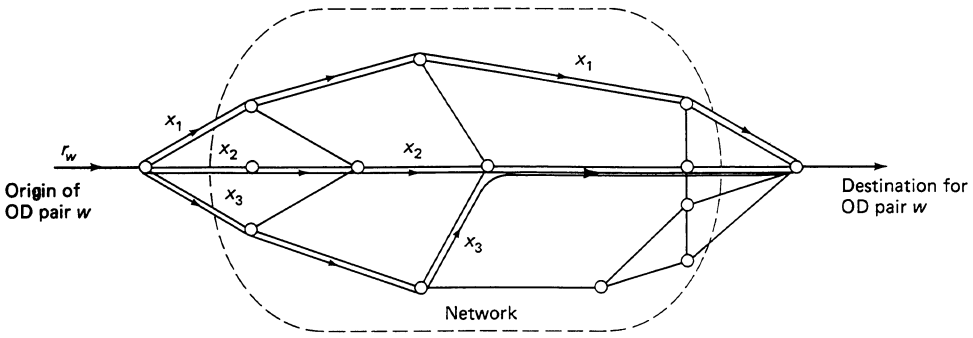



**Figure 5.6.1** Constraints for the path flows of an OD pair $w$. The path flows should be nonnegative and add up to the given traffic input $r_w$ of the OD pair.

A frequently used function $D_{ij}$ in packet routing applications is

$$D_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij} - F_{ij}} + d_{ij}F_{ij}, \tag{6.5}$$

where $C_{ij}$ is the transmission capacity of arc $(i,j)$ measured in the same units as $F_{ij}$, and $d_{ij}$ is the processing and propagation delay. (This function is defined for $F_{ij} \in [0, C_{ij})$ and is usually modified during algorithmic solution so that it is defined as a convex function for all $F_{ij} \geq 0$.) With this formula, the cost function (6.4) expresses the average number of packets in the system based on the hypothesis that each queue behaves as an M/M/1 queue of packets [BeG87].

By expressing the total flows $F_{ij}$ in terms of the path flows in the cost function (6.4) [using Eq. (6.3)], the problem can be formulated in terms of the path flow variables $\{x_p \mid p \in P_w,\ w \in W\}$ as

$$\begin{aligned} &\text{minimize} && D(x) \\ &\text{subject to} && \sum_{p \in P_w} x_p = r_w, \qquad \forall\ w \in W, \\ &&& x_p \geq 0, \qquad \forall\ p \in P_w,\ w \in W, \end{aligned} \tag{6.6}$$

where

$$D(x) = \sum_{(i,j)} D_{ij} \left( \sum_{\substack{\text{all paths } p \\ \text{containing } (i,j)}} x_p \right)$$

and $x$ is the vector of path flows $x_p$.

We now turn to the characterization of an optimal solution. In particular, we will show that optimal routing directs traffic exclusively along paths that are shortest with respect to some arc lengths that depend on the flows carried by the arcs. For this, we assume that each $D_{ij}$ is defined on $[0, \infty)$ and is twice differentiable on $(0, \infty)$. The case where $D_{ij}$ is defined in an interval $[0, C_{ij})$, where $C_{ij}$ is a positive number [the capacity of the arc in a routing context, cf. Eq. (6.5)] can be handled by extending the definition of $D_{ij}$ beyond the interval $[0, C_{ij})$, and by suitably modifying $D_{ij}(F_{ij})$ for $F_{ij}$ near $C_{ij}$; see [BeG87], p. 416. The first and second derivatives of $D_{ij}$ are denoted by $D'_{ij}$ and $D''_{ij}$, respectively, and are assumed positive for all $F_{ij}$. This implies in particular that $D_{ij}$ is strictly convex and monotonically increasing. It is seen that the optimization problem of Eq. (6.6), viewed as a problem in the path flow variables $\{x_p\}$, has a convex twice differentiable cost function and a convex, compact constraint set.

The partial derivative of $D$ with respect to $x_p$ is given by

$$\frac{\partial D(x)}{\partial x_p} = \sum_{\substack{\text{all arcs } (i,j) \\ \text{on path } p}} D'_{ij}, \tag{6.7}$$

where the first derivatives $D'_{ij}$ are evaluated at the arc flows $F_{ij}$ corresponding to $x$. From this equation, it is seen that *$\partial D/\partial x_p$ is the length of path $p$ when the length of each arc $(i,j)$ is taken to be the first derivative $D'_{ij}$ evaluated at $x$.* Consequently, in what follows, $\partial D/\partial x_p$ is called the *first derivative length of path $p$.*

According to Prop. 3.1 in Section 3.3, $x^* = \{x_p^*\}$ is an optimal path flow vector if and only if it is feasible, that is, it satisfies the constraints

$$\sum_{p \in P_w} x_p = r_w, \qquad x_p \geq 0, \qquad \forall\, p \in P_w,\ w \in W, \tag{6.8}$$

and the rate of change of the cost function is nonnegative along every "feasible direction", that is,

$$\sum_{w \in W} \sum_{p \in P_w} \frac{\partial D(x^*)}{\partial x_p}(x_p - x_p^*) \geq 0, \tag{6.9}$$

for all $\{x_p\}$ that are feasible. Note that the optimality conditions (6.8) and (6.9) represent a variational inequality which is identical to the one encountered in the traffic assignment problem (Section 3.5) if the path travel time $t_p(x)$ in the latter problem is identified with the first derivative length of path $p$, that is,

$$t_p(x) = \frac{\partial D(x)}{\partial x_p}. \tag{6.10}$$

From the equivalence of Eqs. (5.2) and (5.3) in Section 3.5, it is seen that the optimality conditions (6.8) and (6.9) are equivalent to having, for all $w \in W$ and $p \in P_w$

$$x_p^* > 0 \quad \text{only if} \quad \left[\frac{\partial D(x^*)}{\partial x_{p'}} \geq \frac{\partial D(x^*)}{\partial x_p}, \qquad \forall\, p' \in P_w\right]. \tag{6.11}$$

In words, the above condition says that *a set of path flows is optimal if and only if path flow is positive only on paths with a minimum first derivative length.* This parallels the minimum travel time condition implied by the user optimization principle in the traffic assignment problem. The condition of Eq. (6.11) also implies that at an optimum, the paths along which the input flow $r_w$ of OD pair $w$ is split must have *equal* length (and less than or equal length to that of all other paths of $w$).

We now consider iterative algorithms for solving the optimal routing problem. One possible method is based on the scaled gradient projection method discussed in Section 3.3. To apply this method, we write the optimization problem of Eq. (6.6) as

$$\begin{aligned} &\text{minimize} && D(x) \\ &\text{subject to} && x_w \in X_w, \qquad \forall\, w \in W, \end{aligned} \tag{6.12}$$

where $x_w$ is the vector of path flows of OD pair $w$:

$$x_w = \{x_p \mid p \in P_w\}, \tag{6.13}$$

and $X_w$ is the feasible set for $x_w$:

$$X_w = \left\{ x_w \,\middle|\, \sum_{p \in P_w} x_p = r_w,\ x_p \geq 0,\ p \in P_w \right\}. \tag{6.14}$$

The gradient projection method replaces the current iterates $x_w$, $w \in W$, with the corresponding solutions $\bar{x}_w$ of the quadratic programming problems

$$\begin{aligned} &\text{minimize} && (\bar{x}_w - x_w)' \nabla_w D(x) + \tfrac{1}{2}(\bar{x}_w - x_w)' M_w (\bar{x}_w - x_w) \\ &\text{subject to} && \bar{x}_w \in X_w, \end{aligned} \tag{6.15}$$

where $\nabla_w D$ denotes the gradient of $D$ with respect to $x_w$, and $M_w$ is a matrix that satisfies $x_w' M_w x_w > 0$, for every nonzero $x_w$ in the subspace

$$\left\{ x_w \,\middle|\, \sum_{p \in P_w} x_p = 0 \right\}. \tag{6.16}$$

A suitable practical choice is to select $M_w$ diagonal with the scalars

$$\frac{1}{\gamma} \frac{\partial^2 D(x)}{\partial x_p^2}, \qquad p \in P_w$$

along the diagonal, where $\gamma$ is a positive stepsize parameter. This choice corresponds to an approximation of a constrained form of Newton's method, where the off–diagonal terms of the Hessian matrix have been set to zero. Note that $M_w$ can change from one iteration to the next. The convergence properties of the algorithm can be inferred from the convergence result for the gradient projection method given in Section 3.3 (Prop. 3.6). Note that the algorithm admits massive parallelism, since the OD pairs $w$ can be assigned to separate processors that can solve the quadratic subproblems (6.15) in parallel.

Another gradient projection algorithm (which in fact can be shown to be a special case of the preceding one for a particular choice of the matrices $M_w$) is based on converting the simplex constraints of the optimization problem (6.6) (for the purpose of the next iteration) into nonnegativity constraints. This is done as follows: at each iteration, we calculate for each OD pair $w \in W$ a path $\bar{p}_w$ of minimum first derivative length (MFDL). We then express the flows of the MFDL paths $\bar{p}_w$ in terms of the other path flows while eliminating the equality constraints

$$\sum_{p \in P_w} x_p = r_w$$

in the process. For each $w$, we eliminate $x_{\bar{p}_w}$ from the cost function $D(x)$ using the equation

$$x_{\overline{p}_w} = r_w - \sum_{\substack{p \in P_w \\ p \neq \overline{p}_w}} x_p, \tag{6.17}$$

thereby obtaining a problem of the form

$$\begin{aligned} &\text{minimize} && \tilde{D}(\tilde{x}) \\ &\text{subject to} && x_p \geq 0, \quad \forall\, w \in W,\ p \in P_w,\ p \neq \overline{p}_w, \end{aligned} \tag{6.18}$$

where $\tilde{x}$ is the vector of all path flows $x_p$ with $p \neq \overline{p}_w$ for all $w \in W$. Note that the constraint $x_{\overline{p}_w} \geq 0$ has been ignored because the flow $x_{\overline{p}_w}$ can only be increased during the iteration as will be seen shortly. Note also that the path $\overline{p}_w$ may change from one iteration to the next, so the vector $\tilde{x}$ and the function $\tilde{D}$ depend on the iteration count, but for simplicity we have not shown this dependence.

Using Eq. (6.17) and the definition of $\tilde{D}(\tilde{x})$, we obtain

$$\frac{\partial \tilde{D}(\tilde{x}^k)}{\partial x_p} = \frac{\partial D(x^k)}{\partial x_p} - \frac{\partial D(x^k)}{\partial x_{\overline{p}_w}}, \qquad \forall\, p \in P_w,\ p \neq \overline{p}_w,\ w \in W. \tag{6.19}$$

Regarding second derivatives, a straightforward differentiation of the first derivative expressions (6.19) and (6.7) shows that

$$\frac{\partial^2 \tilde{D}(\tilde{x}^k)}{\partial x_p^2} = \sum_{(i,j) \in L_p} D''_{ij}(F_{ij}^k), \qquad \forall\, w \in W,\ p \in P_w,\ p \neq \overline{p}_w, \tag{6.20}$$

where, for each $p$,

$L_p$: Set of arcs belonging to either $p$ or the corresponding MFDL path $\bar{p}_w$, but not both.

We now use the gradient projection iteration for positivity constraints and diagonal second derivative scaling applied to the "reduced" cost function $\tilde{D}(\tilde{x})$. The iteration takes the form

$$x_p^{k+1} = \max\bigl\{0, x_p^k - \gamma^k H_p^{-1}(d_p - d_{\overline{p}_w})\bigr\}, \qquad \forall\, w \in W,\ p \in P_w,\ p \neq \overline{p}_w, \tag{6.21}$$

where $d_p$ and $d_{\overline{p}_w}$ are the first derivative lengths of the paths $p$ and $\overline{p}_w$, respectively, given by [cf. Eq. (6.7)]

$$d_p = \sum_{\substack{\text{all arcs } (i,j) \\ \text{on path } p}} D'_{ij}(F_{ij}^k), \qquad d_{\overline{p}_w} = \sum_{\substack{\text{all arcs } (i,j) \\ \text{on path } \overline{p}_w}} D'_{ij}(F_{ij}^k), \tag{6.22}$$

and $H_p$ is the "second derivative length"

$$H_p = \sum_{(i,j) \in L_p} D''_{ij}(F_{ij}^k) \tag{6.23}$$

given by Eq. (6.20). The stepsize $\gamma^k$ is some positive scalar which may be chosen by a variety of methods to be discussed shortly.

The following observations can be made regarding the gradient projection iteration (6.21):

1. Since for each OD pair $w \in W$, we have $d_p \geq d_{\overline{p}_w}$ for all $p \neq \overline{p}_w$, it follows that all the nonshortest path flows $x_p$ ($p \neq \overline{p}_w$) that are positive will be reduced with the corresponding increment of flow being shifted to the MFDL path $\overline{p}_w$.
2. Those nonshortest path flows $x_p$, $p \neq \overline{p}_w$, that are zero will stay at zero. Therefore, the calculation indicated in the gradient projection iteration (6.21) should only be carried out for paths that carry positive flow.
3. Only paths that carried positive flow at the start or were MFDL paths at some previous iteration can carry positive flow at the beginning of an iteration. This is important in that it tends to keep the number of flow carrying paths small with a corresponding reduction in the amount of calculation and overhead needed at each iteration.

Regarding the choice of the stepsize $\gamma^k$, there are several possibilities. It is possible to select $\gamma^k$ to be constant ($\gamma^k \equiv \gamma$ for all $k$). With this choice, it can be shown that given any starting set of path flows, there exists $\overline{\gamma} > 0$ such that if $\gamma \in (0, \overline{\gamma}]$, then a sequence generated by the gradient projection iteration (6.21) converges to the optimal cost of the problem [this can be proved by viewing the iteration as a special case of the gradient projection method based on Eq. (6.15) for a particular choice of $M_w$, and will also be proved in more general form in Chapter 7]. A crucial question has to do with the magnitude of the constant stepsize. It is known from nonlinear programming experience and analysis that a stepsize equal to unity usually works well with Newton's method as well as diagonal approximations to Newton's method that employ scaling based on second derivatives ([Ber82a] and [Lue84]). Experience has verified that a choice of the stepsize $\gamma^k$ in iteration (6.21) near unity typically works quite well regardless of the values of the input flows $r_w$ [BGV79]. On a serial machine, better performance is usually obtained if iteration (6.21) is carried out *one OD pair (or one origin) at a time*, that is, first carry out the iteration with $\gamma^k = 1$ for a single OD pair (or origin), adjust the corresponding total arc flows to account for the effected change in the path flows of this OD pair (or origin), and continue with the next OD pair until all OD pairs are taken up cyclically. This is a form of linearized block Gauss–Seidel method of the type discussed in Section 3.3. The rationale here is based on the fact that dropping the off–diagonal terms of the Hessian matrix, in effect, neglects the interaction between the flows of different OD pairs. In other words, iteration (6.21) is based to some extent on the premise that each OD pair will adjust its own path flows while the other OD pairs will keep theirs unchanged. Carrying out iteration (6.21) one OD pair at a time reduces the potentially detrimental effect of the neglected off–diagonal terms of the Hessian, and increases the likelihood that the unity stepsize is appropriate and effective. Under these circumstances, experience shows that iteration (6.21) typically works well with a unity stepsize.

For serial computation, it is possible to choose $\gamma^k$ by a simple form of line search. For example, start with a unity stepsize, evaluate the corresponding cost, and if no reduction is obtained over $D(x^k)$, successively reduce the stepsize until a cost reduction $D(x^{k+1}) < D(x^k)$ is obtained. (It is noted that this scheme cannot be shown to converge to the optimal solution. However, it typically works well in practice. Similar schemes with better theoretical convergence properties are described in [Ber76a] and [Ber82b].)

The projection algorithm typically yields rapid convergence to a neighborhood of an optimal solution. Once it comes near a solution (how "near" depends on the problem), it tends to slow down. For a projection algorithm to converge fast after it approaches an optimal solution, it is necessary that the off–diagonal terms of the Hessian matrix are taken into account. Surprisingly, it is possible to implement sophisticated methods of this type (see [BeG83]), but we will not go into this topic further. These methods are based on a more accurate approximation of a constrained version of Newton's method (using the conjugate gradient method) near an optimal solution. However, when far from a solution, their speed of convergence is usually only slightly superior to that of the simple diagonally scaled gradient projection iteration (6.21). Furthermore, these methods are typically less suitable for parallelization than iteration (6.21).

Consider now synchronous parallel and distributed implementations of the gradient projection iteration (6.21). There are two main situations of interest. The first arises in a distributed data network routing context, where each node (arc) of the network flow problem is also a node (communication link) of the data network. The most straightforward possibility for distributed implementation is for all nodes $i$ to broadcast to all other nodes the current total flows $F_{ij}^k$ of their outgoing arcs $(i,j)$ [BeG87]. Each node can then compute the MFDL paths of OD pairs for which it is the origin, and can execute iteration (6.21) for some fixed stepsize. The method can also be implemented in an asynchronous, distributed format, whereby the computation and information reception are not synchronized at each node. An analysis under these conditions is given in Chapter 7.

The second type of synchronous parallel implementation involves the use of a regular interconnection network of processors such as a hypercube. The $i$th processor is then assigned to updating the path flows of all OD pairs in a subset $W_i$ by using iteration (6.21); for example, $W_i$ can consist of all OD pairs that have a given node as origin. Let us assume for concreteness that each $P_w$ is the set of all simple positive paths connecting the origin node and the destination node of $w$. To execute the $k$th iteration, processor $i$ must know the shortest paths of its own OD pairs $w \in W_i$; this requires a shortest path computation of the type discussed in Section 4.1, using the link flows $F_{ij}^k$ (from which the corresponding first derivative lengths $D'_{ij}(F_{ij}^k)$ can be obtained). (If additional processors are available, the shortest path computation can be parallelized along the lines discussed in Section 4.1.) Processor $i$ can then proceed to update the path flows of the OD pairs in $W_i$ using iteration (6.21) and the corresponding first and second derivative lengths. The processors must then cooperatively calculate the arc flows $F_{ij}^{k+1}$ needed for the next iteration. One way to do this is for each processor $i$ to calculate the change of the flow of every arc that results from its own updating of the path flows of the OD pairs in $W_i$. These arc flow changes can be communicated to all other nodes using a

multinode broadcast (cf. Subsection 1.3.4). Upon reception from all processors of the corresponding changes of all arc flows, each processor, knowing the arc flows of the previous iteration $F_{ij}^k$, can calculate the arc flows $F_{ij}^{k+1}$ needed for the next iteration.

## NOTES AND SOURCES

**5.1** There are a number of standard references on network flow optimization ([BaJ77], [Dan63], [FoF62], [JeB80], [KeH80], [Law76], [Min78], [PaS82], [Roc84], and [Zou76]). Of these, [Roc84] emphasizes duality and is closest in spirit to the material of the chapter. For parallel implementations of network simplex methods see [Pet88]. Methods of multipliers (Exercises 1.7 and 1.8) are interesting but untested possibilities for massively parallel solution of network flow problems.

**5.2.2** The relaxation method with multiple node price changes was first proposed in the context of the assignment problem in [Ber81]. Its extension to the general linear network flow problem is due to [Ber82c] and [BeT85]. The method is implemented in a public domain code called RELAX, which is described in [BeT85] and [BeT88]. The main difference between this method and the more traditional primal–dual method (cf. Exercise 2.3) is that the direction of ascent used in the primal–dual method is the vector of the form (2.7) along which the directional derivative of the dual cost is maximized, whereas the direction used in the relaxation method is the first found vector of the form (2.7) along which the directional derivative is positive. Thus the direction of ascent is usually obtained with a lot less computation in the relaxation method than in the primal–dual method, while the "quality" of the ascent directions used in the relaxation method is typically not much worse than for the primal–dual method. For this reason the relaxation method, for most practical problems, terminates much faster than the primal–dual method. The method of this subsection has been extended to network flow problems with gains ([BeT85] and [Tse86]), to general linear programs ([Tse86] and [TsB87b]), to network flow problems with nonlinear arc cost functions [BHT87], to separable convex problems with linear constraints ([Tse86] and [TsB87c]), and to linear programs with decomposable structure of the type discussed in Section 3.4 [Tse86].

**5.3** The $\epsilon$–relaxation method is due to [Ber86a] and [Ber86b]. The auction algorithm was first given together with a complexity analysis in [Ber79b] (see also [Ber85] and [Ber88]). As shown in Exercise 3.5, it is possible to obtain the auction algorithm by means of a variation of the $\epsilon$–relaxation method. The reverse is also true. The general linear network flow problem (LNF) can be converted first into a transportation problem and then into an assignment problem (see Fig. 5.1.4). By applying the auction algorithm to this assignment problem, one can derive a generic form of the $\epsilon$–relaxation method.

The notion of $\epsilon$–complementary slackness was first introduced in connection with the auction algorithm [Ber79b], and later in the context of linear and nonlinear network flow problems ([BeT85] and [BHT87]), and convex separable problems with linear

constraints [TsB87c]. It was also developed independently in the analysis of [Tar85] for the special case where the flow vector $f$ is feasible. There is a conceptual relationship between the use of the $\epsilon$–complementary slackness idea in the relaxation methods of this section, and the use of the notion of an $\epsilon$–subgradient in nondifferentiable optimization methods [BeM73], [Lem74], [Lem75], and [Roc84].

**5.4** The sweep implementation was first given, except for some details, in [Ber86a] together with its complexity analysis without the use of scaling. The max–flow version of the $\epsilon$–relaxation method bears a close resemblance with the max–flow algorithm of [Gol85a] and [GoT86], which was developed independently of the relaxation and auction ideas. This max–flow algorithm uses node labels that estimate distances over usable paths. In the context of the $\epsilon$–relaxation approach, these labels can be viewed as dual prices. Because all arc costs are zero in the max–flow formulation of Fig. 5.1.2, the value of $\epsilon$ is immaterial as long as it is positive ($\epsilon = 1$ is used in [Gol85a] and [GoT86]). The max–flow version of the $\epsilon$–relaxation method, first given in [Ber86b], is simpler than the algorithm of [Gol85a], [GoT86] in that it obtains an optimal primal solution in a single phase rather than two (a single phase algorithm was also given later in [Gol87a]). It can also be initialized with arbitrary prices whereas in the max–flow algorithm of [GoT86] the initial prices must satisfy $p_i \leq p_j + 1$ for all arcs $(i, j)$. Our complexity analysis uses the general line of argument given for the max–flow algorithm of [GoT86]. In contrast with the latter algorithm, however, the $\epsilon$–relaxation method requires the use of the admissible graph and either the sweep implementation or some other implementation that maintains the acyclicity of the admissible graph. It is also necessary to use the admissible graph and something like the sweep implementation in the max–flow version of the $\epsilon$–relaxation method when the initial prices are arbitrary. We note that sharper complexity results are possible for the max–flow problem than for the general linear problem [AhO86]. A different type of parallel max–flow algorithm is given in [ScV82]. Parallel algorithms for the problem of maximal matching are given in [IsS86] and [KiC87].

**5.4.1** Scaling is the standard technique for improving the complexity of network flow algorithms ([BlJ85], [EdK72], and [Roc80]). There are two main scaling approaches in connection with the $\epsilon$–relaxation method. The first is based on scaling the arc costs similarly as the method given in Section 5.4. The second is based on scaling the value of $\epsilon$ and consists of applying the $\epsilon$–relaxation method repeatedly with successively smaller values of $\epsilon$, starting with a large value and ending with a value less than $1/|N|$ (see Exercise 4.4). This method, called *$\epsilon$–scaling*, was mentioned in [Ber79] as a method for improving the performance of the auction algorithm based on the result of computational experimentation. $\epsilon$–scaling was first analyzed for the general linear network flow problem in [Gol86b], where an algorithm with an $O\big(|N||A|\log(|N|)\log(|N|C)\big)$ running time was suggested. The timing estimate for this algorithm, which uses a dynamic tree data structure, was more fully established in [Gol87a] and [GoT87], where algorithms with $O\left(|N|^{5/3}|A|^{2/3}\log(|N|C)\right)$, and $O\left(|N|^3\log(|N|C)\right)$ running times were also given. The $O\left(|N|^3\log(|N|C)\right)$ algorithm uses the sweep implementation. The scaling analysis given here is due to [BeE87a]

(see also [BeE88]), and uses some of the earlier ideas of [Gol86b]. Related work includes [BIJ85], which provides a scaling analysis of an $O\left(|N|^4 \log C\right)$ algorithm based on the traditional primal–dual method (cf. Exercise 2.3).

**5.5.1** The relaxation algorithm has been applied to the dual problem of a convex programming problem by a number of authors ([CDZ86], [CoP82], [Cry71], [Hil57], and [OhK84]). [Roc84] introduces the class of monotropic programming problems, and gives an extensive analysis of the corresponding duality framework. The convergence analysis given here is due to [BHT87], which also provides an algorithm for network problems with arc cost functions that are not strictly convex. This algorithm generalizes both the relaxation algorithm of Section 5.5 and the multiple node relaxation algorithm of Subsection 5.2.2. Extensions of the convergence analysis of this section to problems that do not have a network structure are given in [TsB87a] and [TsB87c].

**5.5.3** Constrained matrix problems and special cases thereof have a long history in telephone communications and other contexts ([BaK78], [BaK80], [ChC58], [Cot84], [Gra71], [Kru37], [LaS81], and [Mac79]).

**5.6** Multicommodity network flow problems have been considered by many authors in the context of the traffic assignment problem (see the references for Section 3.5). Distributed routing algorithms for data networks were introduced in [Gal77] (whose algorithm bears a relation with the gradient projection method; see [Ber79c], [Gaf79], and [BGG84]), and [Ste77] (which is a dual relaxation method, similar to the ones discussed in Section 5.5). The method of [Gal77] is suitable for distributed real–time implementation. The algorithm of this section was introduced in [Ber80] (see also [BeG82], [BeG83], and [TsB86]).

# 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 $\epsilon$–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 $j$th 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 $i$th 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 $i$th 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 $i$th 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.

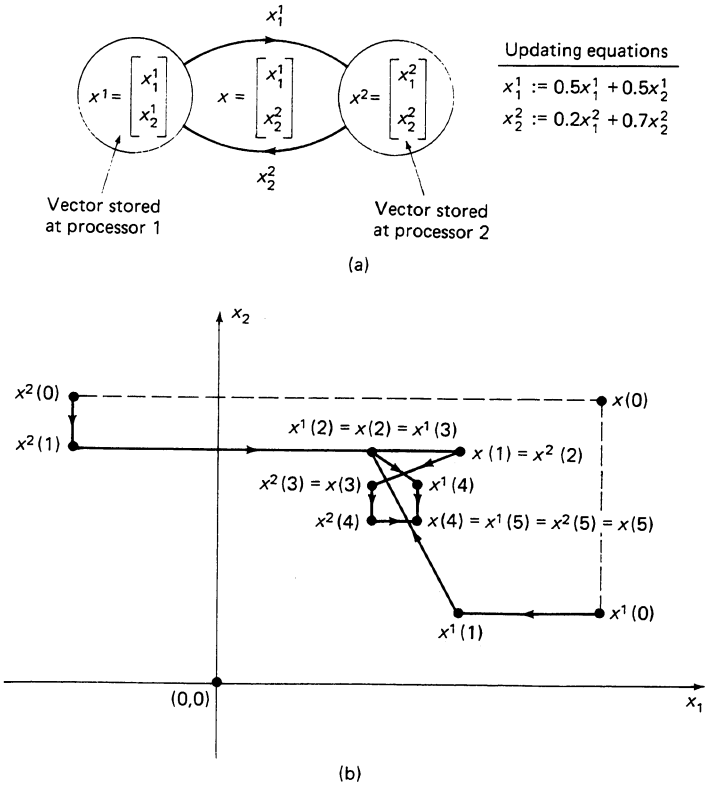



**Figure 6.1.1** (a) An example of an asynchronous algorithm involving two processors and a problem with two variables. Processor $i$, $(i = 1, 2)$, stores $x^i = (x_1^i, x_2^i)$, updates the variable $x_i^i$ at times $t \in T^i$, and transmits it to the other processor. (b) Evolution of the vector $x(t)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

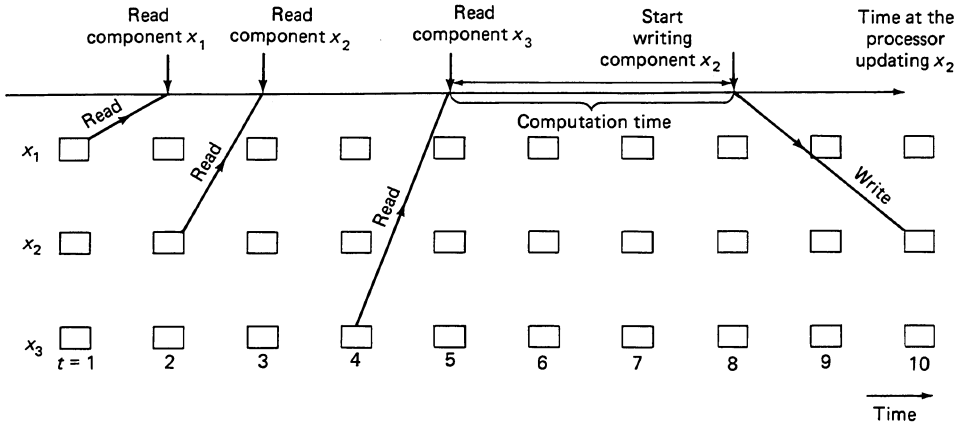



**Figure 6.1.2** Illustration of a component update in a shared memory multiprocessor. Here $x_2$ is viewed as being updated at time $t = 9$ ($9 \in T^2$), with $\tau_1^2(9) = 1$, $\tau_2^2(9) = 2$, and $\tau_4^2(9) = 4$. The updated value of $x_2$ is entered at the corresponding register at $t = 10$. 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 $i$th component of $x$ with the $i$th 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 $ij$th 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 $\gamma$ is a positive scalar stepsize. Note that for $\gamma$ 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 $\gamma$ 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 $\gamma$ 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 $\epsilon$ 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 $i$th 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 $\alpha$ 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, $\rho_A$ 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, $\rho_A$ 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 $\rho_A$ and compares it with $\rho_S$, 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

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

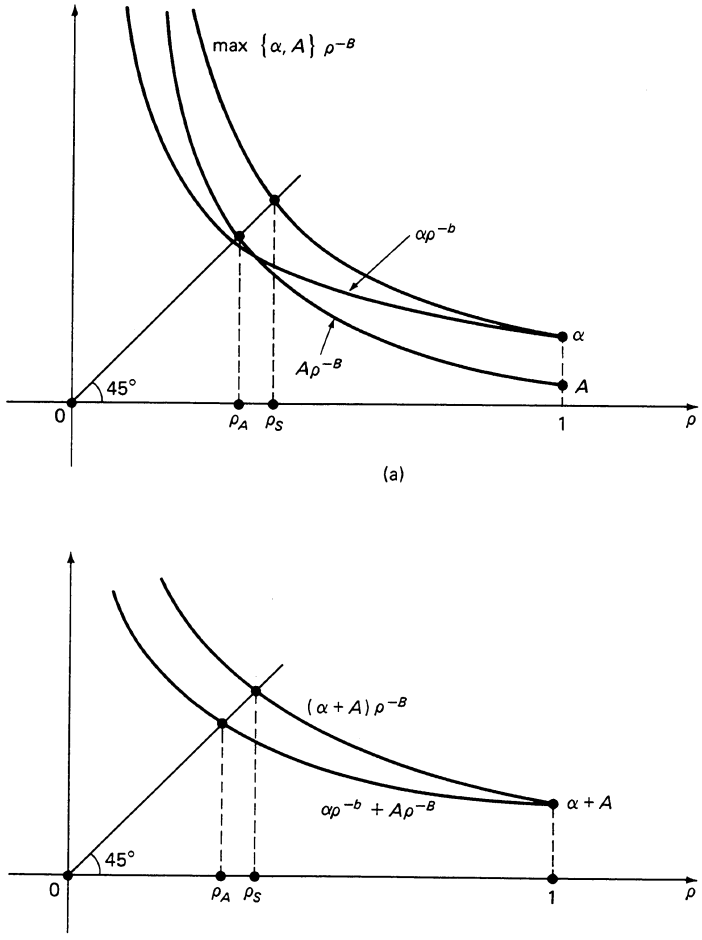



**Figure 6.3.1** Comparison of the convergence rates of the asynchronous iteration (3.12) and its synchronous counterpart under (a) Assumption 3.1 and (b) Assumption 3.2. The synchronous algorithm executes an iteration every $(B + 1)$ time units, after the results of all updates from the previous iteration 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 $\Delta$ 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 $k$th 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

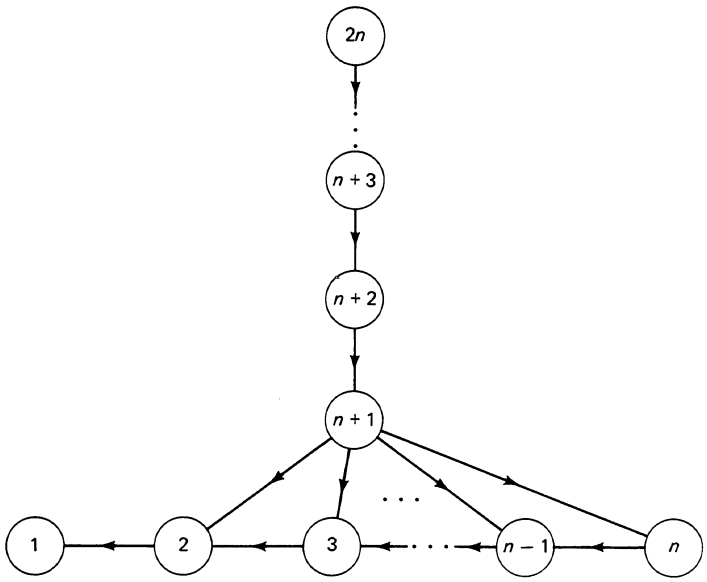



**Figure 6.4.1** An example where the asynchronous version of the Bellman–Ford algorithm requires many more message transmissions than a synchronous version (from [BeG87]). All arc lengths are unity. The initial estimates of shortest distance of all nodes are $\infty$. Consider the following sequence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

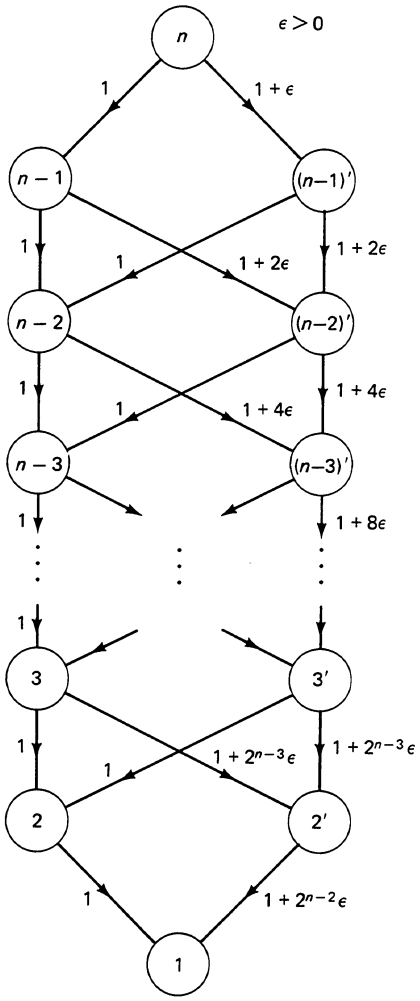



**Figure 6.4.2** An example of a shortest path problem where the asynchronous Bellman–Ford algorithm requires an exponentially large number of messages for termination starting from the initial condition $x_i(0) = \infty$, $i \neq 1$ (due to E. M. Gafni and R. G. Gallager, a private communication). The 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 $\infty$.

**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 $k$th 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 $\epsilon$–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 $\epsilon$–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 $\epsilon$–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 $\epsilon$–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 $\epsilon$–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 $\epsilon^+$–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 $\epsilon^-$–balanced. But this is impossible since, when $p_i$ and $p_j$ converge, arc $(i,j)$ cannot become both $\epsilon^+$–balanced and $\epsilon^-$–balanced an infinite number of times.

The second possibility is that there is a nonempty subset of nodes $N^\infty$ whose prices increase to $\infty$. 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 $N^\infty$ is a strict subset of $N$. Since the algorithm maintains $\epsilon$–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 $N^\infty$ have nonnegative surplus, and each must have positive surplus infinitely often. Adding Eq. (5.1) for all $i$ in $N^\infty$, 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 $\xi$ that maximize the dual cost along the $i$th 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 $i$th 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 $i$th 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 $i$th 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

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

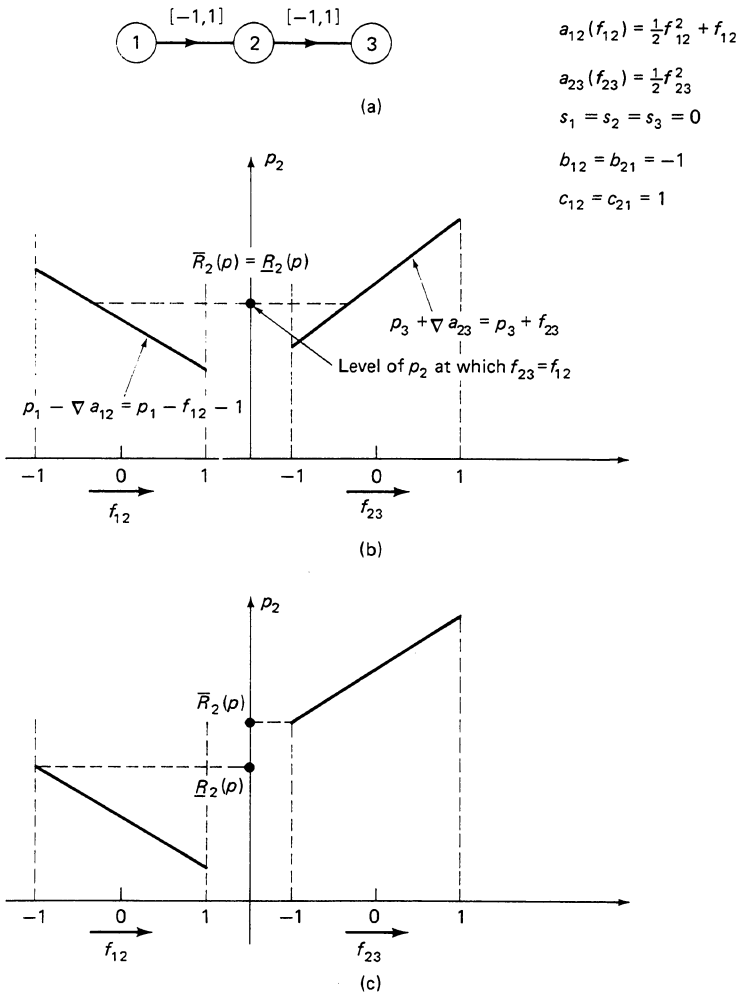



**Figure 6.6.1** Illustration of mappings $\underline{R}_i$, $\overline{R}_i$, and $R_i$. (a) An example problem involving three nodes with zero supplies and two arcs with the arc costs and flow bounds shown. (b) Here $R_2(p)$ consists of a single element. (c) Here $\underline{R}_2(p) < \overline{R}_2(p)$ 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

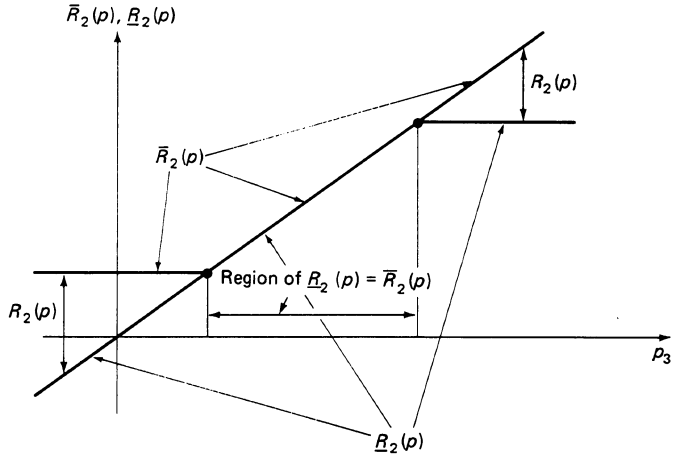



**Figure 6.6.2** Illustration of continuity and monotonicity of $\underline{R}_2(p)$ and $\overline{R}_2(p)$ in the example of Fig. 6.1. The figure shows $\underline{R}_2(p)$, $\overline{R}_2(p)$, and $R_2(p)$ as $p_1$ is fixed and $p_3$ varies.

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

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

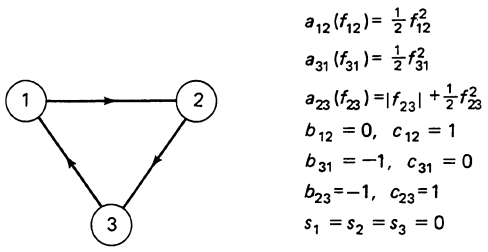
$$\nabla q_{ij}(p_i' - p_j') = -f_{ij}^*, \qquad \forall\, (i,j) \in A \text{ with } i \in I,\ j \in I, \tag{6.29}$$

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

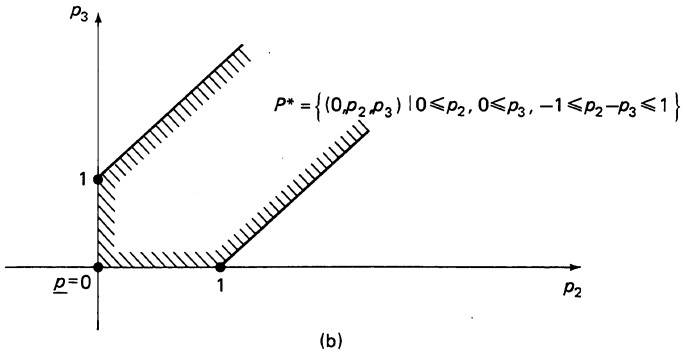
By combining Eqs. (6.26) through (6.30) we obtain $\nabla q_{ij}(\hat{p}_i - \hat{p}_j) = -f_{ij}^*$ for all $(i,j) \in A$. Therefore $\hat{p} \in P^*$.

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

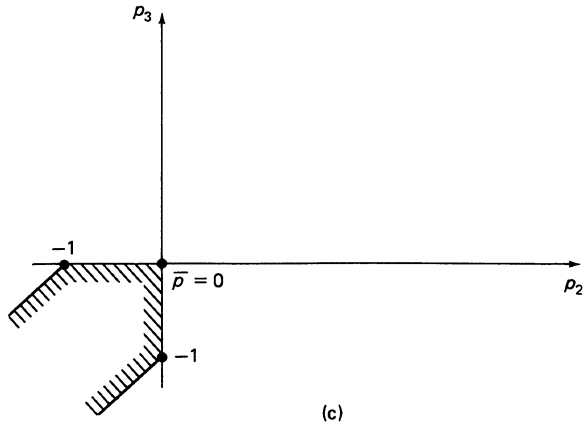
Figures 6.6.3 and 6.6.4 provide examples of problems where $P^*$ is bounded above and/or below and illustrates the maximal and/or minimal elements of $P^*$. Exercise 6.1 provides conditions under which boundedness above and/or below of $P^*$ can be verified. A typical case where $P^*$ is unbounded above 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.

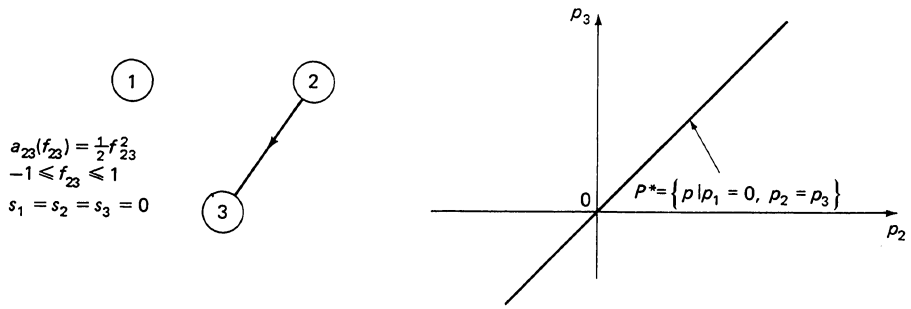



**Figure 6.6.4** An example where the reduced dual optimal solution set $P^*$ is not bounded below or above.

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 $i$th 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.**

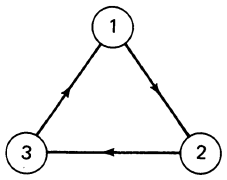



**Figure 6.6.5** A three node network example. The arc cost functions and the lower and upper arc flow bounds are

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

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

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

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

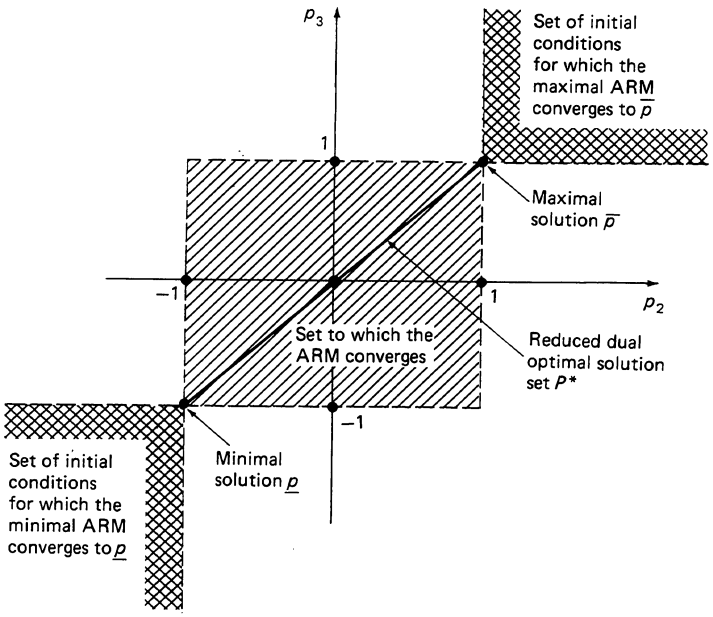



**Figure 6.6.6** The structure of the reduced dual optimal solution set, the convergence regions of the ARM, the maximal ARM, and the minimal ARM for the example of Fig. 6.6.5.

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 $i$th 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 $i$th 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 $i$th subsystem, and the matrices $B_{ij}(t)$ are meant to describe the interaction of the $i$th 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 $i$th 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 $i$th subsystem. (This step is called *relaxing* the $i$th 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 $i$th 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 $i$th 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 $i$th diagonal element, and let $|B|$ be a matrix whose $ij$th 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 $p$th component of any vector $x_i \in R^{n_i}$. Accordingly, let $D_{i,pq}(t)$, $B_{ij,pq}(t)$ be the $pq$th 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 $i$th 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 $i$th subsystem. If we have initial conditions for the $i$th 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].

# 7

# Partially Asynchronous Iterative Methods

In Chapter 6, we studied asynchronous algorithms under minimal interprocessor synchronization assumptions. We merely required that no processor quits forever. In particular, we allowed the possibility of arbitrarily large communication delays and arbitrary differences in the frequency of computation of different processors. We saw that excessive asynchronism can be detrimental to the convergence of some natural algorithms, such as the gradient algorithm (Example 3.1 in Subsection 6.3.2). It turns out that the desired convergence properties of several interesting algorithms are recovered, provided that certain bounds are imposed on the amount of asynchronism present in the execution of the algorithm, and this is the subject of this chapter. In particular, we shall assume that there exists a constant $B$ (which we call the *asynchronism measure*) such that:

**(a)** Each processor performs an update at least once during any time interval of length $B$.

**(b)** The information used by any processor is outdated by at most $B$ time units.

An asynchronous algorithm satisfying these two conditions will be called a *partially asynchronous* algorithm, as opposed to the totally asynchronous algorithms of Chapter 6. The convergence analysis of partially asynchronous algorithms cannot be reduced to a single general convergence result such as the Asynchronous Convergence Theorem of

Section 6.2. In fact, it will be seen in this chapter that there are two different types of partially asynchronous algorithms:

**(a)** Algorithms that do not converge totally asynchronously, but their partially asynchronous execution converges for any choice of the asynchronism measure $B$, as long as $B$ is finite.

**(b)** Algorithms that converge partially asynchronously if the asynchronism measure $B$ is small enough, and diverge if $B$ is large.

This chapter can be roughly divided along the above lines. Sections 7.2–7.4 and 7.8 deal with algorithms of the first type, and Sections 7.5 and 7.6 deal with algorithms of the second type. We start in Section 7.1, with a definition of the partially asynchronous algorithmic model, and prove a general result that is sometimes useful in establishing convergence. This result is exploited in Section 7.2, where we prove partially asynchronous convergence of certain iterative algorithms involving nonexpansive mappings (with respect to the maximum norm). Two applications are provided: partially asynchronous convergence is established for a certain class of linear iterative algorithms, as well as for the dual relaxation algorithm for the solution of strictly convex network flow problems. In Section 7.3, we consider an iterative process whereby a set of processors try to reach agreement by receiving tentative values from other processors and forming convex combinations. We then proceed to show that the convergence of the agreement algorithm also implies partially asynchronous convergence of the algorithm $\pi := \pi P$ for computing the invariant distribution of an irreducible Markov chain. In Section 7.4, we consider an algorithm for load balancing in a computer network in which processors keep transferring parts of their own load to their lightly loaded neighbors. In Section 7.5, we study descent (gradient–like) algorithms for unconstrained and constrained optimization. Partially asynchronous convergence is established provided that the stepsize employed is small enough. We will also see that if the value of the asynchronism measure $B$ is increased, a smaller stepsize is required. Thus, for a fixed stepsize, the algorithm converges if $B$ is small enough, but can diverge when $B$ is large. In Section 7.6, we consider the problem of optimal routing in a data communication network, which we formulate as a multicommodity network flow problem (cf. Section 5.6). We prove partially asynchronous convergence of an algorithm of the gradient projection type, under fairly realistic assumptions. In Section 7.7, we refine the model of computation of Section 7.1, by allowing the possibility that several processors update the same component of the vector being iterated. This can result in disagreement between different processors and we employ the scheme of Section 7.3 to eliminate this disagreement. Then, in Section 7.8, we consider asynchronous stochastic gradient–like algorithms. In such algorithms, processors have access to noisy measurements of the gradient of a cost function being optimized; it is then reasonable to have several processors update the same component, in the hope that the effects of the noise are averaged out. Thus, the extended model of computation introduced in Section 7.7 becomes applicable.

Several sections in this chapter can be omitted without any loss of continuity. In particular, the results in each one of Sections 7.2, 7.4, 7.5, and 7.6 are not used elsewhere. Furthermore, the results of Section 7.3 are only used in Sections 7.7 and 7.8.

## 7.1 THE PARTIALLY ASYNCHRONOUS ALGORITHMIC MODEL

In this section, we present a model of partially asynchronous iterative algorithms, provide some illustrative examples, and prove a result that is sometimes useful in establishing convergence. The model we use is the same as the totally asynchronous model of Section 6.1 except that certain bounds are placed on the time between consecutive updates by each processor and on the magnitude of the communication delays. We keep the same notation as in Chapter 6, which we now review.

Let $X_1, X_2, \ldots, X_n$ be given subsets of Euclidean spaces and let $X = X_1 \times X_2 \times \cdots \times X_n$ be their Cartesian product. Accordingly, elements of $X$ are decomposed into block–components, that is, $x = (x_1, x_2, \ldots, x_n)$, with $x_i \in X_i$. We are given functions $f_i : X \mapsto X_i$, $i = 1, \ldots, n$, and we consider the asynchronous iteration

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

An execution of this iteration is completely determined by the following:

**(a)** Initial conditions $x(t) \in X$ for each $t \leq 0$.

**(b)** A set $T^i$ of times at which the $i$th component $x_i$ is updated.

**(c)** A variable $\tau_j^i(t)$ for each $i$ and $j$, and each $t \in T^i$. These variables determine the amount by which the information used in an update of $x_i$ is outdated and satisfy $0 \leq \tau_j^i(t) \leq t$. For the purposes of mathematical analysis, it is sometimes convenient to allow $\tau_j^i(t)$ to be negative; this is the reason why we require initial conditions $x(t)$ for negative times.

The equations describing the algorithm are, for $t \geq 0$:

$$x_i(t+1) = x_i(t), \qquad \text{if } t \notin T^i, \tag{1.1}$$

$$x_i(t+1) = f_i\Big(x_1\big(\tau_1^i(t)\big), x_2\big(\tau_2^i(t)\big), \ldots, x_n\big(\tau_n^i(t)\big)\Big), \qquad \text{if } t \in T^i. \tag{1.2}$$

Any particular choice of the sets $T^i$ and the values of the variables $\tau_j^i(t)$ will be called a *scenario*. It is seen that for any fixed scenario, the value of $x(t)$, for $t > 0$, is uniquely determined by the initial conditions. However, the values obtained under different scenarios could be very different. The assumption that follows is a restriction on the set of scenarios under consideration and should be contrasted with the Total Asynchronism Assumption of Section 6.1.

**Assumption 1.1.** (*Partial Asynchronism*) There exists a positive integer $B$ such that:

**(a)** For every $i$ and for every $t \geq 0$, at least one of the elements of the set $\{t, t+1, \ldots, t+B-1\}$ belongs to $T^i$.

**(b)** There holds

$$t - B < \tau_j^i(t) \leq t, \tag{1.3}$$

for all $i$ and $j$, and all $t \geq 0$ belonging to $T^i$.

(c) There holds $\tau_i^j(t) = t$ for all $i$ and $t \in T^i$.

The partial asynchronism assumption is often easy to enforce in a practical implementation. It is by no means assumed that the time variable $t$ is accessible to the processors; rather, $t$ should be viewed as a time variable measured by an external observer. In fact, $t$ need not correspond to real time; it could simply be an artificial variable used to index the events of interest (e.g., the times at which some variables are updated). The following example demonstrates that Assumption 1.1 is typically easy to satisfy in message–passing systems.

**Example 1.1.**

Let $\tau$ denote the true (global) time. Suppose that the value of $\tau$ is unknown to the processors and that each processor maintains its own clock. Let $t_i(\tau)$ be the reading of the clock of processor $i$ when $\tau$ is the true time. Let us now assume that the local clocks can be neither arbitrarily fast nor arbitrarily slow compared to the true time. In particular, we assume that

$$A_1|\tau - \tau'| \leq |t_i(\tau) - t_i(\tau')| \leq A_2|\tau - \tau'|, \qquad \forall \tau, \tau', \ \forall i,$$

where $A_1$ and $A_2$ are some positive constants. Suppose that an update by any processor takes anywhere between $A_3$ and $A_4$ time units, and that a processor can remain idle up to $A_5$ time units before initiating the next update. Here $A_3$, $A_4$, and $A_5$ are positive constants and time is measured by the local clock of the processor under consideration. Finally, suppose that the local time difference between the initiation of two consecutive broadcasts of any variable $x_i$ is bounded by some $A_6$ and that such a broadcast takes up to $A_7$ global time units to be completed.

We introduce an index variable $t$ that is incremented by 1 whenever some processor has completed a variable update, and we identify it with the time variable in the model of Eqs. (1.1) and (1.2). Under these assumptions, it is not hard to verify that the mathematical description (1.1)–(1.2) of the algorithm satisfies parts (a) and (b) of Assumption 1.1 (Exercise 1.1).

Part (c) of Assumption 1.1 is very natural and holds automatically in message–passing systems where the $i$th processor maintains the latest version of $x_i$ and is the only processor that is allowed to update $x_i$. In shared memory systems, however, this assumption is not necessarily valid although it can be enforced quite easily (recall also the discussion of Example 1.2 in Section 6.1). It will be seen (Exercise 2.1 in the next section) that if Assumption 1.1(c) is violated, an otherwise convergent algorithm can diverge.

We continue with two examples that illustrate the different types of convergence behavior exhibited by partially asynchronous algorithms.

**Example 1.2.** *Convergence for All Values of the Asynchronism Measure*

We consider the linear iteration $x := Ax$, where $A$ is a $2 \times 2$ matrix given by

$$A = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

It is seen that the set $X^*$ of fixed points of the mapping $f(x) = Ax$ is the set of all vectors $(x_1, x_2) \in \Re^2$ such that $x_1 = x_2$. Notice that the synchronous iteration $x(t+1) = Ax(t)$ converges in a single step to the vector $x = (y, y)$, where $y = (x_1 + x_2)/2$.

We consider the following scenario. There are two processors $i = 1, 2$, and each processor $i$ updates the variable $x_i$ at each time step. At certain times $t_1, t_2, \ldots$, each processor transmits its value, which is received with zero delay and is immediately incorporated into the computations of the other processor. We then have

$$x_1(t+1) = \frac{x_1(t)}{2} + \frac{x_2(t_k)}{2}, \qquad t_k \leq t < t_{k+1},$$

$$x_2(t+1) = \frac{x_1(t_k)}{2} + \frac{x_2(t)}{2}, \qquad t_k \leq t < t_{k+1}.$$

Thus,

$$x_1(t_{k+1}) = (\tfrac{1}{2})^{t_{k+1}-t_k} x_1(t_k) + \left(1 - (\tfrac{1}{2})^{t_{k+1}-t_k}\right) x_2(t_k),$$

$$x_2(t_{k+1}) = (\tfrac{1}{2})^{t_{k+1}-t_k} x_2(t_k) + \left(1 - (\tfrac{1}{2})^{t_{k+1}-t_k}\right) x_1(t_k).$$

Subtracting these two equations, we obtain

$$|x_2(t_{k+1}) - x_1(t_{k+1})| = \left(1 - 2(\tfrac{1}{2})^{t_{k+1}-t_k}\right)|x_2(t_k) - x_1(t_k)| = (1 - \epsilon_k)|x_2(t_k) - x_1(t_k)|,$$

where $\epsilon_k = 2(\frac{1}{2})^{t_{k+1}-t_k}$. In particular, the quantity $|x_2(t_k) - x_1(t_k)|$ keeps decreasing. Nevertheless, convergence to a vector in $X^*$ is not guaranteed unless $\prod_{k=1}^{\infty}(1 - \epsilon_k) = 0$, which is not necessarily the case. For example, it can be shown that $\prod_{k=1}^{\infty}(1 - k^{-2}) > 0$, and if we choose the differences $t_{k+1} - t_k$ to be large enough so that $\epsilon_k < k^{-2}$, then convergence to a vector in $X^*$ does not take place. (See Fig. 7.1.1 for an illustration.) This shows that the iteration $x := Ax$ does not converge totally asynchronously. On the other hand, if the partial asynchronism assumption is imposed, we must have $t_{k+1} - t_k \leq B$ in the scenario just described. We then obtain $\epsilon_k > 1/2^B$ and $\prod_{k=1}^{\infty}(1 - \epsilon_k) = 0$, which proves that $x_1(t_k) - x_2(t_k)$ converges to zero, and the sequence $\{x(t_k)\}$ converges to the set $X^*$. This conclusion will be generalized in Section 7.3, where convergence is proved for every partially asynchronous scenario.

This example is typical of algorithms that converge partially asynchronously for every choice of $B$, but do not converge totally asynchronously. In such algorithms, there is usually a function that measures distance from the set $X^*$ of fixed points and that is guaranteed to decrease once in a while. However, the factor by which this distance function decreases is reduced when the communication delays increase. We then need to assume that communication delays are bounded in order to guarantee that the distance function decreases at a fixed rate.

**Example 1.3.** *Convergence Only When the Asynchronism Measure is Small*

Consider the linear iteration $x := x - \gamma Ax$, where $\gamma$ is a small positive stepsize, the matrix $A$ is given by

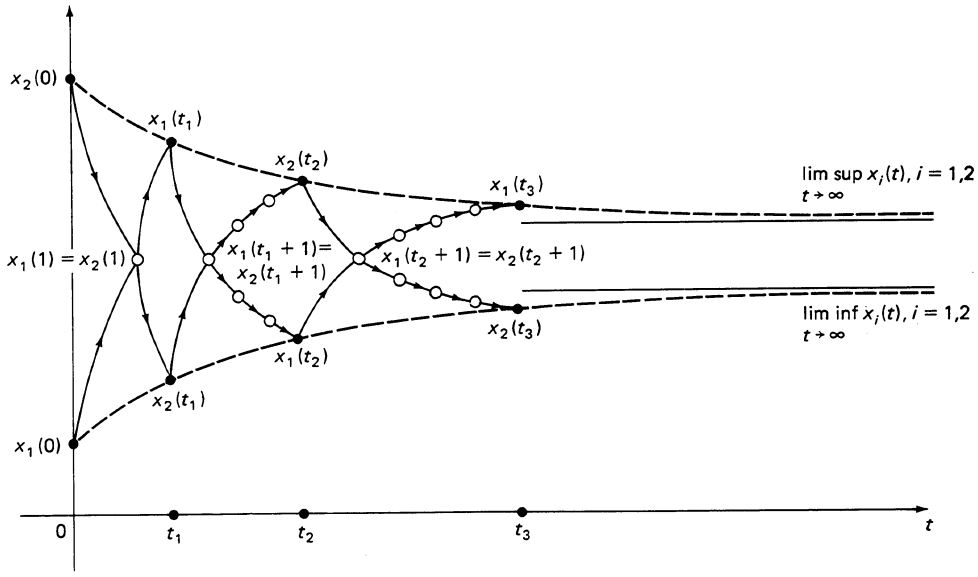



**Figure 7.1.1** Nonconvergence under a totally asynchronous scenario in Example 1.2. Here, for each integer $t$ in the range $[t_k, t_{k+1})$, each processor $i$ averages its value $x_i(t)$ with the value $x_j(t_k)$ of the other processor $j$ computed $t - t_k$ time units earlier. The difference $|x_1(t_k) - x_2(t_k)|$ is monotonically decreasing, but does not converge to zero if the intervals $[t_k, t_{k+1}]$ are increasingly large.

$$A = \begin{bmatrix} 1+\epsilon & 1 & 1 \\ 1 & 1+\epsilon & 1 \\ 1 & 1 & 1+\epsilon \end{bmatrix} = \begin{bmatrix} \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} [1 \quad 1 \quad 1],$$

and $\epsilon$ is a scalar satisfying $0 < \epsilon < 1$. (Notice that $A$ is positive definite.) This iteration is the gradient algorithm for minimizing the quadratic cost function $\frac{1}{2}x'Ax$ and, if $\gamma$ is small enough, it converges synchronously (Chapters 2 and 3). The present example is the same as Example 3.1 of Section 6.3.2, where it was shown that the corresponding totally asynchronous iteration does not converge. We now consider a partially asynchronous scenario whereby each processor performs an iteration at each time step and information is exchanged once every $B$ time units. We let $t_k = kB$, and the update equation for $x_1$ is given by

$$x_1(t+1) = x_1(t) - \gamma\big((1+\epsilon)x_1(t) + x_2(t_k) + x_3(t_k)\big), \qquad t_k \le t < t_{k+1}. \tag{1.4}$$

Similar equations are valid for $x_2$ and $x_3$. We solve the difference equation (1.4) to obtain

$$x_1(t_{k+1}) = \big(1 - \gamma(1+\epsilon)\big)^B x_1(t_k) - \frac{1 - \big(1 - \gamma(1+\epsilon)\big)^B}{1+\epsilon}\big(x_2(t_k) + x_3(t_k)\big),$$

and similar equations hold for $x_2(t_k)$ and $x_3(t_k)$. We conclude that for this particular scenario, we have

$$x(t_{k+1}) = Cx(t_k),$$

where

$$C = \begin{bmatrix} \alpha & \beta & \beta \\ \beta & \alpha & \beta \\ \beta & \beta & \alpha \end{bmatrix},$$

$$\alpha = \big(1 - \gamma(1+\epsilon)\big)^B, \qquad \beta = -\frac{1 - \big(1 - \gamma(1+\epsilon)\big)^B}{1+\epsilon}.$$

The eigenvalues of $C$ are equal to $\alpha + 2\beta$ and $\alpha - \beta$, the latter with multiplicity two. [This is because $C = (\alpha - \beta)I + \beta ee'$, where $e \in \Re^3$ is the vector with all entries equal to 1. It is then seen that the eigenvalues of $ee'$ are equal to 3 and 0, the latter with multiplicity two.] As $B$ increases to infinity, $\alpha$ approaches 0 and $\beta$ approaches $-1/(1+\epsilon)$, assuming that $\gamma$ is small enough to satisfy $\gamma(1+\epsilon) < 1$. In particular, $\alpha + 2\beta$ approaches $-2/(1+\epsilon)$, which is larger than 1 in magnitude. It follows that for $B$ sufficiently large, we have $\rho(C) > 1$ and the iteration does not converge under the previously considered partially asynchronous scenario. On the other hand, if $B$ is a moderate number and $\gamma$ is very small, we have $\alpha \approx 1 - B\gamma(1+\epsilon)$ and $\beta \approx -B\gamma$, where we have neglected terms of the order of $\gamma^2$ or smaller. Thus, the eigenvalues of $C$ are approximately equal to $1 - B\gamma(3+\epsilon)$ and $1 - B\gamma\epsilon$. Assuming that $\gamma$ is sufficiently small so that $B\gamma(3+\epsilon) < 1$, we see that the eigenvalues of $C$ are smaller than 1 in magnitude and convergence follows for the previously considered scenario.

A different approach for establishing convergence, which is also easier to generalize, is to notice that the differences $x_2(t) - x_2(t_k)$ and $x_3(t) - x_3(t_k)$ for $t_k \le t < t_{k+1}$ are of the order of $\gamma B$. It then follows that the update $x_1(t+1) - x_1(t)$ in Eq. (1.4) differs from the synchronous update $-\gamma\big((1+\epsilon)x_1(t) + x_2(t) + x_3(t)\big)$ by a factor of the order of $B\gamma^2$. Thus, even though the values in Eq. (1.4) are outdated, this can only have effects of the order of $B\gamma^2$. As long as $B\gamma$ is much smaller than 1, the effects of asynchronism are small compared with the magnitude of the updates and the algorithm can be approximated (up to first order terms in $\gamma$) with the synchronous algorithm that is known to be convergent. On the other hand, if $B\gamma$ is larger than 1, then the effect of asynchronism on an update becomes comparable to the magnitude of that update and is therefore big enough to cause divergence. Although this argument is somewhat sketchy, it will be made rigorous in Section 7.5.

This example is representative of algorithms that converge partially asynchronously when $B$ is small and can diverge when $B$ is large. A typical property of such algorithms is that the effects of asynchronism are negligible when delays are small, but can become significant when delays are large. Such a property is naturally present in algorithms that employ a small stepsize.

We continue by developing some background on the structure of partially asynchronous iterations. An important consequence of Eqs. (1.1) and (1.2) and Assumption 1.1 is that the value of $x(t+1)$ depends only on $x(t), x(t-1), \ldots, x(t-B+1)$, and not on any earlier information $x(\tau)$, $\tau \le t - B$. Thus, old information is purged from the system after at most $B$ time units, and this suggests that we introduce the vector

$$z(t) = \big(x(t), x(t-1), \ldots, x(t-B+1)\big),$$

which summarizes all unpurged information. From the preceding discussion, for any given scenario, the value of $x(t+1)$ can be determined from knowledge of $z(t)$. It follows that $z(t+1)$ can be determined from $z(t)$. Proceeding inductively, we can see that for any fixed scenario and any positive integer $s$, the value of $z(t+s)$ is uniquely determined by $z(t)$. Another important observation is that if the function $f$ is continuous, then for any fixed scenario and any $s > 0$, $z(t+s)$ is a continuous function of $z(t)$.

We denote by $X^*$ the set $\{x \in X \mid x = f(x)\}$ of fixed points of $f$, and by $Z$ the Cartesian product of $B$ copies of $X$. We also use $Z^*$ to denote the set of all elements of $Z$ of the form $(x^*, x^*, \ldots, x^*)$, where $x^*$ is an arbitrary element of $X^*$. It is not hard to see that if $z(t) = z^* \in Z^*$, then $z(t+s) = z^*$ for every $s > 0$ and for every scenario.

The result that follows is sometimes useful in establishing convergence of a partially asynchronous algorithm to $X^*$ and will be invoked in Section 7.2. It involves a distance function $d : Z \mapsto [0, \infty)$, which measures the progress of $z(t)$ toward the set $Z^*$ and an assumption that this function has to decrease once in a while. In this respect, Prop. 1.1 has the flavor of a Lyapunov stability theorem and can be successfully applied only if a suitable function $d$ is available.

**Proposition 1.1.** (*Lyapunov Theorem*) Consider the asynchronous iteration (1.1)–(1.2). Suppose that $f$ is continuous and that Assumption 1.1 (partial asynchronism) holds. Suppose also that there exists some positive integer $t^*$ and a continuous function $d : Z \mapsto [0, \infty)$ with the following properties:

**(a)** For every $z(0) \notin Z^*$ and for every scenario, we have $d\big(z(t^*)\big) < d\big(z(0)\big)$.

**(b)** For every $z(0) \in Z$, for every $t \geq 0$, and for every scenario, we have $d\big(z(t+1)\big) \leq d\big(z(t)\big)$.

Then, we have $z^* \in Z^*$ for every limit point $z^* \in Z$ of the sequence $\{z(t)\}$.

***Proof.*** By the continuity of $f$ and our earlier discussion, $z(t^*)$ is a continuous function of $z(0)$ for every scenario. Using the continuity of $d$, $d\big(z(t^*)\big)$ is also a continuous function of $z(0)$ for every scenario. We define a function $h : Z \mapsto \Re$ by letting

$$h\big(z(0)\big) = \max\big\{d\big(z(t^*)\big) - d\big(z(0)\big)\big\}, \tag{1.5}$$

where the maximum is taken over all possible scenarios. Although there are infinitely many possible scenarios, if we consider the iteration over a time interval of $t^*$ units, there are only a finite number of choices. This is because we have to deal with only a finite number of variables $\tau_j^i(t)$, $0 \leq t \leq t^*$, and because each one of these variables can take a finite number $B$ of different values. Therefore, the maximum in the definition of $h$ is over a finite number of choices. For each such choice, $d\big(z(t^*)\big) - d\big(z(0)\big)$ is a continuous function of $z(0)$. It is not hard to see that the maximum of a finite number of continuous functions is continuous. It follows that $h$ is continuous. Furthermore, for every scenario and every $z(0) \notin Z^*$, we have $d\big(z(t^*)\big) - d\big(z(0)\big) < 0$, from which it follows that $h(z) < 0$ for every $z \notin Z^*$.

Notice that if there exists a scenario that starts with some $z(0) = z$ at time 0 and generates some $z(t^*) = \tilde{z}$ at time $t^*$, then there also exists a scenario that starts with $z(t) = z$ at time $t$ and produces $z(t + t^*) = \tilde{z}$ at time $t + t^*$. The converse is also true. For this reason, the function $h$ defined by Eq. (1.5) is also given by

$$h\big(z(t)\big) = \max\big\{d\big(z(t + t^*)\big) - d\big(z(t)\big)\big\}, \tag{1.6}$$

where the maximum is again taken over all scenarios.

Let us now fix a scenario and a choice of $z(0)$. From condition (b) in the statement of the proposition, the sequence $\big\{d\big(z(t)\big)\big\}$ is nonincreasing. Furthermore, it is bounded below by zero and, therefore, converges to a limit that will be denoted by $d^*$. Let $z^* \in Z$ be a limit point of the sequence $\{z(t)\}$ and choose a sequence $\{t_k\}$ of times such that $z(t_k)$ converges to $z^*$. From Eq. (1.6) and the continuity of $h$,

$$d^* = \lim_{k\to\infty} d\big(z(t_k + t^*)\big) \leq \lim_{k\to\infty} d\big(z(t_k)\big) + \lim_{k\to\infty} h\big(z(t_k)\big) = d^* + h(z^*).$$

Since, as observed earlier, we have $h(z) < 0$ for all $z \notin Z^*$, it follows that $z^* \in Z^*$. **Q.E.D.**

Proposition 1.1 does not deal with convergence rates and is in fact too general to have useful convergence rate implications. However, it can be shown (Exercise 1.2) that for the case of linear iterative algorithms, convergence is guaranteed to be geometric.

## EXERCISES

**1.1.** Show that for a sufficiently large $B$, parts (a) and (b) of Assumption 1.1 are satisfied in the context of Example 1.1.

**1.2.** **(Geometric Convergence of Linear Partially Asynchronous Iterations.)** Consider a partially asynchronous iteration in which $X = \Re^n$ and the iteration function $f$ is of the form $f(x) = Ax$, where $A$ is an $n \times n$ matrix. Notice that the set $X^*$ of fixed points of $f$ is a subspace of $\Re^n$. Let $Z$ and $Z^*$ be as defined in this section, and let $\|\cdot\|$ be some vector norm on $Z$. For any $z \in Z$, we define $d(z) = \inf_{z^* \in Z^*} \|z - z^*\|$.

- **(a)** Show that the infimum in the definition of $d$ is attained and that the function $d$ is continuous.
- **(b)** Show that $d(cz) = cd(z)$ and $d(z - z^*) = d(z)$ for every $z \in Z$, $z^* \in Z^*$, and every positive scalar $c$.
- **(c)** Suppose that conditions (a) and (b) of Prop. 1.1 are satisfied. Show that $d\big(z(t)\big)$ converges to zero geometrically.
- **(d)** Extend the result of part (b) to the case where $f$ is of the form $f(x) = Ax + b$, where $b$ is a given vector in $\Re^n$, assuming that the function $f$ has a fixed point.

*Hints:* For part (a), see the proof of Prop. 2.1 in the next section. For part (c), we let $S(z^*) = \big\{z \in Z \mid d(z) = \|z - z^*\|\big\}$ for any $z^* \in Z^*$. We also let $R(c) = \{z \in Z \mid d(z) = c\}$ and

$$\rho = \sup \frac{d\big(z(t^*)\big)}{d\big(z(0)\big)}, \tag{1.7}$$

where the supremum is over all $z(0) \in S(0) \cap R(1)$ and all scenarios. Use the continuity of $d$ and the compactness of $S(0) \cap R(1)$ to show that $\rho < 1$. Then notice that for any scenario, the mapping that determines $z(t)$ as a function of $z(0)$ is linear. Show that the value of $\rho$ remains the same if the supremum in Eq. (1.7) is taken over all $z(0) \in S(0)$ such that $z(0) \notin Z^*$. Next, notice that if $z^* \in Z^*$ and $z(0)$ is changed to $z(0) - z^*$, then for any fixed scenario, $z(t^*)$ is changed to $z(t^*) - z^*$. Use the property $d(z) = d(z - z^*)$ to conclude that $\rho$ remains the same if the supremum in Eq. (1.7) is taken over all $z(0) \notin Z^*$. For part (d), use a change of variables to replace $b$ by the zero vector.

**1.3.** Formulate and prove an extension of Prop. 1.1 for the case where Eq. (1.2) is generalized to

$$x_i(t+1) = f_i\Big(x_1\big(\tau_1^i(t)\big), x_2\big(\tau_2^i(t)\big), \ldots, x_n\big(\tau_n^i(t)\big), \theta_i(t)\Big), \qquad \text{if } t \in T^i.$$

Here each $\theta_i(t)$ is a parameter belonging to a compact set $\Theta$. *Hint:* The maximum in Eq. (1.5) has to be taken over all choices of the parameters $\theta_i(t)$.

## 7.2 ALGORITHMS FOR FIXED POINTS OF NONEXPANSIVE MAPPINGS

We consider a partially asynchronous iteration of the form $x := f(x)$, where $f : \Re^n \mapsto \Re^n$. We let $X^* = \{x \in \Re^n \mid f(x) = x\}$ be the set of fixed points of $f$, and we study the case where $f$ has the following properties:

**Assumption 2.1.**

**(a)** The set $X^*$ is nonempty.

**(b)** The function $f$ is continuous.

**(c)** The function $f$ is *nonexpansive*, that is, it satisfies

$$\|f(x) - x^*\|_\infty \leq \|x - x^*\|_\infty, \qquad \forall x \in \Re^n, \ \forall x^* \in X^*. \tag{2.1}$$

We develop a general convergence result (Prop. 2.3) which we then apply to two specific problems: solution of systems of equations involving a weakly diagonally dominant matrix (Subsection 7.2.2) and nonlinear convex network flow problems (Subsection 7.2.3).

### 7.2.1 A Convergence Result

For any $x \in \Re^n$, we denote by $g(x)$ the distance of $x$ from $X^*$, defined by

$$g(x) = \inf_{x^* \in X^*} \|x - x^*\|_\infty. \tag{2.2}$$

The following preliminary result will be needed in the sequel.

**Proposition 2.1.** Suppose that $f : \Re^n \mapsto \Re^n$ satisfies Assumption 2.1. Then:

**(a)** The set $X^*$ is closed.

**(b)** For every $x \in \Re^n$, there exists some $x^* \in X^*$ such that $g(x) = \|x - x^*\|_\infty$.

**(c)** The function $g : X \mapsto \Re^n$, is continuous.

**(d)** For every $x \in \Re^n$ we have $g\big(f(x)\big) \leq g(x)$.

***Proof.***

**(a)** Let $\{x^k\}$ be a sequence of elements of $X^*$ converging to some $x^*$. Using the continuity of $f$, we have

$$f(x^*) = \lim_{k\to\infty} f(x^k) = \lim_{k\to\infty} x^k = x^*,$$

and $x^*$ belongs to $X^*$. This shows that the set $X^*$ contains all of its limit points and is therefore closed.

**(b)** Let us fix some $x \in \Re^n$, and let $y^*$ be an arbitrary element of $X^*$. A definition equivalent to Eq. (2.2) is

$$g(x) = \inf_{\{x^* \in X^* \mid \|x-x^*\|_\infty \leq \|x-y^*\|_\infty\}} \|x - x^*\|_\infty.$$

The set over which this minimization is carried out is the intersection of the closed set $X^*$ with a closed and bounded set. It is therefore itself closed and bounded. Furthermore, the function being minimized is continuous and it follows (Prop. A.8 in Appendix A) that the infimum is attained at some element of $X^*$.

**(c)** Let $x, y \in \Re^n$ and let $x^* \in X^*$ be such that $\|x - x^*\|_\infty = g(x)$. Then

$$g(y) \leq \|y - x^*\|_\infty \leq \|y - x\|_\infty + \|x - x^*\|_\infty = \|y - x\|_\infty + g(x).$$

A similar argument shows that $g(x) \leq \|x - y\|_\infty + g(y)$. Thus, $|g(y) - g(x)| \leq \|y - x\|_\infty$, which shows that $g$ is continuous.

**(d)** Let $x \in \Re^n$ and let $x^*$ be such that $\|x - x^*\|_\infty = g(x)$. Then

$$g\big(f(x)\big) \leq \|f(x) - x^*\|_\infty \leq \|x - x^*\|_\infty = g(x).$$

**Q.E.D.**

We introduce some more notation. Given any $x \in \Re^n$ and $x^* \in X^*$ we let $I(x; x^*)$ be the set of indices of coordinates of $x$ that are farthest away from $x^*$. That is,

$$I(x; x^*) = \big\{i \mid |x_i - x_i^*| = \|x - x^*\|_\infty\big\}. \tag{2.3}$$

Notice that $I(x;x^*)$ is always nonempty. We also let

$$U(x;x^*) = \Big\{ y \in \Re^n \;\Big|\; y_i = x_i \text{ for all } i \in I(x;x^*), \text{ and } |y_i - x_i^*| < \|x - x^*\|_\infty \text{ for all } i \notin I(x;x^*) \Big\}. \tag{2.4}$$

Notice that $I(x;x^*) = I(y;x^*)$ and $\|x - x^*\|_\infty = \|y - x^*\|_\infty$ for every $y \in U(x;x^*)$. Furthermore, $x \in U(x;x^*)$. Loosely speaking, $U(x;x^*)$ is the set of all vectors $y$ with $\|y - x^*\|_\infty = \|x - x^*\|_\infty$ that agree with $x$ in the components that are farthest away from $x^*$ (see Fig. 7.2.1).

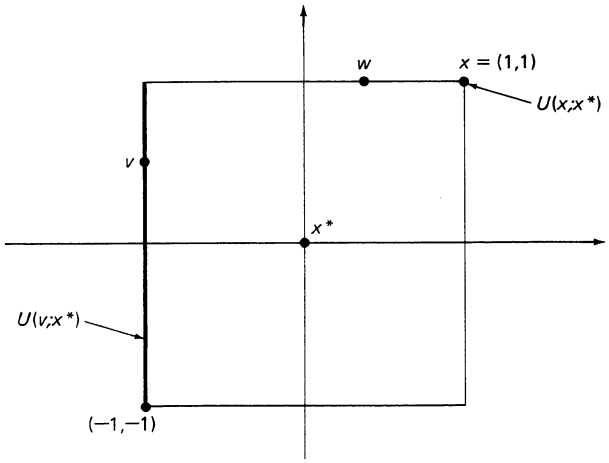



**Figure 7.2.1** Illustration of the sets $I(\cdot\,;x^*)$ and $U(\cdot\,;x^*)$. Let $n = 2$ and suppose that $x^* = (0,0) \in X^*$. For the indicated points $x$, $v$, and $w$, we have $I(x;x^*) = \{1,2\}$, $I(v;x^*) = \{1\}$, $I(w;x^*) = \{2\}$. The set $U(v;x^*)$ is the set of all vectors of the form $(-1,c)$, where $c$ satisfies $-1 < c < 1$, which is the segment joining the points $(-1,-1)$ and $(-1,1)$, the endpoints excluded. Similarly, $U(w;x^*) = \{(c,1) \mid -1 < c < 1\}$. Finally, we have $U(x;x^*) = \{x\}$.

Our convergence result uses the following assumption on $f$.

**Assumption 2.2.**

**(a)** The set $X^*$ is convex.

**(b)** For every $x \in \Re^n$ and $x^* \in X^*$ such that $\|x - x^*\|_\infty = g(x) > 0$, there exists some $i \in I(x;x^*)$ such that $f_i(y) \neq y_i$ for all $y \in U(x;x^*)$.

**(c)** If $x \in \Re^n$, $f_i(x) \neq x_i$, and $x^* \in X^*$, then $|f_i(x) - x_i^*| < \|x - x^*\|_\infty$.

We interpret Assumption 2.2(b). Consider some $x \notin X^*$. Then $f(x) \neq x$, and there exists some $i$ such that $f_i(x) \neq x_i$. Assumption 2.2(b) imposes the additional requirement that such an $i$ can be found among the set of worst indices, that is, $i$ belongs to the set $I(x;x^*)$ of indices corresponding to components farthest away from a closest element of $X^*$. Furthermore, if we change some of the components of $x$ to obtain another vector $y \in U(x;x^*)$, we still retain the property $f_i(y) \neq y_i$, for this particular $i$. Assumption 2.2(b) can be difficult to verify, in general, but it will be shown to be true for two interesting problems in the next two subsections.

The following result shows that Assumption 2.2(c) is automatically true for certain algorithms involving a relaxation parameter.

**Proposition 2.2.** Suppose that a function $h : \Re^n \mapsto \Re^n$ satisfies Assumptions 2.1, 2.2(a), and 2.2(b). Let $\gamma \in (0, 1)$. Then the mapping $f : \Re^n \mapsto \Re^n$ defined by

$$f(x) = (1 - \gamma)x + \gamma h(x)$$

satisfies Assumptions 2.1 and 2.2.

***Proof.*** We first notice that $f$ and $h$ have the same set $X^*$ of fixed points. Thus, $f$ satisfies Assumptions 2.1(a) and 2.2(a). Since $h$ is assumed continuous, $f$ is also continuous and satisfies Assumption 2.1(b). Furthermore, for any $x \in \Re^n$ and $x^* \in X^*$, we have

$$|f_i(x)-x_i^*| = \left|(1-\gamma)(x_i-x_i^*)+\gamma\big(h_i(x)-x_i^*\big)\right| \leq (1-\gamma)|x_i-x_i^*|+\gamma\|x-x^*\|_\infty \leq \|x-x^*\|_\infty.$$

Therefore, Eq. (2.1) holds for $f$ and Assumption 2.1(c) is satisfied. Finally, we have $f_i(x) \neq x_i$ if and only if $h_i(x) \neq x_i$; since $h$ satisfies Assumption 2.2(b), so does $f$.

We now fix some $x \in \Re^n$ such that $f_i(x) \neq x_i$ and some $x^* \in X^*$. Since $f_i(x) \neq x_i$, we have $x \notin X^*$, $x \neq x^*$ and $\|x - x^*\|_\infty > 0$. Consider the interval

$$A = \{y_i \mid |y_i - x_i^*| \leq \|x - x^*\|_\infty\}.$$

Clearly, $x_i \in A$. Also, by the nonexpansive property of $h$, we have $|h_i(x) - x_i^*| \leq \|x - x^*\|_\infty$ and $h_i(x)$ also belongs to $A$. Since $f_i(x) \neq x_i$, we also have $h_i(x) \neq x_i$ and $f_i(x)$ is a convex combination of two distinct points in the interval $A$. Since $\gamma \neq 0$ and $\gamma \neq 1$, such a convex combination must lie in the interior of the interval. This shows that $|f_i(x) - x_i^*| < \|x - x^*\|_\infty$ and $f$ satisfies Assumption 2.2(c). **Q.E.D.**

We now explore the consequences of Assumption 2.2. Part (b) states that one of the components of $x$ farthest away from the closest element $x^*$ of $X^*$ will move when the mapping $f$ is applied; part (c) states that when this happens, the new value of that component will be closer to the corresponding component of $x^*$; this guarantees that the algorithm makes positive progress toward the set $X^*$. In fact, a convergence proof for the synchronous iteration $x(t + 1) = f\big(x(t)\big)$ under Assumptions 2.1 and 2.2 is fairly straightforward and the key ideas are illustrated in Fig. 7.2.2. The proof of asynchronous convergence (Prop. 2.3) will follow the same reasoning as in Fig. 7.2.2. A key difference, however, is that once the cardinality of the set $I\big(x(t); x^*\big)$ is reduced, it takes up to $B$ time units for this information to be incorporated into the information available to other processors, and it can take an additional $B$ time units for the other processors to perform an update using this new information. For this reason, the distance function employed in the proof of Prop. 2.3 is guaranteed to decrease once every (approximately) $2nB$ time units, as opposed to $n$ time units in the argument outlined in Fig. 7.2.2. Another

difference is that neither the convexity of $X^*$ nor the full power of Assumption 2.2(b) are used in Fig. 7.2.2, but they play an essential role in the proof of asynchronous convergence.

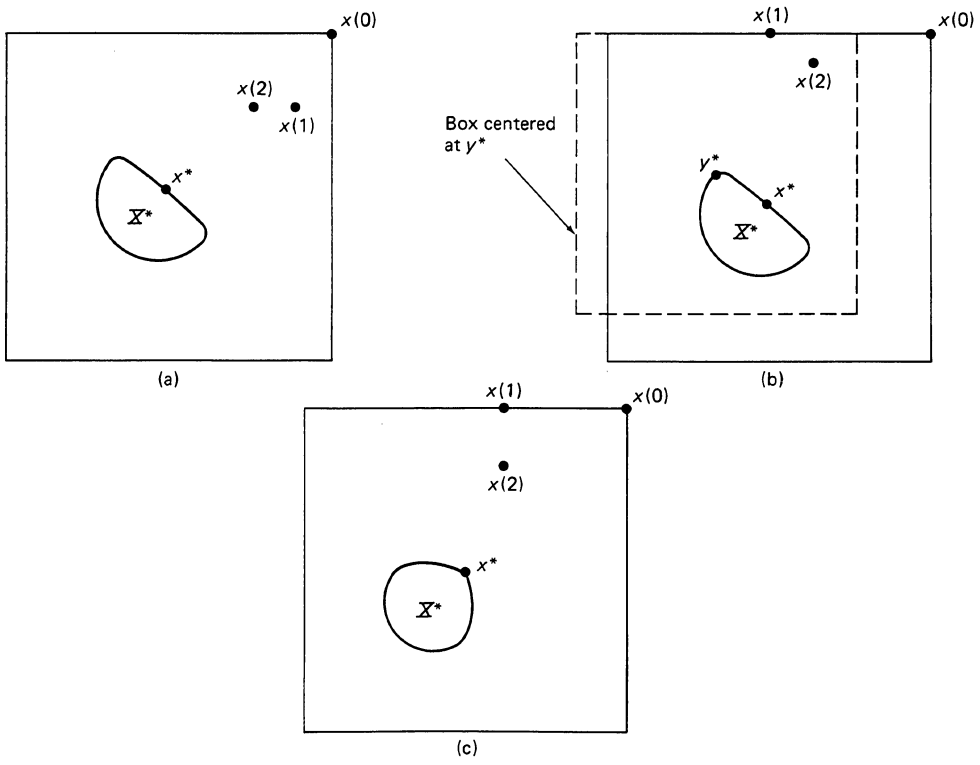



**Figure 7.2.2** Illustration of a convergence proof for the synchronous iteration $x(t+1) = f\big(x(t)\big)$ under Assumptions 2.1 and 2.2. Suppose that $n = 2$ and let $x(0)$, $x^* \in X^*$ be as indicated. Suppose that $g\big(x(0)\big) = \|x(0) - x^*\|_\infty$. Since $x(0) \notin X^*$, there exists some $i \in \{1,2\}$ such that $x_i(1) \neq x_i(0)$. Let us assume that $i = 1$. There are three cases to consider.

**(a)** If $x_2(1) \neq x_2(0)$, then Assumption 2.2(c) implies that $g\big(x(2)\big) \leq g\big(x(1)\big) \leq \|x(1) - x^*\|_\infty < \|x(0) - x^*\|_\infty = g\big(x(0)\big)$.

**(b)** If $x_2(1) = x_2(0)$ and if there exists some $y^* \in X^*$ such that $\|x(1) - y^*\|_\infty < \|x(1) - x^*\|_\infty$, then $g\big(x(2)\big) \leq g\big(x(1)\big) \leq \|x(1) - y^*\|_\infty < \|x(1) - x^*\|_\infty \leq g\big(x(0)\big)$.

**(c)** If $x_2(1) = x_2(0)$ and $\|x(1) - y^*\|_\infty \geq \|x(1) - x^*\|_\infty$ for all $y^* \in X^*$, then $g\big(x(1)\big) = \|x(1) - x^*\|_\infty$. Furthermore, $I\big(x(1); x^*\big) = \{2\}$. It follows from Assumption 2.2(b) that $x_2(2) \neq x_2(1)$ and from Assumption 2.2(c), $|x_2(2) - x_2^*| < \|x(1) - x^*\|_\infty = g\big(x(0)\big)$.

In all three cases, we have $g\big(x(2)\big) < g\big(x(0)\big)$ and convergence follows from Prop. 1.1 of Section 7.1 applied to the function $g$. In the more general case of $n$–dimensional problems, a similar argument would yield $g\big(x(n)\big) < g\big(x(0)\big)$.

**Proposition 2.3.** Suppose that $f : \Re^n \mapsto \Re^n$ satisfies Assumptions 2.1 and 2.2. Suppose that Assumption 1.1 (partial asynchronism) holds. Then the sequence $\{x(t)\}$ generated by the asynchronous iteration $x := f(x)$ converges to some element of $X^*$.

***Proof.*** As in Section 7.1, we consider vectors of the form $z = (x^1, \ldots, x^B)$, where each $x^i$ belongs to $\Re^n$, and we let $Z$ be the set of all such vectors. We also let $Z^* = \{(x^*, \ldots, x^*) \in Z \mid x^* \in X^*\}$. Given any $z = (x^1, \ldots, x^B)$ and $x^* \in X^*$, we let

$$D(z; x^*) = \max_{1 \leq i \leq B} \|x^i - x^*\|_\infty$$

and

$$d(z) = \inf_{x^* \in X^*} D(z; x^*).$$

Our intention is to apply Prop. 1.1 of Section 7.1 to the function $d$. We start by noticing that the function $d$ is continuous and that the infimum in its definition is attained at some $x^* \in X^*$. (The proof of these assertions is identical to the proof of Prop. 2.1.)

We continue with a useful property of the function $d$, illustrated in Fig. 7.2.3.

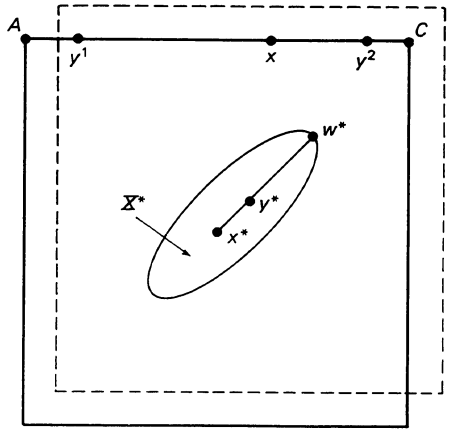



**Figure 7.2.3** Illustration of the proof of Lemma 2.1. Let $X^*$, $x$, and $x^*$ be as indicated. Let $B = 2$. Notice that $I(x; x^*) = \{2\}$ and that $U(x, x^*)$ is the segment $AC$, excluding the end points. Here $w^* \neq x^*$ is the fixed point closest to the vector $x$. We fix some $y^1, y^2 \in U(x; x^*)$ and we let $y^*$ be a point on the segment joining $x^*$ and $w^*$. The dashed–line rectangle is the set of all points $y$ satisfying $\|y - y^*\|_\infty \leq \|x - x^*\|_\infty$. We notice that if $y^*$ is chosen close enough to $x^*$, then $y^1$ and $y^2$ both lie strictly inside the dashed–line rectangle. Thus, $\|y^1 - y^*\|_\infty < \|x - x^*\|_\infty$ and $\|y^2 - y^*\|_\infty < \|x - x^*\|_\infty$, which shows that $d(y^1, y^2) < g(x)$.

**Lemma 2.1.** Suppose that $x \notin X^*$, $x^* \in X^*$, and that $g(x) = \inf_{w^* \in X^*} \|x - w^*\|_\infty < \|x - x^*\|_\infty$. Consider a vector $z \in Z$ of the form $z = (y^1, \ldots, y^B)$, with each $y^k$ belonging to $U(x; x^*)$. Then $d(z) < \|x - x^*\|_\infty$.

***Proof.*** Let $w^* \in X^*$ be such that $\|x - w^*\|_\infty = g(x)$. Let $\epsilon$ be a small positive scalar to be determined later and let $y^* = (1 - \epsilon)x^* + \epsilon w^*$. In particular, if $0 < \epsilon < 1$, then $y^* \in X^*$, because of the convexity of $X^*$. We will now show that $D(z; y^*) < g(x)$ when $\epsilon$ is chosen sufficiently small.

Fix some $k$ and consider the vector $y^k$. For any $i \in I(x; x^*)$, we have $y_i^k = x_i$, because $y^k \in U(x; x^*)$. Therefore,

$$|y_i^k - y_i^*| = |x_i - y_i^*| \leq (1-\epsilon)|x_i - x_i^*| + \epsilon|x_i - w_i^*| \leq (1-\epsilon)\|x - x^*\|_\infty + \epsilon g(x) < \|x - x^*\|_\infty.$$

Also, if $i \notin I(x;x^*)$, we have $|y_i^k - x_i^*| < \|x - x^*\|_\infty$, because $y^k \in U(x;x^*)$. Let $\alpha > 0$ be such that $\|x-x^*\|_\infty - |y_i^k - x_i^*| > \alpha$ for every $k$ and every $i \notin I(x;x^*)$. Then

$$|y_i^k - y_i^*| \leq |y_i^k - x_i^*| + |y_i^* - x_i^*| < \|x - x^*\|_\infty - \alpha + \epsilon\|w^* - x^*\|_\infty < \|x - x^*\|_\infty,$$

where the last inequality is valid provided that $\epsilon$ is chosen so that it satisfies $\epsilon\|w^* - x^*\|_\infty < \alpha$. With such a choice of $\epsilon$, we have $\|y^k - y^*\|_\infty < \|x - x^*\|_\infty$ for all $k$. This shows that $d(z) \leq D(z;y^*) < \|x - x^*\|_\infty$. **Q.E.D.**

As in Section 7.1, we let $z(t) = \big(x(t), x(t-1), \ldots, x(t-B+1)\big)$. We also use the notation, for $t \in T^i$,

$$x^i(t) = \Big(x_1\big(\tau_1^i(t)\big), \ldots, x_n\big(\tau_n^i(t)\big)\Big).$$

**Lemma 2.2.**

**(a)** If $x^* \in X^*$, then $\|x^i(t) - x^*\|_\infty \leq D\big(z(t);x^*\big)$ for every $t \in T^i$.

**(b)** If $x^* \in X^*$, then $D\big(z(t+1);x^*\big) \leq D\big(z(t);x^*\big)$ for all $t \geq 0$.

**(c)** There holds $d\big(z(t+1)\big) \leq d\big(z(t)\big)$ for all $t \geq 0$.

***Proof.***

**(a)** By definition, $|x_j(\tau) - x_j^*| \leq D\big(z(t);x^*\big)$ for all $\tau$ satisfying $t - B + 1 \leq \tau \leq t$, and the result follows because $t - B + 1 \leq \tau_j^i(t) \leq t$.

**(b)** If $t \in T^i$, we have $|x_i(t+1) - x_i^*| = |f\big(x^i(t)\big) - x_i^*| \leq \|x^i(t) - x^*\|_\infty \leq D\big(z(t);x^*\big)$, where the last step made use of part (a). Also, if $t \notin T^i$, then $|x_i(t+1) - x_i^*| = |x_i(t) - x_i^*| \leq D\big(z(t);x^*\big)$. We thus conclude that $\|x(t+1) - x^*\|_\infty \leq D\big(z(t);x^*\big)$, from which the desired conclusion follows.

**(c)** Let $x^*$ be such that $d\big(z(t)\big) = D\big(z(t);x^*\big)$. Then $d\big(z(t+1)\big) \leq D\big(z(t+1);x^*\big) \leq D\big(z(t);x^*\big) = d\big(z(t)\big)$. **Q.E.D.**

Notice that part (c) of Lemma 2.2 establishes that condition (b) in Prop. 1.1 is valid. In the remainder of the proof, we will show that if $z(0) \notin Z^*$, then $d\big(z(2nB + B)\big) < d\big(z(0)\big)$. This will establish that condition (a) of Prop. 1.1 is also valid, with $t^* = 2nB + B$.

Let us fix some $z(0) \notin Z^*$ and a particular scenario satisfying Assumption 1.1. We use the notation $d^* = d\big(z(0)\big)$ and from now on, we use $x^*$ to indicate a particular element of $X^*$ for which $d^* = d\big(z(0)\big) = D\big(z(0);x^*\big)$. We let

$$J(t) = \big\{i \mid |x_i(t) - x_i^*| = d^*\big\}.$$

**Lemma 2.3.**

**(a)** If $x_i(t+1) \neq x_i(t)$ for some $t \geq 0$, then $i \notin J(t+1)$.

**(b)** For every $t \geq 0$, we have $J(t+1) \subset J(t)$.

***Proof.***

**(a)** If $x_i(t+1) \neq x_i(t)$, we have $t \in T^i$. Furthermore, $f_i\big(x^i(t)\big) = x_i(t+1) \neq x_i(t) = x_i^i(t)$, where the last equality follows from Assumption 1.1(c). Using Assumption 2.2(c), we obtain

$$|x_i(t+1)-x_i^*| = |f_i\big(x^i(t)\big)-x_i^*| < \|x^i(t)-x^*\|_\infty \leq D\big(z(t);x^*\big) \leq D\big(z(0);x^*\big) = d^*.$$

(We have made use of parts (a) and (b) of Lemma 2.2.)

**(b)** If $i \in J(t+1)$, then part (a) shows that $x_i(t) = x_i(t+1)$, which implies that $i \in J(t)$. **Q.E.D.**

For any finite set $J$, we use $|J|$ to denote its cardinality.

**Lemma 2.4.** Suppose that $d^* = d\big(z(0)\big) > 0$. Then at least one of the following is true:

**(i)** Either $d\big(z(2nB+B)\big) < d^*$,

**(ii)** Or for every $t_0$ in the range $0 \leq t_0 \leq 2(n-1)B$, if $|J(t_0)| > 0$, then $|J(t_0+2B)| < |J(t_0)|$.

***Proof.*** Suppose that $d\big(z(2nB+B)\big) = d^*$. We shall prove that if $|J(0)| > 0$, then $|J(2B)| < |J(0)|$, which will establish alternative (ii), at least for $t_0 = 0$. The argument for the case of an arbitrary $t_0$ in the given range is identical. Assume, in order to obtain a contradiction, that $J(0)$ is nonempty and that $|J(0)| = |J(2B)|$. Then Lemma 2.3 shows that $J(t) = J(0)$ for every $t$ satisfying $0 \leq t \leq 2B$; furthermore, for any such $t$ and every $i \in J(0)$, we have $x_i(t) = x_i(0)$.

Consider some $t$ satisfying $0 \leq t \leq 2B$. Then $\|x(t) - x^*\|_\infty \leq D\big(z(t);x^*\big) \leq D\big(z(0);x^*\big) = d^*$; also, since $J(t)$ is nonempty, we have $\|x(t) - x^*\|_\infty \geq d^*$. Thus, $\|x(t) - x^*\|_\infty = d^*$ and $J(0) = J(t) = I\big(x(t);x^*\big)$. This implies that $x_i(t) = x_i(0)$ for every $i \in I\big(x(t);x^*\big) = I\big(x(0);x^*\big)$ and, from the definition of $U(x;x^*)$, it is seen that $x(t) \in U\big(x(0);x^*\big)$.

Consider now some $t \in T^i$ satisfying $B-1 < t \leq 2B-1$. Then $0 \leq \tau_j^i(t) \leq 2B-1$ and this implies that $x_j^i(t) = x_j\big(\tau_j^i(t)\big) = x_j(0)$ for every $j \in J(0)$. Also, $|x_j\big(\tau_j^i(t)\big) - x_j^*| < d^*$ for every $j \notin J\big(\tau_j^i(t)\big) = J(0)$. We conclude that $x^i(t) \in U\big(x(0);x^*\big)$.

We will now apply Assumption 2.2(b). We distinguish two cases:

*Case A.* Suppose that $g\big(x(0)\big) < \|x(0) - x^*\|_\infty = d^*$. Since $x(t) \in U\big(x(0);x^*\big)$ for $t = 0, 1, \ldots, B$, Lemma 2.1 implies that $d\big(z(B)\big) < d^*$. Since $d\big(z(t)\big)$ is

monotonically nonincreasing, this contradicts our assumption that $d\big(z(2nB + B)\big) = d^*$.

*Case B.* Suppose that $g\big(x(0)\big) = \|x(0)-x^*\|_\infty = d^*$. Assumption 2.2(b) states that there exists some $i \in I\big(x(0); x^*\big) = J(0)$ such that $f_i(y) \neq y_i$ for all $y \in U\big(x(0); x^*\big)$. Let $t_i$ be an element of $T^i$ such that $B - 1 < t_i \leq 2B - 1$. As shown earlier, $x^i(t_i) \in U\big(x(0); x^*\big)$ and it follows that $x_i(t_i + 1) \neq x_i(t)$. This implies that $i \notin J(t_i + 1)$. Therefore, $|J(2B)| \leq |J(t_i + 1)| < |J(0)|$, which contradicts the assumption made in the beginning of the proof of the lemma and concludes the proof. **Q.E.D.**

**Proof of Proposition 2.3. (cont.)** According to Lemma 2.4, if $d\big(z(0)\big) > 0$, then there are two cases to consider. Either $d\big(z(2nB + B)\big) < d\big(z(0)\big)$ or the cardinality of $J(t)$ decreases every $2B$ time units until it is empty. In the latter case, we conclude that $J(2nB)$ is empty. Using Lemma 2.3, it follows that $J(t)$ is empty and $\|x(t)-x^*\|_\infty < d^*$ for all $t$ satisfying $2nB \leq t \leq 2(n + 1)B$. This implies that $d\big(z(2nB + B)\big) < d^*$. We have, therefore, reached the conclusion that the inequality $d\big(z(2nB + B)\big) < d\big(z(0)\big)$ holds in both cases. This establishes condition (a) of Prop. 1.1. Notice that the sequence $\{z(t)\}$ is bounded, because of Lemma 2.2(b), and therefore has a limit point $z^*$. By Prop. 1.1, $z^* \in Z^*$. Let $x^*$ be such that $z^* = (x^*, \ldots, x^*)$. Then $x^* \in X^*$ and $\liminf_{t\to\infty} D\big(z(t); x^*\big) = 0$. Since $D\big(z(t); x^*\big)$ is nonincreasing, we conclude that it converges to zero. This establishes that $z(t)$ converges to $z^*$ and $x(t)$ converges to $x^*$. **Q.E.D.**

It should be pointed out that the limit of $x(t)$ in Prop. 2.3 depends in general on the particular scenario as well as on the initial conditions.

### 7.2.2 Weakly Diagonally Dominant Systems of Linear Equations

Consider a system $Ax = b$ of linear equations, where $A$ is an $n \times n$ matrix with entries $a_{ij}$, and $b$ is a vector in $\Re^n$. We study the case where $A$ and $b$ have the following properties.

**Assumption 2.3.**

**(a)** For every $i$, we have

$$|1 - a_{ii}| + \sum_{j\neq i} |a_{ij}| \leq 1. \tag{2.5}$$

**(b)** The set $X^*$ of solutions of the equation $Ax = b$ is nonempty.

**(c)** The matrix $A$ is irreducible.

Notice that Assumption 2.3(a) implies that $a_{ii} \in [0, 2]$. For the special case where $0 \leq a_{ii} \leq 1$, inequality (2.5) is equivalent to the condition $\sum_{j\neq i} |a_{ij}| \leq a_{ii}$,

which is a weak row diagonal dominance condition. Notice also that an alternative statement of condition (2.5) is that $\|I - A\|_\infty \leq 1$. It should be pointed out that no conclusions can be drawn from this condition on the existence and uniqueness of solutions of $Ax = b$. Existence is explicitly assumed in Assumption 2.3(b), but $X^*$ could have several elements.

We consider the partially asynchronous iteration

$$x := f(x) = x - \gamma(Ax - b), \tag{2.6}$$

where $\gamma$ is a relaxation parameter belonging to $(0, 1)$. Notice that the iteration matrix in Eq. (2.6) is equal to $I - \gamma A$. Under Assumption 2.3(a), we have

$$\|I - \gamma A\|_\infty \leq (1 - \gamma) + \gamma\|I - A\|_\infty \leq (1 - \gamma) + \gamma = 1.$$

If we had $\|I - A\|_\infty < 1$, then the last inequality would be strict and totally asynchronous convergence would be guaranteed, according to the results of Section 6.3. Thus, the following result is interesting primarily in the case where $\|I - A\|_\infty = 1$.

**Proposition 2.4.** If Assumption 2.3 holds and if $0 < \gamma < 1$, then the mapping $f$ of Eq. (2.6) satisfies Assumptions 2.1 and 2.2. In particular, under Assumption 1.1 (partial asynchronism), the asynchronous iteration (2.6) converges to some $x^*$ satisfying $Ax^* = b$.

***Proof.*** Consider the mapping $h$ defined by $h(x) = x - (Ax - b)$ and notice that $f(x) = (1 - \gamma)x + \gamma h(x)$. In view of Prop. 2.2, it is sufficient to show that $h$ satisfies Assumptions 2.1, 2.2(a), and 2.2(b).

The set of fixed points of $h$ is nonempty by Assumption 2.3(b). The function $h$ is clearly continuous. Let $x^*$ satisfy $Ax^* = b$. Then $\|h(x) - x^*\|_\infty = \|x - Ax + b - x^* + Ax^* - b\|_\infty = \|(I - A)(x - x^*)\|_\infty \leq \|I - A\|_\infty \cdot \|x - x^*\|_\infty \leq \|x - x^*\|_\infty$, where the last inequality follows from Eq. (2.5). We conclude that $h$ satisfies Assumption 2.1.

Suppose that $x^* \in X^*$ and $y^* \in X^*$. Then $Ax^* = Ay^* = b$, which implies that $cAx^* + (1 - c)Ay^* = b$ for every scalar $c$. Thus, $cx^* + (1 - c)y^* \in X^*$. This shows that $X^*$ is convex and Assumption 2.2(a) holds.

It remains to establish the validity of Assumption 2.2(b) for the function $h$. Fix some $x \in \Re^n$ and some $x^* \in X^*$ such that $\|x - x^*\|_\infty = g(x) > 0$, and let $S = I(x; x^*) = \{i \mid |x_i - x_i^*| = \|x - x^*\|_\infty\}$. If Assumption 2.2(b) fails to hold, then for every $i \in S$, there exists a vector $y^i \in U(x; x^*)$ such that $h_i(y^i) = y_i^i$; equivalently, $[Ay^i]_i - b_i = 0$. We now fix some $i \in S$ and a vector $y^i$ with these properties. Since $y^i \in U(x; x^*)$, we have $|y_j^i - x_j^*| = \|x - x^*\|_\infty$ for $j \in S$ and $|y_j^i - x_j^*| < \|x - x^*\|_\infty$ for $j \notin S$. Since $Ax^* = b$, we obtain $[A(y^i - x^*)]_i = 0$, which can be written as

$$y_i^i - x_i^* = (1 - a_{ii})(y_i^i - x_i^*) - \sum_{\{j \in S \mid j \neq i\}} a_{ij}(y_j^i - x_j^*) - \sum_{j \notin S} a_{ij}(y_j^i - x_j^*).$$

Taking absolute values, we obtain

$$\begin{aligned}\|x - x^*\|_\infty &= |y_i^i - x_i^*| \\ &\leq |1 - a_{ii}| \cdot |y_i^i - x_i^*| + \sum_{\{j \in S | j \neq i\}} |a_{ij}| \cdot |y_j^i - x_j^*| + \sum_{j \notin S} |a_{ij}| \cdot |y_j^i - x_j^*| \\ &\leq \Big(|1 - a_{ii}| + \sum_{j \neq i} |a_{ij}|\Big) \|x - x^*\|_\infty \leq \|x - x^*\|_\infty. \end{aligned} \tag{2.7}$$

Therefore, equality holds throughout. However, for the second inequality in Eq. (2.7) to be an equality, we need

$$\sum_{j \notin S} |a_{ij}| \cdot |y_j^i - x_j^*| = \sum_{j \notin S} |a_{ij}| \cdot \|x - x^*\|_\infty.$$

Since $|y_j^i - x_j^*| < \|x - x^*\|_\infty$ for $j \notin S$, we conclude that $\sum_{j \notin S} |a_{ij}| = 0$, which shows that $a_{ij} = 0$ for every $j \notin S$. Since this is true for every $i \in S$ and since $A$ is irreducible, we conclude that $S = \{1, \ldots, n\}$. We therefore have $U(x; x^*) = \{x\}$. Thus, $y^i = x$ for all $i$, and since $[Ay^i]_i = b_i$, we conclude that $Ax = b$ and $x \in X^*$. Thus, $g(x) = 0 < \|x - x^*\|_\infty$. This contradicts our initial assumptions for $g(x)$ and shows that Assumption 2.2(b) is satisfied. Convergence of the iteration (2.6) follows from Prop. 2.3. **Q.E.D.**

According to Exercise 1.2, the convergence of the partially asynchronous iteration (2.6) is guaranteed to be geometric. (This is because, convergence was proved by appealing to Prop. 2.3 which, in turn was proved by showing that the conditions of Prop. 1.1 are satisfied. But these are precisely the conditions assumed in Exercise 1.2.) On the other hand, convergence is not guaranteed, in general, if the relaxation parameter $\gamma$ is equal to 1 (see Example 2.1 to follow or Example 3.1 in the next section). Furthermore, convergence fails to hold, in general, if the iteration is executed totally asynchronously (see Example 1.2). The following example shows that a relaxation parameter $\gamma \in (0, 1)$ may be needed even if the matrix $A$ is invertible, in which case, the set $X^*$ of vectors $x^*$ satisfying $Ax^* = b$ consists of a single element.

**Example 2.1.**

Let

$$A = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

and $b = 0$. Then the iteration $x := x - Ax$ takes the form $x_1 := x_2$ and $x_2 := -x_1$. This iteration does not converge when executed in Gauss–Seidel fashion [see Fig. 7.2.4(a)]. Since Gauss–Seidel execution is a special case of partially asynchronous execution, it follows that the partially asynchronous iteration $x := x - Ax$ does not converge. On the other hand, if a relaxation parameter $\gamma \in (0, 1)$ is introduced, the iteration $x := x - \gamma Ax$ converges when executed in Gauss–Seidel fashion [see Fig. 7.2.4(b)]. Furthermore, it is seen that

Assumption 2.3 is satisfied and, according to Prop. 2.4, the partially asynchronous iteration $x := x - \gamma Ax$ converges to zero.

### 7.2.3 Strictly Convex Network Flow Problems

Consider the nonlinear network flow problem of Sections 5.5 and 6.6:

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

$$\text{subject to} \quad \sum_{\{j|(i,j)\in A\}} f_{ij} - \sum_{\{j|(j,i)\in A\}} f_{ji} = s_i, \qquad \forall\, i \in N, \tag{2.8}$$

$$b_{ij} \le f_{ij} \le c_{ij}, \qquad \forall\, (i,j) \in A, \tag{2.9}$$

under the assumption of Sections 5.5 and 6.6:

**Assumption 2.4.** (*Strict Convexity*) The functions $a_{ij}(\cdot)$, $(i,j) \in A$, are strictly convex real–valued functions, and problem (CNF) has at least one feasible solution.

We recall from Section 5.5 that for a given flow vector $f$, the surplus of a node $i$ is

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

The gradient of the dual functional is given by

$$\frac{\partial q(p)}{\partial p_i} = \sum_{\{j|(i,j)\in A\}} \nabla q_{ij}(p_i - p_j) - \sum_{\{j|(j,i)\in A\}} \nabla q_{ji}(p_j - p_i) + s_i = g_i(p), \tag{2.10}$$

where

$$\nabla q_{ij}(p_i - p_j) = -[\text{the unique flow } f_{ij} \text{ of arc } (i,j) \text{ satisfying complementary slackness together with } p],$$

$$g_i(p) = \text{surplus of node } i \text{ corresponding to the unique } f \text{ satisfying complementary slackness together with } p.$$

For $i = 1, \ldots, |N|$, we consider, as in Section 6.6, the point–to–set mapping $R_i$ that assigns to a price vector $p$ the interval of all prices $\xi$ that maximize the dual cost along the $i$th price starting from $p$, that is,

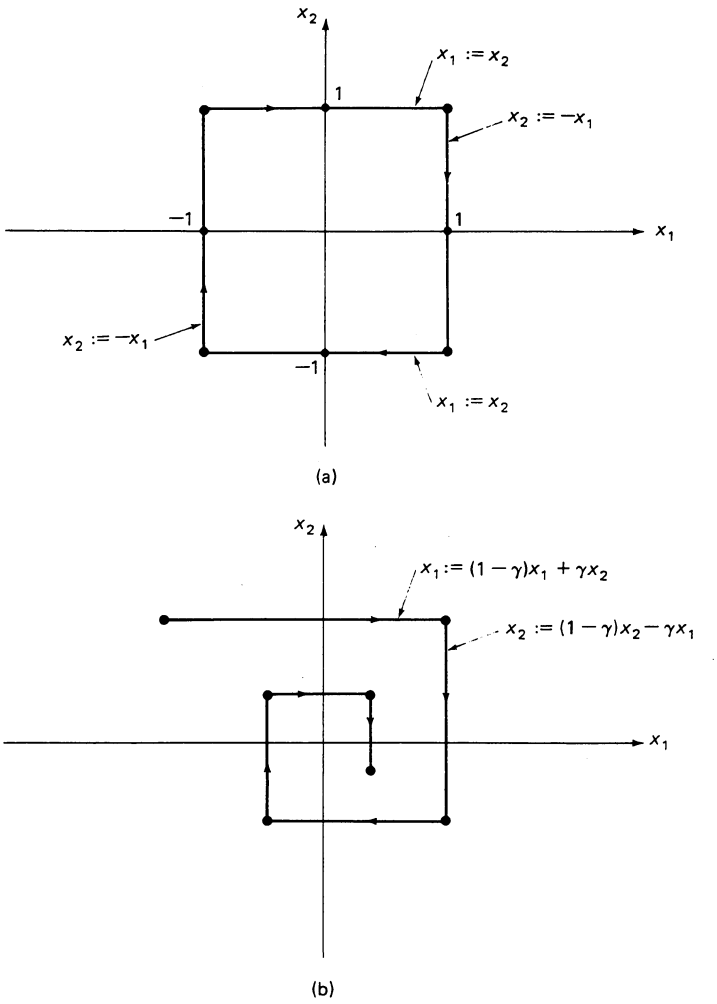



**Figure 7.2.4** The need for a relaxation parameter in the iteration $x_1 := x_2$ and $x_2 := -x_1$. In part (a), the trajectory corresponding to a nonconvergent Gauss–Seidel execution is shown. When a relaxation parameter $\gamma \in (0, 1)$ is introduced, the iteration becomes $x_1 := (1-\gamma)x_1 + \gamma x_2$ and $x_2 := (1-\gamma)x_2 - \gamma x_1$, and Gauss–Seidel convergence is obtained [part (b)].

$$R_i(p) = \{\xi \mid g_i(p_1, \ldots, p_{i-1}, \xi, p_{i+1}, \ldots, p_{|N|}) = 0\}$$
$$= \Big\{\xi \;\Big|\; \sum_{\{j|(j,i)\in A\}} \nabla q_{ji}(p_j - \xi) = \sum_{\{j|(i,j)\in A\}} \nabla q_{ij}(\xi - p_j) + s_i\Big\}.$$

It was shown in Section 5.5 that $R_i(p)$ is nonempty for all $i$ and $p$. We also recall from Lemma 6.1 in Section 6.6 that the set $R_i(p)$ is bounded above for every $p$ if and only if we have

$$s_i < \sum_{\{j|(i,j)\in A\}} c_{ij} - \sum_{\{j|(j,i)\in A\}} b_{ji}. \tag{2.11}$$

Similarly, the set $R_i(p)$ is bounded below for every $p$ if and only if we have

$$\sum_{\{j|(i,j)\in A\}} b_{ij} - \sum_{\{j|(j,i)\in A\}} c_{ji} < s_i. \tag{2.12}$$

We consider the mappings

$$\overline{R}_i(p) = \sup_{\xi\in R_i(p)} \xi, \tag{2.13}$$

$$\underline{R}_i(p) = \inf_{\xi\in R_i(p)} \xi, \tag{2.14}$$

and the mapping $\tilde{R} : R^{|N|} \to R^{|N|}$ given by

$$\tilde{R}_i(p) = \begin{cases} p_i, & \text{if } g_i(p) = 0, \\ \overline{R}_i(p), & \text{if } g_i(p) < 0, \\ \underline{R}_i(p), & \text{if } g_i(p) > 0. \end{cases} \tag{2.15}$$

Note that $g_i(p) \geq \sum_{\{j|(j,i)\in A\}} b_{ji} - \sum_{\{j|(i,j)\in A\}} c_{ij} + s_i$, so if $g_i(p) < 0$, then inequality (2.11) holds and we have that $\overline{R}_i(p)$ is finite for all $p$. Similarly, if $g_i(p) > 0$, then $\underline{R}_i(p)$ is finite for all $p$. As a result, the mapping $\tilde{R}(p)$ is well defined by Eq. (2.15). Another fact that will be useful to us is that $\overline{R}_i(p)$ is continuous whenever it is finite, that is, when inequality (2.11) holds. This was proved in Prop. 6.1 in Section 6.6. Similarly, $\underline{R}_i(p)$ is continuous whenever it is finite, that is, when inequality (2.12) holds.

We now focus on the relaxation iteration

$$p := R^\gamma(p),$$

where $R^\gamma(p)$ is defined for all $\gamma \in (0, 1]$ by

$$R^\gamma(p) = p + \gamma\big(\tilde{R}(p) - p\big). \tag{2.16}$$

We note that the set of all dual optimal price vectors $\hat{P}$ coincides with the set of all fixed points of $R^\gamma$, that is, we have

$$\hat{P} = \big\{p \mid p = R^\gamma(p)\big\}, \qquad \forall\, \gamma \in (0, 1].$$

We will show that the function $\tilde{R}$ satisfies Assumptions 2.1, 2.2(a), and 2.2(b) (with $f$, $x$, and $X^*$ in these assumptions replaced by $\tilde{R}$, $p$, and $\hat{P}$, respectively). In view of Props. 2.2 and 2.3, this implies partially asynchronous convergence of the iteration $p := R^\gamma(p)$ for all $\gamma \in (0, 1)$.

**Proposition 2.5.** If Assumption 2.4 holds, the function $\tilde{R}$ satisfies Assumptions 2.1, 2.2(a), and 2.2(b). In particular, under Assumption 1.1 (partial asynchronism), the asynchronous iteration $p := R^{\gamma}(p)$, where $\gamma \in (0, 1)$, converges to some element of $\hat{P}$.

***Proof.*** The set $\hat{P}$ is convex because it is the optimal solution set of the unconstrained dual maximization problem which has a concave cost function, and it was shown to be nonempty in Section 5.5, so Assumptions 2.1(a) and 2.2(a) are satisfied. Lemma 5.3 shows that for all $p \in R^{|N|}$ and $p^* \in P^*$, we have

$$\min_{i \in N}\{p_i - p_i^*\} \leq \min_{i \in N}\{\tilde{R}_i(p) - p_i^*\} \leq \max_{i \in N}\{\tilde{R}_i(p) - p_i^*\} \leq \max_{i \in N}\{p_i - p_i^*\}.$$

It follows that

$$\|\tilde{R}(p) - p^*\|_\infty \leq \|p - p^*\|_\infty, \qquad \forall\, p \in R^{|N|},\ p^* \in \hat{P},$$

so $\tilde{R}$ is nonexpansive and Assumption 2.1(c) is satisfied.

We now show that Assumption 2.1(b) is satisfied, that is, $\tilde{R}$ is continuous. This is a straightforward consequence of the continuity of the mappings $\underline{R}_i$ and $\overline{R}_i$, which was shown in Section 6.6. More specifically, let $\{p(t)\}$ be a sequence converging to some $p$. If $g_i(p) > 0$, then $\tilde{R}_i\big(p(t)\big) = \underline{R}_i\big(p(t)\big)$ for $t$ sufficiently large, so by continuity of $\underline{R}_i$ (Prop. 6.1 in Section 6.6), we have $\tilde{R}_i\big(p(t)\big) \to \tilde{R}_i(p)$. Similarly, we show that if $g_i(p) < 0$, then $\tilde{R}_i\big(p(t)\big) \to \tilde{R}_i(p)$. Finally, if $g_i(p) = 0$, there are four possibilities: (a) $p_i = \underline{R}_i(p) < \overline{R}_i(p)$, (b) $p_i = \overline{R}_i(p) > \underline{R}_i(p)$, (c) $p_i = \underline{R}_i(p) = \overline{R}_i(p)$, and (d) $\underline{R}_i(p) < p_i < \overline{R}_i(p)$. In case (a) [or (b)], we have for sufficiently large $t$, either $\tilde{R}\big(p(t)\big) = p(t)$ or $\tilde{R}\big(p(t)\big) = \underline{R}_i\big(p(t)\big)$ [$\tilde{R}\big(p(t)\big) = \overline{R}_i\big(p(t)\big)$, respectively], and the continuity of $\underline{R}_i$ and $\overline{R}_i$ implies that $\tilde{R}_i\big(p(t)\big) \to \tilde{R}_i(p)$. In case (c), the continuity of $\underline{R}_i$ and $\overline{R}_i$ implies that

$$\lim_{t \to \infty} \underline{R}_i\big(p(t)\big) = \lim_{t \to \infty} \overline{R}_i\big(p(t)\big) = p_i = \tilde{R}_i(p),$$

and since we have for all $t$, $\underline{R}_i\big(p(t)\big) \leq \tilde{R}_i\big(p(t)\big) \leq \overline{R}_i\big(p(t)\big)$, we obtain $\tilde{R}_i\big(p(t)\big) \to \tilde{R}_i(p)$. Finally, in case (d), we have $\tilde{R}\big(p(t)\big) = p(t)$ for sufficiently large $t$, and since $p(t) \to p$, we have $\tilde{R}_i\big(p(t)\big) \to \tilde{R}_i(p)$. Thus, $\tilde{R}$ is continuous.

There remains to show that Assumption 2.2(b) holds. Fix some $p \in R^{|N|}$ and some $p^* \in \hat{P}$ such that $\|p - p^*\|_\infty = \min_{\hat{p} \in \hat{P}} \|p - \hat{p}\|_\infty > 0$, and let

$$I^+ = \big\{i \mid p_i - p_i^* = \|p - p^*\|_\infty\big\},$$

$$I^- = \big\{i \mid p_i - p_i^* = -\|p - p^*\|_\infty\big\},$$

$$U(p; p^*) = \Big\{\tilde{p} \;\Big|\; \tilde{p}_i = p_i,\ \forall\, i \in I^+ \cup I^-,$$

$$\text{and } |\tilde{p}_i - p_i^*| < \|p - p^*\|_\infty,\ \forall\, i \notin I^+ \cup I^-\Big\}.$$

We want to show that there exists some $i \in I^+ \cup I^-$ such that $\tilde{R}_i(\tilde{p}) \neq \tilde{p}_i$ for all $\tilde{p} \in U(p; p^*)$. We argue by contradiction. Assume that for every $i \in I^+ \cup I^-$, there exists a price vector $p^i \in U(p; p^*)$ such that $\tilde{R}_i(p^i) = p_i^i$ or, equivalently, the surplus of the $i$th node corresponding to $p^i$ is zero:

$$g_i(p^i) = 0.$$

Define the positive scalar $\delta$ by

$$\delta = \begin{cases} \|p - p^*\|_\infty, & \text{if } I^+ \cup I^- = \{1, \ldots, |N|\}, \\ \|p - p^*\|_\infty - \beta & \text{otherwise,} \end{cases}$$

where

$$\beta = \max_{i \notin I^+ \cup I^-,\ k \in I^+ \cup I^-} |p_i^k - p_i^*|.$$

We assume in what follows that both $I^+$ and $I^-$ are nonempty; if either $I^+$ or $I^-$ is empty, a similar (and in fact simpler) proof can be used.

We have for each $i \in I^+$,

$$p_i^i - p_j^i \geq p_i^* - p_j^* + \delta, \qquad \forall\, j \notin I^+, \tag{2.17}$$

$$p_i^i - p_j^i = p_i^* - p_j^*, \qquad \forall\, j \in I^+. \tag{2.18}$$

Let $f_{mn}^i$ (or $f_{mn}^*$) be the unique flow of each arc $(m, n)$ that satisfies complementary slackness together with $p^i$ (or $p^*$, respectively). Using inequality (2.17) and Eq. (2.18), we obtain for all $i \in I^+$,

$$f_{ij}^i = -\nabla q_{ij}(p_i^i - p_j^i) \geq -\nabla q_{ij}(p_i^* - p_j^* + \delta) \geq -\nabla q_{ij}(p_i^* - p_j^*) = f_{ij}^*,$$
$$\text{if } j \notin I^+ \text{ and } (i, j) \in A,$$
$$f_{ji}^i = -\nabla q_{ji}(p_j^i - p_i^i) \leq -\nabla q_{ji}(p_j^* - p_i^* - \delta) \leq -\nabla q_{ji}(p_j^* - p_i^*) = f_{ji}^*,$$
$$\text{if } j \notin I^+ \text{ and } (j, i) \in A,$$
$$f_{ij}^i = -\nabla q_{ij}(p_i^* - p_j^*) = f_{ij}^*, \qquad \text{if } j \in I^+ \text{ and } (i, j) \in A,$$
$$f_{ji}^i = -\nabla q_{ji}(p_j^* - p_i^*) = f_{ji}^*, \qquad \text{if } j \in I^+ \text{ and } (j, i) \in A.$$

By adding the preceding inequalities, we obtain

$$
\begin{aligned}
g_i(p^i) &= \sum_{\{j|(j,i)\in A\}} f_{ji}^i - \sum_{\{j|(i,j)\in A\}} f_{ij}^i + s_i \\
&\le -\sum_{\{j|(j,i)\in A, j\notin I^+\}} \nabla q_{ji}(p_j^* - p_i^* - \delta) - \sum_{\{j|(j,i)\in A, j\in I^+\}} \nabla q_{ji}(p_j^* - p_i^*) \\
&\quad + \sum_{\{j|(i,j)\in A, j\notin I^+\}} \nabla q_{ij}(p_i^* - p_j^* + \delta) + \sum_{\{j|(i,j)\in A, j\in I^+\}} \nabla q_{ij}(p_i^* - p_j^*) + s_i \\
&\le \sum_{\{j|(j,i)\in A\}} f_{ji}^* - \sum_{\{j|(i,j)\in A\}} f_{ij}^* + s_i \\
&= g_i(p^*).
\end{aligned}
$$

Since $g_i(p^i) = g_i(p^*) = 0$, we obtain

$$
\begin{aligned}
&-\sum_{\{j|(j,i)\in A, j\notin I^+\}} \nabla q_{ji}(p_j^* - p_i^* - \delta) - \sum_{\{j|(j,i)\in A, j\in I^+\}} \nabla q_{ji}(p_j^* - p_i^*) \\
&+ \sum_{\{j|(i,j)\in A, j\notin I^+\}} \nabla q_{ij}(p_i^* - p_j^* + \delta) + \sum_{\{j|(i,j)\in A, j\in I^+\}} \nabla q_{ij}(p_i^* - p_j^*) + s_i = 0
\end{aligned}
$$

or equivalently, $g_i(\tilde{p}) = 0$ for all $i = 1, \ldots, |N|$, where the vector $\tilde{p}$ is given by

$$
\tilde{p}_i = \begin{cases} p_i^* + \delta, & \text{if } i \in I^+, \\ p_i^*, & \text{if } i \notin I^+. \end{cases}
$$

Therefore $\tilde{p} \in \hat{P}$. By construction we have

$$
\|p - \tilde{p}\|_\infty = \|p - p^*\|_\infty = \min_{\hat{p}\in\hat{P}} \|p - \hat{p}\|_\infty \tag{2.19}
$$

$$
p_i - \tilde{p}_i = -\|p - \tilde{p}\|_\infty, \qquad \forall\, i \in I^-, \tag{2.20}
$$

$$
|p_i - \tilde{p}_i| < \|p - \tilde{p}\|_\infty, \qquad \forall\, i \notin I^-. \tag{2.21}
$$

It follows that we can choose $\epsilon > 0$ which is sufficiently small so that the vector $\hat{p}$ with coordinates $\hat{p}_i = \tilde{p}_i - \epsilon$ for all $i$ satisfies

$$
\|p - \hat{p}\|_\infty = \|p - \tilde{p}\|_\infty - \epsilon. \tag{2.22}
$$

Since the coordinates of $\hat{p}$ differ from those of $\tilde{p}$ by the same constant, we have $\hat{p} \in \hat{P}$. Therefore Eq. (2.22) contradicts Eq. (2.19). **Q.E.D.**

Figure 7.2.5 illustrates the convergence process of the synchronous Jacobi version of the algorithm in a simple two node example and shows why taking $\gamma < 1$ is essential for convergence.

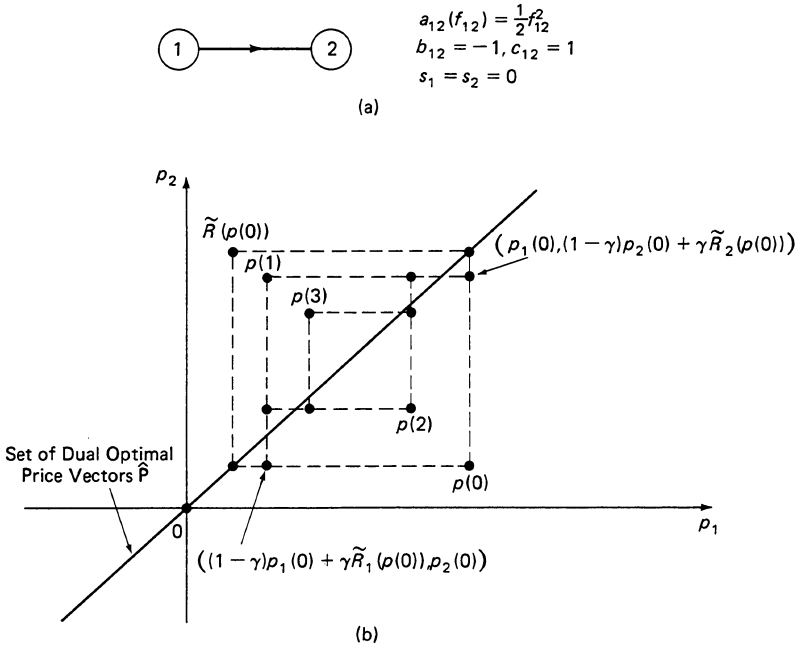



**Figure 7.2.5** (a) Two node example, with data as shown, where the Jacobi version of the relaxation method does not converge to an optimal price vector (cf. Fig. 5.5.5 in Section 5.5). (b) Sequence $\{p(0), p(1), \ldots\}$ generated by the synchronous Jacobi method $p(t+1) = (1-\gamma)p(t) + \gamma\tilde{R}\big(p(t)\big)$ when a relaxation parameter $\gamma \in (0, 1)$ is used.

## EXERCISES

**2.1.** Suppose that a function $h : \Re^n \mapsto \Re^n$ satisfies Assumptions 2.1, 2.2(a), and 2.2(b). Let Assumption 1.1 hold (partial asynchronism), and let $\gamma$ be a scalar satisfying $0 < \gamma < 1$. Propositions 2.2 and 2.3 show that the asynchronous iteration

$$x_i(t+1) = (1-\gamma)x_i(t) + \gamma h_i\big(x^i(t)\big), \qquad t \in T^i, \tag{2.23}$$

converges to a fixed point of $h$. In this exercise, we allow $\tau_i^i(t)$ to be different than $t$, as long as it satisfies $t - B < \tau_i^i(t) \leq t$, that is, we relax Assumption 1.1(c).

**(a)** Show that the sequence $\{x(t)\}$ generated by the asynchronous iteration (2.23) is still guaranteed to converge to a fixed point of $h$.

**(b)** Show that the sequence $\{x(t)\}$ generated by the asynchronous iteration

$$x_i(t+1) = x_i(t) + \gamma\Big(h_i\big(x^i(t)\big) - x_i\big(\tau_i^i(t)\big)\Big), \qquad t \in T^i,$$

can be unbounded and divergent.

(c) Show that the sequence $\{x(t)\}$ generated by the asynchronous iteration

$$x_i(t+1) = (1-\gamma)x_i\big(\tau_i^i(t)\big) + \gamma h_i\big(x^i(t)\big), \qquad t \in T^i,$$

is guaranteed to be bounded, but does not necessarily converge to a fixed point of $h$. *Hints:* For part (a), only a minor modification of the proof of Prop. 2.3 is needed. For part (b), use the function $h(x) = -x$ as an example. For part (c), use the example $h(x) = Ax$, where

$$A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix},$$

and apply Prop. 3.1 of Section 6.3.

**2.2.** Let $A$ be a nonnegative and irreducible matrix of dimensions $n \times n$, with the property $\sum_{j=1}^n a_{ij} \leq 1$. Suppose that all of the diagonal entries of $A$ are positive. Show that under Assumption 1.1, the asynchronous iteration $x := Ax$ converges to a vector $x^*$ satisfying $Ax^* = x^*$. *Hint:* Let $\delta$ be the smallest diagonal entry of $A$. The iteration $x := Ax$ can be written as [cf. Eq. (2.6)]

$$x := x - (1-\delta)\frac{I-A}{1-\delta}x.$$

Show that the matrix $(I-A)/(1-\delta)$ satisfies Assumption 2.3.

## 7.3 ALGORITHMS FOR AGREEMENT AND FOR MARKOV CHAIN PROBLEMS

We now consider a set of processors that try to reach agreement on a common scalar value by exchanging tentative values and combining them by forming convex combinations. Although this algorithm is of limited use on its own, it has interesting applications in certain contexts, such as the computation of invariant distributions of Markov chains (Subsection 7.3.2) and an extended model of asynchronous computation (Section 7.7).

### 7.3.1 The Agreement Algorithm

Consider a set $N = \{1, \ldots, n\}$ of processors, and suppose that the $i$th processor has a scalar $x_i(0)$ stored in its memory. We would like these processors to exchange messages and eventually agree on an intermediate value $y$, that is, the agreed upon value should satisfy

$$\min_{i \in N} x_i(0) \leq y \leq \max_{i \in N} x_i(0).$$

A trivial solution to this problem is to have a particular processor (say processor 1) communicate (directly or indirectly) its own value to all other processors and then all processors can agree on the value $x_1(0)$. We shall impose an additional requirement that excludes such a solution. In particular, we postulate the existence of a set $D \subset N$ of

*distinguished* processors and we are interested in guaranteeing that the values initially possessed by distinguished processors influence the agreed upon value $y$. For example, if $k \in D$, $x_k(0) > 0$, and $x_i(0) = 0$ for all $i \neq k$, then we would like $y$ to be influenced by $x_k(0)$ and be positive. Such a requirement can be met if the processors simply cooperate to compute the average of their initial values. This, however, may require a certain amount of coordination between the processors. We shall instead postulate an asynchronous iterative process whereby each processor receives tentative values from other processors and combines them with its own value by forming a convex combination. We let $x_i(t)$ be the value in the memory of the $i$th processor at time $t$, and we consider the asynchronous execution of the iteration

$$x_i := \sum_{j=1}^{n} a_{ij} x_j, \qquad i = 1, \ldots, n,$$

where the coefficients $a_{ij}$ are nonnegative scalars such that

$$\sum_{j=1}^{n} a_{ij} = 1, \qquad \forall i. \tag{3.1}$$

A precise description of the algorithm, referred to as the *agreement algorithm*, is

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

$$x_i(t+1) = \sum_{j=1}^{n} a_{ij} x_j\big(\tau_j^i(t)\big), \qquad \text{if } t \in T^i. \tag{3.3}$$

Here $T^i$ and $\tau_j^i(t)$ are as in Section 7.1 and will be assumed to satisfy the partial asynchronism Assumption 1.1. In the present context, it would be natural to assume that $0 \leq \tau_j^i(t)$. It is convenient, however, to consider a more general case, allowing $\tau_j^i(t)$ to be negative (as long as Assumption 1.1 is satisfied), and allowing $x_i(t)$, for $t < 0$, to be different from $x_i(0)$.

Let $A$ be the matrix whose $ij$th entry is equal to $a_{ij}$. We are then dealing with the special case of the model of Section 7.1, where the mapping $f$ is of the form

$$f(x) = Ax.$$

Notice that any vector $x \in \Re^n$ whose components are all equal is a fixed point of $f$, because of the condition $\sum_{j=1}^{n} a_{ij} = 1$ for all $i$. In the sequel, we derive conditions under which the sequence $\{x(t)\}$ of the vectors generated by the partially asynchronous agreement algorithm of Eqs. (3.2) and (3.3) converges to such a fixed point. A result of this type is readily obtained if the matrix $A$ is irreducible, a relaxation parameter $\gamma \in (0, 1)$ is employed, and the iteration $x := Ax$ is modified to $x := (1-\gamma)x + \gamma Ax = x - \gamma(I - A)x$. This is because the matrix $I - A$ satisfies Assumption 2.3 of Section

7.2 and Prop. 2.4 in that section applies. Even if no relaxation parameter is used, Prop. 2.4 can be again invoked as long as all of the diagonal entries of $A$ are positive (see Exercise 2.2 in the preceding section). The result derived in this section is more general in that it allows most of the diagonal entries of $A$ to be zero. Let us also mention that the agreement problem can be related to a network flow problem with quadratic costs (see Exercise 5.4 in Section 5.5.) We now consider some examples to motivate our assumptions.

**Example 3.1.**

Suppose that

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Here the set of fixed points of $f$ is $X^* = \{(x_1, x_2) \mid x_1 = x_2\}$, but the iteration $x := f(x)$ is not guaranteed to converge to $X^*$, even if it is executed synchronously. To see this, notice that if the synchronous execution is initialized with $x(0) = (1, 0)$, then $x(t)$ alternates between $(0, 1)$ and $(1, 0)$ (Fig. 7.3.1). The possibility of such nonconvergent oscillations will be eliminated by assuming that some diagonal entry of the matrix $A$ is nonzero. Such an entry has the effect of a relaxation parameter and serves as a damping factor.

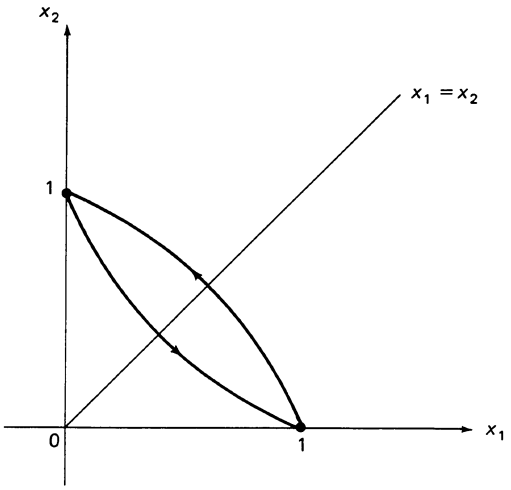



**Figure 7.3.1** Nonconvergence in Example 3.1. Here when $x(t)$ is updated synchronously, it alternates between $(1, 0)$ and $(0, 1)$. The oscillation can be eliminated by introducing a relaxation parameter $\gamma \in (0, 1)$ thereby replacing the iteration $x_1(t+1) = x_2(t)$ and $x_2(t+1) = x_1(t)$ with $x_1(t+1) = (1-\gamma)x_1(t) + \gamma x_2(t)$ and $x_2(t+1) = (1-\gamma)x_2(t) + \gamma x_1(t)$. This amounts to introducing positive diagonal elements in the iteration matrix $A$.

To justify the partial asynchronism assumption, consider the matrix

$$A = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

The agreement algorithm for this choice of $A$ is the same as the iteration considered in Example 1.2 of Section 7.1, where it was established that failure to converge is possible if part (b) of the partial asynchronism Assumption 1.1 is violated. There also exist

examples that demonstrate that parts (a) and (c) of Assumption 1.1 are also necessary for convergence (Exercise 3.1).

We define a directed graph $G = (N, A)$, where $N = \{1, \ldots, n\}$, and $A = \{(i, j) \mid i \neq j \text{ and } a_{ji} \neq 0\}$. Notice that $(i, j) \in A$ if and only if the value possessed by processor $i$ directly influences the value of processor $j$. Convergence will be proved under the following assumption on the matrix $A$.

**Assumption 3.1.** There exists a nonempty set $D \subset N$ of "distinguished" processors such that:

**(a)** For every $i \in D$, we have $a_{ii} > 0$.

**(b)** For every $i \in D$ and every $j \in N$, there exists a positive path from $i$ to $j$ in the previously defined graph $G$.

This assumption is quite natural. Since we wish the initial values of any distinguished processor to affect the value that is eventually agreed upon, we have imposed the condition that every distinguished processor can indirectly affect the value of every other processor [part (b)]. Part (a) of the assumption ensures that a distinguished processor does not forget its initial value when it executes its first iteration; it also serves to eliminate nonconvergent oscillations (cf. Example 3.1).

**Proposition 3.1.** Consider the agreement algorithm of Eqs. (3.2) and (3.3) and let Assumptions 1.1 (partial asynchronism) and 3.1 hold. Let $\alpha > 0$ be the smallest of the nonzero entries of $A$. Then there exist constants $\eta > 0$, $C > 0$, $\rho \in (0, 1)$, depending only on the number $n$ of processors, on $\alpha$, and on the asynchronism measure $B$ of Assumption 1.1, such that for any initial values $x_i(t)$, $t \leq 0$, and for any scenario allowed by Assumption 1.1, the following are true:

**(a)** The sequence $\{x_i(t)\}$ converges and its limit is the same for each processor $i$.

**(b)** There holds

$$\max_i x_i(t) - \min_i x_i(t) \leq C\rho^t \Big( \max_i \max_{-B+1 \leq \tau \leq 0} x_i(\tau) - \min_i \min_{-B+1 \leq \tau \leq 0} x_i(\tau) \Big).$$

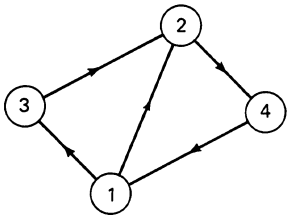
**(c)** If $0 \leq x_i(\tau)$ for every $i$ and every $\tau \leq 0$, and if $k \in D$, then $y \geq \eta x_k(0)$, where $y$ is the common limit whose existence is asserted in part (a).

***Proof.*** We define

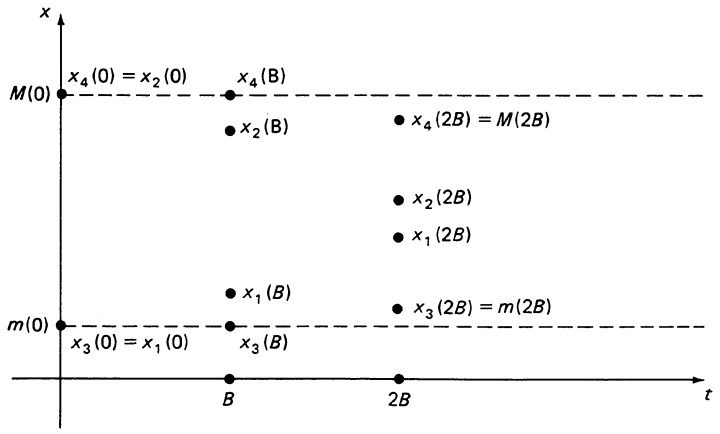
$$M(t) = \max_i \max_{t-B+1 \leq \tau \leq t} x_i(\tau), \tag{3.4}$$

$$m(t) = \min_i \min_{t-B+1 \leq \tau \leq t} x_i(\tau). \tag{3.5}$$

Notice that because of Assumption 1.1, we have $m(t) \leq x_j\left(\tau_j^i(t)\right) \leq M(t)$ for every $i$, $j$, $t \in T^i$, a fact that will be often used. The proof consists of showing that the difference $M(t) - m(t)$ is reduced to zero in the course of the algorithm (see Fig. 7.3.2 for an illustration of the main idea of the proof).




(a)




(b)

**Figure 7.3.2** (a) A possible graph $G$ associated to a matrix $A$ in the agreement algorithm. (b) Illustration of the convergence of the agreement algorithm for the graph of part (a). For simplicity, we assume that $a_{11} > 0$, $a_{44} > 0$, and that information is never outdated. Let the initial conditions be as indicated and let $I$ be the interval $[m(0), M(0)]$. Since $a_{11} > 0$ and $a_{44} > 0$, we have $x_1(t) < M(0)$ and $x_4(t) > m(0)$ for all times $t$. Within the first $B$ time units, processor 2 performs an iteration and $x_2$ is pulled by $x_1$ to a value smaller than $M(0)$. Similarly, $x_1$ is pulled by $x_4$ to a value larger than $m(0)$. After an additional $B$ time units, the variables $x_3$ and $x_4$ are pulled by $x_1$ and $x_2$, respectively, into the interior of $I$. At that point, all the components of $x(2B)$ lie in the interior of $I$ and the maximum disagreement $M(2B) - m(2B)$ is smaller than the initial maximum disagreement $M(0) - m(0)$.

**Lemma 3.1.**

**(a)** For every $t \geq 0$, we have $m(t+1) \geq m(t)$ and $M(t+1) \leq M(t)$.

**(b)** For every $t$ and $t' \geq t - B + 1$, we have $m(t) \leq x_i(t') \leq M(t)$.

***Proof.*** Fix some $i$ and $t$. If $t \notin T^i$, then $x_i(t+1) = x_i(t) \geq m(t)$. If $t \in T^i$, then

$$x_i(t+1) = \sum_{j=1}^{n} a_{ij} x_j\big(\tau_j^i(t)\big) \geq \sum_{j=1}^{n} a_{ij} m(t) = m(t).$$

Thus, in either case, $x_i(t+1) \geq m(t)$ for all $i$, and, using the definition of $m(t+1)$, it is seen that $m(t+1) \geq m(t)$. The proof of the inequality $M(t+1) \leq M(t)$ is similar. Finally, for $t' \geq t - B + 1$, we use the definition of $m(t'+B-1)$ and part (a) of the lemma to obtain $x_i(t') \geq m(t'+B-1) \geq m(t)$. The inequality $x_i(t') \leq M(t)$ is proved similarly. **Q.E.D.**

For the remainder of the proof of the proposition, we fix some $k \in D$. We let $D_0 = \{k\}$ and we let $D_\ell$ be the set of all $i \in N$ such that $\ell$ is the minimum number of arcs in a positive path, in the graph $G$, from node $k$ to node $i$. By Assumption 3.1(b), there exists a path from $k$ to every other processor. It follows that every $i \in N$ belongs to one of the sets $D_0, D_1, \ldots, D_{n-1}$. Furthermore, for every $i \in D_\ell$, there exists some $j \in D_{\ell-1}$, such that $(j,i) \in A$. Let $L \leq n-1$ be such that $D_0 \cup \cdots \cup D_L = N$.

**Lemma 3.2.** For every $\ell \in \{0, 1, \ldots, L\}$, there exists some $\eta_\ell > 0$ (depending only on $n$, $\alpha$, $B$) such that for every positive integer $s$, for every $t \in [s + 2\ell B + 1, s + 2LB + B]$, and for every $i \in D_\ell$, we have

$$x_i(t) \geq m(s) + \eta_\ell\big(x_k(s) - m(s)\big) \tag{3.6}$$

and

$$x_i(t) \leq M(s) - \eta_\ell\big(M(s) - x_k(s)\big). \tag{3.7}$$

***Proof.*** Throughout the proof of this lemma, $k \in D$ is fixed. Without loss of generality, we only consider the case where $s = 0$. Suppose that $t \in T^k$. Then

$$x_k(t+1) - m(0) = \sum_{j=1}^{n} a_{kj}\Big(x_j\big(\tau_j^k(t)\big) - m(0)\Big) \geq a_{kk}\big(x_k(t) - m(0)\big) \geq \alpha\big(x_k(t) - m(0)\big),$$

where we made use of the property $\tau_k^k(t) = t$ [cf. Assumption 1.1(c)]. If $t \notin T^k$, then $x_k(t+1) - m(0) = x_k(t) - m(0) \geq \alpha\big(x_k(t) - m(0)\big)$. It follows that for $t \in [0, 2LB+B]$, we have

$$x_k(t) - m(0) \geq \alpha^t\big(x_k(0) - m(0)\big) \geq \eta_0\big(x_k(0) - m(0)\big),$$

where $\eta_0 = \alpha^{2LB+B}$. This proves inequality (3.6) for $i = k$. Since $D_0 = \{k\}$, inequality (3.6) has been proved for all $i \in D_0$.

We now proceed by induction on $\ell$. Suppose that inequality (3.6) is true for some $\ell < L$. Let $i$ be an element of $D_{\ell+1}$. We shall prove inequality (3.6) for $i$.

Let $j \in D_\ell$ be such that $(j,i) \in A$. Suppose now that $t$ belongs to $T^i$ and satisfies $(2\ell+1)B \le t \le 2LB+B$. We then have $2\ell B+1 \le \tau_j^i(t) \le t \le 2LB+B$ and, by the induction hypothesis,

$$x_j\big(\tau_j^i(t)\big) - m(0) \ge \eta_\ell\big(x_k(0) - m(0)\big).$$

Consequently,

$$\begin{aligned} x_i(t+1) - m(0) &= \sum_{q=1}^{n} a_{iq}\Big(x_q\big(\tau_q^i(t)\big) - m(0)\Big) \ge a_{ij}\Big(x_j\big(\tau_j^i(t)\big) - m(0)\Big) \\ &\ge \alpha\eta_\ell\big(x_k(0) - m(0)\big) = \eta_{\ell+1}\big(x_k(0) - m(0)\big), \end{aligned} \tag{3.8}$$

where $\eta_{\ell+1} = \alpha\eta_\ell$. Let $t_i$ be an element of $T^i$ such that $(2\ell+1)B \le t_i \le 2(\ell+1)B$. Such a $t_i$ exists because of Assumption 1.1(a). Inequality (3.8) has been proved for $t = t_i$, as well for any subsequent $t \in T^i$ such that $t \le 2LB+B$. Furthermore, since $x_i(t) = x_i(t+1)$, if $t \notin T^i$, we conclude that inequality (3.8) holds for all $t$ such that $t_i \le t \le (2L+1)B$. Since $t_i \le 2(\ell+1)B$, we conclude that (3.8) holds for every $t$ such that $2(\ell+1)B \le t \le 2LB+B$. This establishes inequality (3.6) for $i \in D_{\ell+1}$ and for $s = 0$. This completes the induction and the proof of (3.6) for the case $s = 0$. The proof for the case of a general $s$ is identical. Finally, inequality (3.7) is proved by a symmetrical argument. **Q.E.D.**

**Proof of Proposition 3.1. (cont.)** We now complete the proof of the proposition. We have $m(t) \le M(t) \le M(0)$. Furthermore, the sequence $\{m(t)\}$ is nondecreasing. Since it is bounded above, it converges to a limit denoted by $\bar{m}$. Let $\bar{M}$ be the limit of $M(t)$, which exists by a similar argument. Let $\eta = \min\{\eta_0, \ldots, \eta_L\}$. Using Lemma 3.2 we obtain, for every $t \ge 0$,

$$\begin{aligned} m(t+2LB+B) &= \min_\ell \min_{i\in D_\ell} \min_{t+2LB+1\le\tau\le t+2LB+B} x_i(\tau) \ge m(t) + \min_\ell \eta_\ell\big(x_k(t) - m(t)\big) \\ &= m(t) + \eta\big(x_k(t) - m(t)\big). \end{aligned}$$

Similarly,

$$M(t+2LB+B) \le M(t) + \eta\big(x_k(t) - M(t)\big).$$

Subtracting these two inequalities, we obtain

$$M(t+2LB+B) - m(t+2LB+B) \le (1-\eta)\big(M(t) - m(t)\big). \tag{3.9}$$

Thus, $M(t) - m(t)$ decreases at the rate of a geometric progression and $\bar{M} = \bar{m}$. Let $y$ be the common value of $\bar{M}$ and $\bar{m}$. Since $m(t) \le x_i(t) \le M(t)$ for every $i$ and

$t$, it follows that the sequence $\{x_i(t)\}$ also converges to $y$ at the rate of a geometric progression. This establishes parts (a) and (b) of the proposition.

We now prove part (c). We have $m(0) \geq 0$, and using Lemma 3.2, we obtain $y \geq m(2LB + B) \geq \eta x_k(0)$. **Q.E.D.**

It should be emphasized that the value $y$ on which agreement is reached usually depends on the particular scenario.

### 7.3.2 An Asynchronous Algorithm for the Invariant Distribution of a Markov Chain

Notice that the iteration matrix $A$ in the agreement algorithm was a stochastic matrix. This suggests some similarities between the agreement algorithm and the iterative algorithms of Section 2.8 for computing the invariant distribution of a Markov chain. In this subsection, we let $P$ be an irreducible stochastic matrix and we establish partially asynchronous convergence of the iteration $\pi := \pi P$ by suitably exploiting the convergence result for the agreement algorithm. It should be recalled that the totally asynchronous version of this iteration converges if one of the components of $\pi$ is not iterated (Subsection 6.3.1). If all of the components of $\pi$ are iterated, then totally asynchronous convergence is not guaranteed (this can be seen from either Example 3.1 of this section or Example 1.2 of Section 7.1) and, therefore, the partial asynchronism assumption is essential.

Let $P$ be an irreducible and aperiodic stochastic matrix of dimensions $n \times n$ and let $p_{ij}$ denote its $ij$th entry. Let $\pi^*$ be the row vector of invariant probabilities of the corresponding Markov chain. Proposition 8.3 of Section 2.8 states that each component $\pi_i^*$ of $\pi^*$ is positive and $\lim_{t\to\infty} \pi(0)P^t = \pi^*$ for any row vector $\pi(0)$ whose entries add to 1. This leads to the iterative algorithm $\pi := \pi P$ of Section 2.8 and the corresponding synchronous parallel implementation. We now consider its asynchronous version.

We employ again the model of Section 7.1, except that the vector being iterated is denoted by $\pi$ and is a row vector. The iteration function $f$ is defined by $f(\pi) = \pi P$. The iteration is described by the equations

$$\pi_i(t+1) = \pi_i(t), \qquad t \notin T^i, \tag{3.10}$$

$$\pi_i(t+1) = \sum_{j=1}^{n} \pi_j\big(\tau_j^i(t)\big)p_{ji}, \qquad t \in T^i. \tag{3.11}$$

**Proposition 3.2.** Suppose that the matrix $P$ is stochastic, irreducible, and that there exists some $i^*$ such that $p_{i^*i^*} > 0$. Furthermore, suppose that the iteration (3.10)–(3.11) is initialized with positive values [$\pi_i(\tau) > 0$ for $\tau \leq 0$]. Then for every scenario allowed by Assumption 1.1, there exists a positive number $c$ such that $\lim_{t\to\infty} \pi(t) = c\pi^*$. Also, convergence takes place at the rate of a geometric progression.

***Proof.*** We prove this result by showing that it is a special case of the convergence result for the agreement algorithm (Prop. 3.1). We introduce a new set of variables $x_i(t)$ defined by

$$x_i(t) = \frac{\pi_i(t)}{\pi_i^*}.$$

These new variables are well defined because $\pi_i^* > 0$ for all $i$ as a consequence of irreducibility. In terms of the new variables, Eqs. (3.10) and (3.11) become

$$x_i(t+1) = x_i(t), \qquad t \notin T^i, \tag{3.12}$$

$$x_i(t+1) = \sum_{j=1}^{n} \frac{p_{ji}\pi_j^*}{\pi_i^*} x_j\big(\tau_j^i(t)\big), \quad t \in T^i. \tag{3.13}$$

By letting

$$a_{ij} = \frac{p_{ji}\pi_j^*}{\pi_i^*}, \tag{3.14}$$

Eq. (3.13) becomes

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

which is identical to Eq. (3.3) in the agreement algorithm. Furthermore, notice that $a_{ij} \geq 0$ and that

$$\sum_{j=1}^{n} a_{ij} = \sum_{j=1}^{n} \frac{p_{ji}\pi_j^*}{\pi_i^*} = \frac{1}{\pi_i^*} \sum_{j=1}^{n} p_{ji}\pi_j^* = \frac{\pi_i^*}{\pi_i^*} = 1,$$

where we have used the property $\pi^* = \pi^* P$. Thus, Eq. (3.1) holds as well.

We now verify that the remaining assumptions in Prop. 3.1 are satisfied. Let $i^*$ be such that $p_{i^*i^*} > 0$, and let $D = \{i^*\}$. Then $a_{i^*i^*} > 0$ and Assumption 3.1(a) holds. Also, since $P$ is irreducible, the coefficients $a_{ij}$ satisfy Assumption 3.1(b).

Therefore, Prop. 3.1 applies and shows that there exists a constant $c$ such that

$$\lim_{t\to\infty} x_i(t) = c, \qquad \forall i.$$

Equivalently,

$$\lim_{t\to\infty} \pi_i(t) = c\pi_i^*, \qquad \forall i.$$

Geometric convergence follows from part (b) of Prop. 3.1.

Let $m(0) = \min_i \min_{-B+1\leq\tau\leq 0} x_i(\tau)$. Since the algorithm is initialized with positive values, $m(0)$ is positive. As shown in the convergence proof for the agreement algorithm, the limit $c$ of $x_i(t)$ is no smaller than $m(0)$. Thus, $c$ is positive. **Q.E.D.**

As in the agreement algorithm, the constant $c$ whose existence is asserted by Prop. 3.2 depends on the particular scenario. This does not cause any difficulties because $\pi^*$ can be recovered from the limiting value of $\pi(t)$ by normalizing it so that its entries add to 1.

If $P$ is stochastic and irreducible, but all of its diagonal entries are zero, the iteration of Eqs. (3.10) and (3.11) does not converge, in general (see Example 3.1). However, we can let

$$Q = \gamma P + (1 - \gamma)I,$$

and apply the algorithm with $P$ replaced by $Q$. Here $\gamma$ is a constant belonging to $(0, 1)$, and $I$ is the identity matrix. Then $Q$ satisfies the assumptions of Prop. 3.2, and we obtain convergence to a multiple of the invariant distribution vector of $Q$. This is all we need because it is easily seen that $P$ and $Q$ have the same invariant distribution.

As a final extension, suppose that $P$ is nonnegative, irreducible, with $\rho(P) = 1$, but not necessarily stochastic. Then the Perron–Frobenius theorem (Prop. 6.6 in Section 2.6), applied to the transpose of $P$, guarantees the existence of a positive row vector $\pi^*$ such that $\pi^* = \pi^* P$, and the proof of Prop. 3.2 remains valid without any modifications whatsoever.

## EXERCISES

**3.1.** **(a)** Suppose that Assumption 1.1(a) is replaced by the requirement that $T^i$ is infinite, for all $i$. Show that Props. 3.1 and 3.2 are no longer true.

**(b)** Suppose that Assumption 1.1(c) is replaced by the requirement $t - B + 1 \leq \tau_i^i(t) \leq t$ for all $i$ and $t \in T^i$. Show that Props. 3.1 and 3.2 are no longer true.

*Hints:* For part (a), let

$$A = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

and show that if each processor in turn executes a large number of iterations, the algorithm behaves similarly with the iteration $x_1 := x_3$, $x_3 := x_2$, and $x_2 := x_1$ executed in Gauss–Seidel fashion. For part (b), let

$$A = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

Arrange the initial conditions and a scenario so that for every $t$, we have $x(t) = x(t + 2)$, but $x(t + 1) \neq x(t)$.

**3.2.** We consider a variant of the agreement algorithm whereby processors receive messages from other processors, and upon reception, these messages are immediately taken into account by

forming convex combinations. Let $G = (N, A)$ be a directed graph, with $N = \{1, \ldots, n\}$ and with $(j, i) \in A$ if and only if processor $j$ communicates to processor $i$. For any $(j, i) \in A$, let $T_j^i$ be the set of times that processor $i$ receives a message $x_j\big(\tau_j^i(t)\big)$ from processor $j$. We assume that for any fixed $i$, the sets $T_j^i$, $j \neq i$, are disjoint. Let the algorithm be described by the equations

$$x_i(t+1) = a_{ij}x_j\big(\tau_j^i(t)\big) + (1 - a_{ij})x_i(t), \qquad t \in T_j^i,$$

and $x_i(t+1) = x_i(t)$ if $t$ does not belong to any $T_j^i$. Assume that $0 < a_{ij} < 1$ for every $i$ and $j$ such that $(j, i) \in A$.

**(a)** Reformulate appropriately Assumptions 1.1 and 3.1, redefine the constant $\alpha$ of Prop. 3.1, and then show that the conclusions of Prop. 3.1 hold for the present algorithm as well.

**(b)** What happens in this algorithm if one of the processors breaks down and stops transmitting any messages?

**(c)** Answer the question of part (b) for the original algorithm of Eqs. (3.2) and (3.3).

**3.3.** For each $i$ we let $\{\epsilon_i(t)\}$ be a sequence of real numbers that converges geometrically to zero. We consider a perturbed version of the agreement algorithm, whereby Eq. (3.3) is replaced by

$$x_i(t+1) = \sum_{j=1}^{n} a_{ij}x_j\big(\tau_j^i(t)\big) + \epsilon_i(t), \qquad \text{if } t \in T^i.$$

Let Assumptions 1.1 and 3.1 hold, and show that for every scenario and each $i$, the sequence $\{x_i(t)\}$ converges geometrically to a limit independent of $i$. *Hint:* Fix some positive integer $s$. For any scenario, define $v(t)$ by letting $v(t) = x(t)$ if $t \leq s$,

$$v_i(t+1) = \sum_{j=1}^{n} a_{ij}v_j\big(\tau_j^i(t)\big), \qquad \text{if } t \geq s,\ t \in T^i,$$

and $v_i(t+1) = v(t)$ if $t \geq s$ and $t \notin T^i$. Let $q(t) = \min_i \min_{t-B+1 \leq \tau \leq t} v_i(\tau)$ and $Q(t) = \max_i \max_{t-B+1 \leq \tau \leq t} v_i(\tau)$. From Prop. 3.1 we have $Q(s+2LB+B) - q(s+2LB+B) \leq \eta\big(Q(s) - q(s)\big)$. Furthermore, $x_i(s + 2LB + B) - v_i(s + 2LB + B)$ can be bounded by a constant which tends to zero geometrically as $s$ goes to infinity. Combine these two observations to show that $M(s+2B+B) - m(s+2B+B) \leq (1-\eta)\big(M(s) - m(s)\big) + \delta(s)$, where $\{\delta(s)\}$ is a sequence that converges to zero geometrically.

**3.4.** Let all of the assumptions in Prop. 3.2 hold except for the irreducibility of $P$. Assume instead that the Markov chain corresponding to $P$ has a single ergodic class and that the nonzero diagonal entry $p_{i^*i^*}$ corresponds to a recurrent state $i^*$. Show that the sequence $\{\pi(t)\}$ generated by the partially asynchronous iteration (3.10)–(3.11), initialized with positive values, converges geometrically to a positive multiple of the vector of invariant probabilities of the Markov chain. *Hint:* We are dealing with the asynchronous iterations $\pi^{(1)} := \pi^{(1)}P_{11}$ and $\pi^{(2)} := \pi^{(1)}P_{12} + \pi^{(2)}P_{22}$, where $\pi^{(1)}$ and $\pi^{(2)}$ are appropriate subvectors of $\pi$, and $P_{11}$, $P_{12}$ and $P_{22}$ are appropriate submatrices of $P$. Show that $\rho(P_{11}) < 1$ and that $\pi^{(1)}(t)$ converges to zero geometrically. For the second iteration, use a suitable change of variables and the result of Exercise 3.3.

## 7.4 LOAD BALANCING IN A COMPUTER NETWORK

Consider a network of $n$ processors, described by a connected undirected graph $G = (N, A)$, where $N = \{1, \ldots, n\}$, and $A$ is the set of arcs connecting different processors. These processors are cooperating in the execution of some computational task. We assume that the computation consists of the execution of independent subtasks and that each subtask can be executed by any processor. For maximum speed, it is desired to spread the subtasks among processors as evenly as possible. We consider a situation in which the load is initially distributed unevenly and we assume that the highly loaded processors try to remedy this by transferring some load to the less loaded processors. We shall assume that this process of load exchange takes place asynchronously, which is reasonable for a loosely coupled network of processors. Finally, we shall make the simplifying assumption that there is a very large number of very small subtasks, so that the load of each processor can be described by a continuous variable.

Let $x_i(t) \geq 0$ be the load handled by processor $i$ at time $t$, where $t$ is a nonnegative integer time variable. Let $L$ be the total load. Let $A(i)$ be the set of neighbors of the $i$th processor. Each processor $i$ keeps in its memory an estimate $x_j^i(t)$ of the load carried by each neighboring processor $j \in A(i)$. Due to communication delays and asynchronism, this estimate can be outdated and we assume that

$$x_j^i(t) = x_j\big(\tau_j^i(t)\big), \tag{4.1}$$

where $\tau_j^i(t)$ is an integer variable satisfying $0 \leq \tau_j^i(t) \leq t$.

There is a set of times $T^i$ at which processor $i$ compares its load with the load of its neighbors, and if it finds that it is overloaded, transfers some of its load, according to the following rule. For each neighboring processor $j \in A(i)$, if $x_i(t) > x_j^i(t)$, then a nonnegative amount of load, denoted by $s_{ij}(t)$, is transferred from $i$ to $j$; no load is transferred if $x_i(t) \leq x_j^i(t)$, in which case, we let $s_{ij}(t) = 0$. For notational convenience, we also let $s_{ij}(t) = 0$ if $t \notin T^i$. We assume that a load transfer can take some time to be completed. We use $v_{ij}(t)$ to denote the amount of load that has been sent from processor $i$ to processor $j$ before time $t$, but has not been received by processor $j$ before time $t$. Let $r_{ij}(t)$ be the load received by processor $j$ from processor $i$ at time $t$. We then have

$$x_i(t+1) = x_i(t) - \sum_{j \in A(i)} s_{ij}(t) + \sum_{j \in A(i)} r_{ji}(t) \tag{4.2}$$

and

$$v_{ij}(t) = \sum_{\tau=0}^{t-1} \big(s_{ij}(\tau) - r_{ij}(\tau)\big), \tag{4.3}$$

where we are making the implicit assumption that $v_{ij}(0) = 0$. Since no load is assumed to be in transit at time zero, we have $\sum_{i=1}^n x_i(0) = L$, and using Eqs. (4.2) and (4.3), we easily obtain the load conservation equation

$$\sum_{i=1}^{n}\Big(x_i(t)+\sum_{j\in A(i)}v_{ij}(t)\Big)=L, \qquad \forall t\geq 0. \tag{4.4}$$

We now introduce two assumptions without which load equalization can fail to take place asymptotically (Exercise 4.1).

**Assumption 4.1.** (*Partial Asynchronism*) There exists a positive integer $B$ such that:

**(a)** For every $i$ and for every $t\geq 0$, at least one of the elements of the set $\{t,t+1,\ldots,t+B-1\}$ belongs to $T^i$.

**(b)** There holds

$$t-B<\tau_j^i(t)\leq t,$$

for all $i$ and $t$, and all $j\in A(i)$.

**(c)** The load $s_{ij}(t)$ sent from processor $i$ to processor $j$ at some time $t\in T^i$ is received by processor $j$ before time $t+B$.

Assumption 4.1(c) implies that $v_{ij}(t)$, the load in transit just before time $t$, consists exclusively of pieces of the load that were sent during the time interval $(t-B,t-1]$. Accordingly, we have

$$v_{ij}(t)\leq\sum_{\tau=t-B+1}^{t-1}s_{ij}(\tau), \qquad \forall i,\ \forall j\in A(i). \tag{4.5}$$

The next assumption has two parts. Part (a) postulates that when processor $i$ detects a load imbalance, it will transfer a nonnegligible portion of its excess load to some lightest loaded neighbor. Of course, that does not preclude that it also sends some load to other neighbors as well. However, no load is transferred to neighbors carrying a larger load. Part (b) prohibits processor $i$ from transferring a very large amount of load and creating a load imbalance in the opposite direction. The latter assumption is introduced in order to preclude the possibility that two nodes keep sending load to each other back and forth, without ever reaching equilibrium.

**Assumption 4.2.**

**(a)** There exists some constant $\alpha\in(0,1)$ such that for every $i$ and every $t\in T^i$, there exists some $j\in A(i)$ satisfying $x_j^i(t)=\min_{k\in A(i)}x_k^i(t)$ and $s_{ij}(t)\geq\alpha\big(x_i(t)-x_j^i(t)\big)$. Furthermore, if $x_i(t)\leq x_j^i(t)$, then $s_{ij}(t)=0$.

**(b)** For any $i$, any $t\in T^i$, and any $j\in A(i)$ such that $x_i(t)>x_j^i(t)$, we have

$$x_i(t)-\sum_{k\in A(i)}s_{ik}(t)\geq x_j^i(t)+s_{ij}(t). \tag{4.6}$$

The algorithm of this section resembles the agreement algorithm in certain respects, but its convergence cannot be derived from the corresponding result for the agreement algorithm. Still, the general ideas in the proof of convergence are similar. In particular, we will show that if there is a load imbalance, then the load of the lightest loaded processors will eventually increase, thus reducing the load imbalance. The main result is the following.

**Proposition 4.1.** Under Assumptions 4.1 and 4.2, we have $\lim_{t\to\infty} x_i(t) = L/n$ for all $i$.

***Proof.*** For notational convenience, we define $x_i(t) = x_i(0)$ for $t < 0$. Let

$$m(t) = \min_i \min_{t-B<\tau\leq t} x_i(\tau), \tag{4.7}$$

and notice that $x_j^i(t) = x_j\big(\tau_j^i(t)\big) \geq m(t)$, for every $i$, $j$, and $t$. The proof is based on the following lemmas:

**Lemma 4.1.** There exists some $\beta \in (0,1)$ such that

$$x_i(t+1) \geq m(t) + \beta\big(x_i(t) - m(t)\big), \qquad \forall i, t. \tag{4.8}$$

In particular, $\beta$ can be chosen equal to $1/n$.

***Proof.*** Let us fix some $i$ and $t$. If $t \notin T^i$, then $x_i(t+1) = x_i(t) = m(t) + \big(x_i(t) - m(t)\big)$. Since $x_i(t)$ is larger than or equal to $m(t)$, inequality (4.8) follows for any $\beta \in (0,1)$. Now suppose that $t \in T^i$, consider the set $A' = \{j \mid j \in A(i),\ x_i(t) > x_j^i(t)\}$, and let $K$ be its cardinality. Equations (4.2) and (4.6) imply that $x_i(t+1) \geq x_j^i(t) + s_{ij}(t)$ for all $j \in A'$, and adding over all $j \in A'$, we obtain

$$Kx_i(t+1) \geq \sum_{j\in A'} x_j^i(t) + \sum_{j\in A'} s_{ij}(t). \tag{4.9}$$

Furthermore, from Eq. (4.2) and the fact $s_{ij}(t) = 0$ for $j \notin A'$, we obtain

$$\sum_{j\in A'} s_{ij}(t) = \sum_{j\in A(i)} s_{ij}(t) \geq x_i(t) - x_i(t+1). \tag{4.10}$$

Combining Eqs. (4.9) and (4.10) and using the definition of $m(t)$, we obtain $Kx_i(t+1) \geq Km(t) + \big(x_i(t) - x_i(t+1)\big)$, which can be rewritten as

$$x_i(t+1) \geq \frac{K}{K+1} m(t) + \frac{1}{K+1} x_i(t) = m(t) + \frac{1}{K+1}\big(x_i(t) - m(t)\big) \geq m(t) + \frac{1}{n}\big(x_i(t) - m(t)\big).$$

This proves inequality (4.8) with $\beta = 1/n$. **Q.E.D.**

**Lemma 4.2.**

**(a)** The sequence $m(t)$ is nondecreasing and converges.

**(b)** For every $i$ and every $t, s \geq 0$, we have

$$x_i(t+s) \geq m(t) + \beta^s\big(x_i(t) - m(t)\big), \tag{4.11}$$

where $\beta$ is the constant of Lemma 4.1.

***Proof.***

**(a)** Lemma 4.1 implies that $x_i(t+1) \geq m(t)$ and the inequality $m(t+1) \geq m(t)$ follows. Furthermore, $x_i(t)$ is bounded by the total load $L$. Therefore, $m(t)$ is also bounded by $L$, and since it is monotonic, it converges.

**(b)** The proof is by induction on $s$. Lemma 4.1 shows that inequality (4.11) is true for $s = 1$. Assuming that it is true for some general $s$, we use again Lemma 4.1 and the monotonicity of $m(t)$ to obtain

$$\begin{aligned} x_i(t+s+1) \geq m(t+s) + \beta\big(x_i(t+s) - m(t+s)\big) &\geq m(t) + \beta\big(x_i(t+s) - m(t)\big) \\ &\geq m(t) + \beta^{s+1}\big(x_i(t) - m(t)\big), \end{aligned}$$

where the last step follows from the induction hypothesis. **Q.E.D.**

We could have also considered using the variable $M(t) = \max_i \max_{t-B<\tau\leq t} x_i(\tau)$ in the analysis. However, unlike the agreement algorithm of Section 7.3, this variable is not monotonically nonincreasing (Exercise 4.2). This is a major difference with the agreement algorithm (compare with Lemma 3.1 of Section 7.3).

Let us fix a processor $i$ and some time $t_0$. For any $j \in A(i)$ and any time $t \geq t_0$, we say that the event $E_j(t)$ occurs if the following two conditions are true:

**(i)**
$$x_j^i(t) < m(t_0) + \tfrac{\alpha}{2}\beta^{t-t_0}\big(x_i(t_0) - m(t_0)\big). \tag{4.12}$$

**(ii)** Processor $j$ is a lightest loaded neighbor of $i$ to which some load is transferred, as in Assumption 4.2(a), and in particular,

$$s_{ij}(t) \geq \alpha\big(x_i(t) - x_j^i(t)\big). \tag{4.13}$$

**Lemma 4.3.** If $j \in A(i)$, $t_1 \geq t_0$, $t_1 \in T^i$, and the event $E_j(t_1)$ occurs, then the event $E_j(\tau)$ does not occur, for any $\tau \geq t_1 + 2B$ such that $\tau \in T^i$.

***Proof.*** Suppose that $t_1 \geq t_0$, $t_1 \in T^i$, $j \in A(i)$, and the event $E_j(t_1)$ occurs. In particular, inequalities (4.12) and (4.13) are valid for $t = t_1$. Lemma 4.2(b) yields

$$x_i(t_1) \geq m(t_0) + \beta^{t_1-t_0}\big(x_i(t_0) - m(t_0)\big). \tag{4.14}$$

We subtract inequality (4.12) (with $t = t_1$) from (4.14) and use the fact $\alpha < 1$ to obtain

$$x_i(t_1) - x_j^i(t_1) \geq \tfrac{1}{2}\beta^{t_1 - t_0}\big(x_i(t_0) - m(t_0)\big).$$

Then, inequality (4.13) (with $t = t_1$) yields

$$s_{ij}(t_1) \geq \frac{\alpha}{2}\beta^{t_1 - t_0}\big(x_i(t_0) - m(t_0)\big). \tag{4.15}$$

Processor $j$ will receive the load $s_{ij}(t_1)$ at some time $t_2$ satisfying $t_1 \leq t_2 < t_1 + B$ [Assumption 4.1(c)]. If $t_2 \notin T^j$ or, more generally, if $s_{jk}(t_2) = 0$ for all $k \in A(j)$, then Eq. (4.2) yields

$$\begin{aligned} x_j(t_2 + 1) &= x_j(t_2) + \sum_{k \in A(j)} r_{kj}(t_2) \geq x_j(t_2) + r_{ij}(t_2) \\ &\geq x_j(t_2) + s_{ij}(t_1) \geq m(t_2) + s_{ij}(t_1). \end{aligned}$$

If, on the other hand, $t_2 \in T^j$ and there exists some $k^* \in A(j)$ such that $s_{jk^*}(t_2) > 0$, then Eqs. (4.2) and (4.6) yield

$$x_j(t_2+1) = x_j(t_2) - \sum_{k \in A(j)} s_{jk}(t_2) + \sum_{k \in A(j)} r_{kj}(t_2) \geq x_{k^*}^j(t_2) + r_{ij}(t_2) \geq m(t_2) + s_{ij}(t_1).$$

In both cases, we have using inequality (4.15),

$$\begin{aligned} x_j(t_2 + 1) &\geq m(t_2) + s_{ij}(t_1) \geq m(t_0) + \frac{\alpha}{2}\beta^{t_1 - t_0}\big(x_i(t_0) - m(t_0)\big) \\ &\geq m(t_0) + \frac{\alpha}{2}\beta^{t_2 + 1 - t_0}\big(x_i(t_0) - m(t_0)\big). \end{aligned}$$

We use Lemma 4.2 to conclude that

$$\begin{aligned} x_j(t) &\geq m(t_2 + 1) + \beta^{t - t_2 - 1}\big(x_j(t_2 + 1) - m(t_2 + 1)\big) \\ &\geq m(t_0) + \beta^{t - t_2 - 1}\big(x_j(t_2 + 1) - m(t_0)\big) \\ &\geq m(t_0) + \beta^{t - t_2 - 1}\frac{\alpha}{2}\beta^{t_2 + 1 - t_0}\big(x_i(t_0) - m(t_0)\big) \\ &\geq m(t_0) + \frac{\alpha}{2}\beta^{t - t_0}\big(x_i(t_0) - m(t_0)\big), \qquad \forall t \geq t_2 + 1. \end{aligned} \tag{4.16}$$

Since $t_2 < t_1 + B$, we see that inequality (4.16) holds for all $t \geq t_1 + B$.

Let $t_3$ satisfy $t_3 \in T^i$ and $t_3 \geq t_1 + 2B$. Then $\tau_j^i(t_3) > t_3 - B \geq t_1 + B$. Inequality (4.16) applies and yields

$$\begin{aligned} x_j^i(t_3) - m(t_0) = x_j\big(\tau_j^i(t_3)\big) - m(t_0) &\geq \frac{\alpha}{2}\beta^{\tau_j^i(t_3) - t_0}\big(x_i(t_0) - m(t_0)\big) \\ &\geq \frac{\alpha}{2}\beta^{t_3 - t_0}\big(x_i(t_0) - m(t_0)\big). \end{aligned}$$

Thus, inequality (4.12) does not hold for $t = t_3$, and the event $E_j(t_3)$ does not occur. **Q.E.D.**

**Lemma 4.4.** There exists some $\eta > 0$ such that for any $i$, $t_0$, $j \in A(i)$, and any $t \geq t_0 + 3nB$, we have

$$x_j(t) \geq m(t_0) + \eta\beta^{t-t_0}\big(x_i(t_0) - m(t_0)\big).$$

***Proof.*** Let us fix $i$ and $t_0$. Let $t_1, \ldots, t_n$ be elements of $T^i$ such that $t_{k-1} + 2B < t_k \leq t_{k-1} + 3B$, $k = 1, \ldots, n$. According to Lemma 4.3, if $j \in A(i)$ and $k \neq \ell$, then the events $E_j(t_k)$ and $E_j(t_\ell)$ cannot both occur. It follows that for some $t_k$ ($1 \leq k \leq n$), the event $E_j(t_k)$ does not occur for any $j \in A(i)$. According to Assumption 4.2(a) inequality (4.13) must be true for some $j^*$ satisfying

$$x^i_{j^*}(t_k) \leq x^i_j(t_k), \qquad \forall j \in A(i).$$

Since $E_{j^*}(t_k)$ does not occur, inequality (4.12) must be violated for $j = j^*$, and we obtain

$$x_j\big(\tau^i_j(t_k)\big) = x^i_j(t_k) \geq x^i_{j^*}(t_k) \geq m(t_0) + \frac{\alpha}{2}\beta^{t_k - t_0}\big(x_i(t_0) - m(t_0)\big), \qquad \forall j \in A(i).$$

For any $t$ satisfying $t \geq t_0 + 3nB$, we have $t \geq t_k \geq \tau^i_j(t_k)$ and Lemma 4.2 yields

$$\begin{aligned}
x_j(t) &\geq m\big(\tau^i_j(t_k)\big) + \beta^{t-\tau^i_j(t_k)}\Big(x_j\big(\tau^i_j(t_k)\big) - m\big(\tau^i_j(t_k)\big)\Big) \\
&\geq m(t_0) + \beta^{t-\tau^i_j(t_k)}\Big(x_j\big(\tau^i_j(t_k)\big) - m(t_0)\Big) \\
&\geq m(t_0) + \beta^{t-\tau^i_j(t_k)}\frac{\alpha}{2}\beta^{t_k - t_0}\big(x_i(t_0) - m(t_0)\big) \\
&\geq m(t_0) + \frac{\alpha}{2}\beta^B\beta^{t-t_0}\big(x_i(t_0) - m(t_0)\big),
\end{aligned}$$

where the last inequality follows because $t_k - \tau^i_j(t_k) \leq B$. This proves the lemma with $\eta = \alpha\beta^B/2$. **Q.E.D.**

By repeatedly applying Lemma 4.4, we obtain:

**Lemma 4.5.** For any $i$, $t_0$, any $j$ that can be reached from $i$ by traversing $\ell$ arcs, and for any $t \geq t_0 + 3\ell nB$, we have

$$x_j(t) \geq m(t_0) + \big(\eta\beta^{t-t_0}\big)^\ell\big(x_i(t_0) - m(t_0)\big).$$

***Proof.*** Lemma 4.4 establishes Lemma 4.5 for the case $\ell = 1$. We proceed by induction on $\ell$. Assume that the result is true for every $j$ at distance $\ell$ from $i$. Suppose that $k$ is at distance $\ell + 1$ from $i$. Then $k \in A(j)$ for some $j$ at distance $\ell$ from $i$. We apply the induction hypothesis to processor $j$ to obtain

$$x_j(t_0 + 3\ell nB) \geq m(t_0) + \left(\eta\beta^{3\ell nB}\right)^{\ell}\left(x_i(t_0) - m(t_0)\right).$$

We then apply Lemma 4.4 to processor $k$, with $t_0$ replaced by $t_0 + 3\ell nB$ to obtain for any $t \geq t_0 + 3\ell nB + 3nB$,

$$\begin{aligned} x_k(t) &\geq m(t_0 + 3\ell nB) + \eta\beta^{t-t_0-3\ell nB}\left(x_j(t_0 + 3\ell nB) - m(t_0 + 3\ell nB)\right) \\ &\geq m(t_0) + \eta\beta^{t-t_0-3\ell nB}(\eta\beta^{3\ell nB})^{\ell}\left(x_i(t_0) - m(t_0)\right) \\ &\geq m(t_0) + \eta\beta^{t-t_0}(\eta\beta^{t-t_0})^{\ell}\left(x_i(t_0) - m(t_0)\right) \\ &= m(t_0) + (\eta\beta^{t-t_0})^{\ell+1}\left(x_i(t_0) - m(t_0)\right), \end{aligned}$$

which concludes the induction and proves the lemma. **Q.E.D.**

We now conclude the proof of the proposition. Let us again fix some processor $i$ and a time $t_0$. Since every processor is at a distance smaller than $n$ from $i$, Lemma 4.5 yields

$$x_j(t) \geq m(t_0) + \left(\eta\beta^{3n^2B+B}\right)^{n}\left(x_i(t_0) - m(t_0)\right), \qquad \forall j,\ \forall t \in \left[t_0 + 3n^2B,\, t_0 + 3n^2B + B\right].$$

Consequently, there exists some $\delta > 0$ such that

$$m(t_0 + 3n^2B + B) \geq m(t_0) + \delta\left(x_i(t_0) - m(t_0)\right).$$

This inequality is true for every $i$ and therefore,

$$m(t_0 + 3n^2B + B) \geq m(t_0) + \delta\left(\max_i x_i(t_0) - m(t_0)\right). \tag{4.17}$$

If $\max_i x_i(t) - m(t)$ does not converge to zero, then inequality (4.17) shows that $m(t)$ will increase to infinity, which contradicts the boundedness of $m(t)$. Therefore, $\max_i x_i(t) - m(t)$ converges to zero. Furthermore, $m(t)$ converges to some constant $c$ [Lemma 4.2(a)]. It follows that $\lim_{t\to\infty} \max_i x_i(t) = c$. Since $m(t) \leq x_j(t) \leq \max_i x_i(t)$ for every $j$ and $t$, we see that $x_j(t)$ also converges to $c$, for every $j$. In view of inequality (4.6) we obtain $\lim_{t\to\infty} s_{ij}(t) = 0$ for all $(i,j) \in A$. We then use inequality (4.5) to conclude that $\lim_{t\to\infty} v_{ij}(t) = 0$ for all $(i,j) \in A$. Using Eq. (4.4) we obtain $nc = \lim_{t\to\infty} \sum_{i=1}^{n} x_i(t) = L$, from which we conclude that $c = L/n$. **Q.E.D.**

It can be shown that convergence of the algorithm takes place geometrically (Exercise 4.3).

It should be noted that this algorithm operates without the processors knowing the value of $L$. If the value of $L$ changes to $L'$ while the algorithm is executed, then each $x_i(t)$ will converge to $L'/n$, and this happens without the processors being informed of the change from $L$ to $L'$. Thus, the algorithm is able to adapt to changes in the problem data without a need for being restarted, which is a characteristic feature of asynchronous algorithms.

## EXERCISES

**4.1.** Show by means of examples that Prop. 4.1 fails to hold in the following cases:

**(a)** If part (b) of Assumption 4.1 is replaced by the assumption that $\lim_{t\to\infty} \tau_j^i(t) = \infty$.

**(b)** If part (c) of Assumption 4.1 is replaced by the assumption that the load sent by any processor eventually reaches its destination.

**(c)** If part (a) of Assumption 4.2 is replaced by the following requirement: for any $t \in T^i$, let $A' = \{j \mid j \in A(i), x_j^i(t) < x_i(t)\}$, and assume that if $A'$ is nonempty, there exists some $j \in A'$ such that $s_{ij}(t) \geq \alpha\big(x_i(t) - x_j^i(t)\big)$, where $\alpha$ is a constant belonging to $(0, 1)$.

**(d)** If part (b) of Assumption 4.2 is replaced by the requirement that $x_i(t) - \sum_{k\in A(i)} s_{ik}(t) \geq x_j^i(t)$, for all $j \in A(i)$ and all $t \in T^i$.

**4.2.** Let $M(t) = \max_i \max_{t-B\leq\tau\leq t} x_i(\tau)$. Show by means of an example that $M(t)$ is not necessarily monotonically nonincreasing.

**4.3.** **(Geometric Convergence.)** Let, for each $i$ and $t$,

$$y_i(t) = x_i(t) + \sum_{j\in A(i)} v_{ji}(t).$$

**(a)** Fix some $t$. Show that there exists some $t' \in [t+1, t+B]$ for which

$$\sum_{j\in A(i)} r_{ji}(t') \geq \frac{y_i(t) - x_i(t)}{B}.$$

**(b)** Show that for some $\beta \in (0, 1)$,

$$x_i(t'+1) \geq m(t) + \beta^{t'-t}\big(x_i(t) - m(t)\big) + \frac{y_i(t) - x_i(t)}{B}.$$

**(c)** Show that there exists some $\gamma > 0$ such that

$$x_i(t+B+1) \geq m(t) + \gamma\big(y_i(t) - m(t)\big).$$

**(d)** Show that

$$\max_i x_i(t+B+1) - m(t) \geq \gamma\left(\frac{L}{n} - m(t)\right).$$

**(e)** Use inequality (4.17) to prove that $m(t)$, as well as each $x_i(t)$, converges geometrically to $L/n$.

**4.4.** **[Cyb87]** Suppose that the network of processors is a $d$–cube and consider the following synchronous algorithm. At the $i$th phase of the algorithm ($i = 1, \ldots, d$) each processor exchanges load with its neighbor whose identity differs in the $i$th bit. Furthermore, the load exchanged is such that after the exchange, the load of these two processors is the same. Show that all processors have the same load at the end of the $d$th phase.

## 7.5 GRADIENT–LIKE OPTIMIZATION ALGORITHMS

In this section, we take a new look at gradient–like algorithms for unconstrained and constrained optimization. We recall that the totally asynchronous gradient algorithm is guaranteed to converge if the Hessian matrix $\nabla^2 F$ of the cost function satisfies a diagonal dominance condition, guaranteeing that the iteration $x := x - \gamma \nabla F(x)$ is a maximum norm contraction mapping for sufficiently small $\gamma$ (see Section 6.3). On the other hand, if such a condition is not satisfied, then the totally asynchronous algorithm can be nonconvergent, no matter how small the stepsize $\gamma$ is chosen, as demonstrated in Example 3.1 of Subsection 6.3.2. We will show that the desired convergence properties of asynchronous gradient–like algorithms are recovered once we impose the assumption of partial asynchronism.

The results to be derived in this section differ from those of Sections 7.2–7.4 in an important respect. In those sections, we established convergence of certain algorithms under the assumption that the asynchronism measure $B$ was finite but otherwise arbitrary. In contrast, in this section, we can establish convergence of the iteration $x := x - \gamma \nabla F(x)$ only if the stepsize $\gamma$ is small compared to $1/B$; equivalently, only if $B$ is small compared to $1/\gamma$. This is not a deficiency of the method of proof. As demonstrated in Example 1.3 of Section 7.1, the partially asynchronous gradient algorithm, with a fixed value of $\gamma$, does not converge, in general, if $B$ is sufficiently large. In other words, asynchronous gradient–like algorithms can only tolerate a limited amount of asynchronism.

Throughout this section, $\|\cdot\|$ stands for the Euclidean norm $\|\cdot\|_2$.

### 7.5.1 The Algorithm and its Convergence

Let $F : \Re^n \mapsto \Re$ be a cost function to be minimized subject to no constraints. We impose the same assumptions on $F$ as in Section 3.2, which dealt with synchronous algorithms for unconstrained optimization.

**Assumption 5.1.**

**(a)** There holds $F(x) \geq 0$ for every $x \in \Re^n$.

**(b)** (*Lipschitz Continuity of* $\nabla F$) The function $F$ is continuously differentiable and there exists a constant $K_1$ such that

$$\|\nabla F(x) - \nabla F(y)\| \leq K_1 \|x - y\|, \qquad \forall x, y \in \Re^n. \tag{5.1}$$

The asynchronous gradient algorithm is the asynchronous iteration

$$x := x - \gamma \nabla F(x). \tag{5.2}$$

A more general version will be considered here. Our model is similar to the one introduced in Section 7.1 and is the following. The algorithm iterates on a vector $x(t)$ whose $i$th coordinate, denoted by $x_i(t)$, is updated by processor $i$ according to

$$x_i(t+1) = x_i(t) + \gamma s_i(t), \qquad i = 1, \ldots, n, \tag{5.3}$$

where $\gamma$ is a positive stepsize, and $s_i(t)$ is the update direction. We let $T^i$ be the set of times when the $i$th processor performs an update. Thus, we assume that

$$s_i(t) = 0, \qquad \forall t \notin T^i. \tag{5.4}$$

Processor $i$ has knowledge at any time $t$ of a vector $x^i(t)$ that is a possibly outdated version of $x(t)$. That is,

$$x^i(t) = \Big(x_1\big(\tau_1^i(t)\big), \ldots, x_n\big(\tau_n^i(t)\big)\Big). \tag{5.5}$$

Here the variables $\tau_j^i(t)$ have the same meaning as in the model of Section 7.1. They are assumed to be defined for all $t$ and to satisfy $0 \leq \tau_j^i(t) \leq t$ for all $t \geq 0$.

For those times $t$ that belong to $T^i$, we assume that the update direction is such that the cost function does not increase; equivalently, $s_i(t)$ should have the opposite sign from $\nabla_i F(x)$, where $x$ is the vector in the memory of the processor in charge of this update. We thus assume the following:

**Assumption 5.2.**

**(a)** (*Descent Property Along Each Coordinate*) For every $i$ and $t$, we have

$$s_i(t)\nabla_i F\big(x^i(t)\big) \leq 0. \tag{5.6}$$

**(b)** There exist positive constants $K_2$ and $K_3$ such that

$$K_2\big|\nabla_i F\big(x^i(t)\big)\big| \leq |s_i(t)| \leq K_3\big|\nabla_i F\big(x^i(t)\big)\big|, \quad \forall t \in T^i, \ \forall i. \tag{5.7}$$

The lower bound on $|s_i(t)|$ provided by Eq. (5.7) is introduced because otherwise $s_i(t)$ could be always equal to zero, in which case, convergence cannot be demonstrated. The upper bound in Eq. (5.7) is also essential, because otherwise $s_i(t)$ could be chosen so large that the algorithm would exhibit unstable oscillations.

Assumption 5.2 roughly requires that $s(t)$ is in the same quadrant as $-\nabla F(x)$ and is quite general. For example, if we let $s_i(t) = -\nabla_i F\big(x^i(t)\big)$, for $t \in T^i$, we recover the asynchronous gradient algorithm of Eq. (5.2). For this example, part (a) of Assumption 5.2 is satisfied and part (b) holds with $K_2 = K_3 = 1$.

As another example, consider an asynchronous Jacobi algorithm (cf. Section 3.2). In this case we assume that $F$ is twice differentiable and we let

$$s_i(t) = -\frac{\nabla_i F\big(x^i(t)\big)}{\nabla_{ii}^2 F\big(x^i(t)\big)}, \quad t \in T^i.$$

Assumption 5.2 is satisfied provided that the cost function $F$ has the property $0 < K_2 \leq 1/\nabla_{ii}^2 F(x) \leq K_3$ for some constants $K_2$ and $K_3$ and for all $x \in \Re^n$.

The partial asynchronism assumption that follows is almost identical to Assumption 1.1 of Section 7.1, except that we do not need the condition $\tau_i^i(t) = t$.

**Assumption 5.3.** (*Partial Asynchronism*) There exists a positive integer $B$ such that:

**(a)** For every $i$ and for every $t \geq 0$, at least one of the elements of the set $\{t, t+1, \ldots, t+B-1\}$ belongs to $T^i$.

**(b)** There holds

$$\max\{0, t-B+1\} \leq \tau_j^i(t) \leq t,$$

for all $i$ and $j$ and all $t \geq 0$.

Proposition 5.1 to follow states that under the above assumptions, convergence is obtained provided that the stepsize is small enough. The proof follows the same steps as the proof of convergence of synchronous gradient–like algorithms (Props. 2.1 and 2.2 in Section 3.2) and the reader is advised to consult these proofs at this point. The main difference is that asynchronism introduces a few additional error terms, because the processors are using outdated information. Given the bound $B$ of Assumption 5.3 and given the fact that the processors can only make small steps ($\gamma$ is small), we show that these error terms are small enough to be inconsequential.

**Proposition 5.1.** Under Assumptions 5.1–5.3, there exists some $\gamma_0 > 0$ (depending on $n$, $B$, $K_1$, and $K_3$) such that if $0 < \gamma < \gamma_0$, then $\lim_{t\to\infty} \nabla F\big(x(t)\big) = 0$.

***Proof.*** For notational convenience, we define $s(t)$ to be zero for $t < 0$. We use the descent lemma (Prop. A.32 in Appendix A) and Assumptions 5.1 and 5.2, to obtain

$$\begin{aligned}
F\big(x(t+1)\big) =& F\big(x(t)+\gamma s(t)\big) \\
\leq& F\big(x(t)\big) + \gamma \sum_{i=1}^n s_i(t)\nabla_i F\big(x(t)\big) + K_1\gamma^2\|s(t)\|^2 \\
=& F\big(x(t)\big) + \gamma \sum_{i=1}^n s_i(t)\nabla_i F\big(x^i(t)\big) + \gamma \sum_{i=1}^n s_i(t)\Big(\nabla_i F\big(x(t)\big) - \nabla_i F\big(x^i(t)\big)\Big) \\
&+ K_1\gamma^2\|s(t)\|^2 \qquad (5.8) \\
\leq& F\big(x(t)\big) - \gamma \sum_{i=1}^n \frac{1}{K_3}|s_i(t)|^2 + \gamma K_1 \sum_{i=1}^n |s_i(t)| \cdot \|x(t) - x^i(t)\| + K_1\gamma^2\|s(t)\|^2.
\end{aligned}$$

We now proceed to bound $\|x^i(t) - x(t)\|$. We have

$$|x_j^i(t) - x_j(t)| = \big|x_j\big(\tau_j^i(t)\big) - x_j(t)\big| = \gamma\Big|\sum_{\tau=\tau_j^i(t)}^{t-1} s_j(\tau)\Big| \leq \gamma \sum_{\tau=t-B}^{t-1} |s_j(\tau)|.$$

Therefore, the inequality

$$|x^i(t) - x(t)| \leq \gamma \sum_{\tau=t-B}^{t-1} |s(\tau)|$$

holds (componentwise). Using this and the triangle inequality, we obtain

$$\|x^i(t) - x(t)\| \leq \gamma \Big\| \sum_{\tau=t-B}^{t-1} |s(\tau)| \Big\| \leq \gamma \sum_{\tau=t-B}^{t-1} \|s(\tau)\|. \tag{5.9}$$

We now substitute inequality (5.9) in inequality (5.8) to obtain

$$F\big(x(t+1)\big) \leq F\big(x(t)\big) - \gamma \left(\frac{1}{K_3} - K_1\gamma\right) \|s(t)\|^2 + \gamma^2 K_1 \sum_{i=1}^{n} |s_i(t)| \sum_{\tau=t-B}^{t-1} \|s(\tau)\|.$$

We then use the inequality

$$|s_i(t)| \cdot \|s(\tau)\| \leq |s_i(t)|^2 + \|s(\tau)\|^2,$$

to obtain

$$\begin{aligned} F\big(x(t+1)\big) \leq & F\big(x(t)\big) - \gamma \left(\frac{1}{K_3} - K_1\gamma\right) \|s(t)\|^2 + \gamma^2 K_1 \sum_{i=1}^{n} \sum_{\tau=t-B}^{t-1} \Big(|s_i(t)|^2 + \|s(\tau)\|^2\Big) \\ = & F\big(x(t)\big) - \gamma \left(\frac{1}{K_3} - K_1\gamma\right) \|s(t)\|^2 + \gamma^2 K_1 \left(B\|s(t)\|^2 + n \sum_{\tau=t-B}^{t-1} \|s(\tau)\|^2\right) \\ = & F\big(x(t)\big) - \gamma \left(\frac{1}{K_3} - K_1\gamma - K_1\gamma B\right) \|s(t)\|^2 + \gamma^2 n K_1 \sum_{\tau=t-B}^{t-1} \|s(\tau)\|^2. \end{aligned} \tag{5.10}$$

We have an inequality of the form (5.10) for each different $t$. By adding those inequalities, we obtain

$$\begin{aligned} F\big(x(t+1)\big) \leq & F\big(x(0)\big) - \gamma \left(\frac{1}{K_3} - K_1\gamma - K_1\gamma B\right) \sum_{\tau=0}^{t} \|s(\tau)\|^2 + nB\gamma^2 K_1 \sum_{\tau=0}^{t} \|s(\tau)\|^2 \\ = & F\big(x(0)\big) - \gamma \left(\frac{1}{K_3} - K_1\gamma - K_1\gamma B - nB\gamma K_1\right) \sum_{\tau=0}^{t} \|s(\tau)\|^2. \end{aligned} \tag{5.11}$$

Let $\gamma_0 = 1/\big[(1 + B + nB)K_1K_3\big]$, and assume that $\gamma \in (0, \gamma_0)$. Then $C = (1/K_3) - K_1\gamma - K_1\gamma B - nBK_1\gamma$ is positive, and we can divide both sides of inequality (5.11)

by $C$, without reversing the inequality. Since $F$ is assumed nonnegative, we have $F\big(x(t+1)\big) \geq 0$ and inequality (5.11) becomes

$$\sum_{\tau=0}^{t} \|s(\tau)\|^2 \leq \frac{1}{C\gamma} F\big(x(0)\big). \tag{5.12}$$

Inequality (5.12) holds for every $t \geq 0$. Therefore,

$$\sum_{\tau=0}^{\infty} \|s(\tau)\|^2 \leq \frac{1}{C\gamma} F\big(x(0)\big) < \infty,$$

which implies that

$$\lim_{t\to\infty} s(t) = 0. \tag{5.13}$$

Using Eq. (5.3), we conclude that

$$\lim_{t\to\infty} \|x(t+1) - x(t)\| = 0, \tag{5.14}$$

and using inequality (5.9) we also obtain

$$\lim_{t\to\infty} \|x^i(t) - x(t)\| = 0. \tag{5.15}$$

We now combine Eq. (5.13) and the left hand side of Eq. (5.7) to conclude that $\nabla_i F\big(x^i(t)\big)$ converges to zero, as $t \to \infty$, along any sequence of times that belong to $T^i$. Given any time $t$, let $r_i(t)$ be an element of $T^i$ such that $|t - r_i(t)| \leq B$. Such an $r_i(t)$ exists because of Assumption 5.3(a). As $t$ tends to infinity, $r_i(t)$ also tends to infinity, and we have $\lim_{t\to\infty} \nabla_i F\big(x^i\big(r_i(t)\big)\big) = 0$. Using Eq. (5.15) and the Lipschitz continuity of $\nabla F$, we obtain $\lim_{t\to\infty} \nabla_i F\big(x\big(r_i(t)\big)\big) = 0$; finally, Eq. (5.14) implies that $\lim_{t\to\infty} \big\|x(t) - x\big(r_i(t)\big)\big\| = 0$, which, together with the Lipschitz continuity of $\nabla F$, implies that $\lim_{t\to\infty} \nabla_i F\big(x(t)\big) = 0$ and completes the proof of the proposition. **Q.E.D.**

Suppose now that $F(x) = \frac{1}{2}x'Ax$, where $A$ is a positive definite symmetric matrix such that the totally asynchronous version of the gradient algorithm $x := x - \gamma Ax$ diverges for every choice of $\gamma > 0$. (The existence of such a matrix was demonstrated in Example 3.1 of Subsection 6.3.2.) Proposition 3.1 of Section 6.3 shows that $\rho(|I-\gamma A|) \geq 1$ for every $\gamma > 0$, and, furthermore, for any $\gamma > 0$, there exists a scenario, with $B = 2$, under which the asynchronous algorithm diverges. On the other hand, Prop. 5.1 shows that if $B = 2$ and $\gamma$ is sufficiently small, then the asynchronous algorithm converges. The reason for the apparent contradiction lies in a minor difference between the model of this section and the model of Section 6.3. In particular, according to the model of Section 6.3, the asynchronous iteration $x := x - \gamma Ax$ is described by the equation

$$x_i(t+1) = x_i\big(\tau_i^i(t)\big) - \gamma \sum_{j=1}^{n} a_{ij} x_j\big(\tau_j^i(t)\big), \quad t \in T^i,$$

whereas the asynchronous gradient algorithm of this section, applied to the minimization of $\frac{1}{2}x'Ax$, is described by

$$x_i(t+1) = x_i(t) - \gamma \sum_{j=1}^{n} a_{ij} x_j\big(\tau_j^i(t)\big), \quad t \in T^i.$$

The two iterations are different if we allow $\tau_i^i(t)$ to be different from $t$. As we have just seen, this seemingly small difference is important enough to affect convergence.

### 7.5.2 The Role of the Various Parameters

There are some interesting qualitative observations that can be made regarding the relationship between the maximum stepsize $\gamma_0$, the allowed amount of asynchronism $B$, and the structure of the cost function, the latter being captured by the various constants $K_i$ introduced in the assumptions. Let us recall that in the course of the proof of Prop. 5.1, we found the choice

$$\gamma_0 = \frac{1}{K_3 K_1 (1 + B + nB)} \tag{5.16}$$

to be sufficient for convergence. This formula suggests that the stepsize should be made smaller as the asynchronism measure $B$ increases. This is very intuitive: if processors make larger steps, then they should inform the others more often. In particular, suppose that we had the objective of keeping the quantity $|x_i(t) - x_i^j(t)|$ bounded. Assuming that $\|s(t)\|$ is bounded, Eq. (5.9) suggests that the product $\gamma B$ should be bounded, consistently with Eq. (5.16).

It should be stressed here that Eq. (5.16) is associated with a sufficient, not necessary, condition for convergence. Nevertheless, this equation has the right qualitative properties. It can be shown that if nothing is known about the optimization problem at hand, other than the constants $K_1$ and $K_3$, then in order to guarantee convergence, it is necessary to take $\gamma_0$ smaller than $c/B$, where $c$ is some constant, depending on $K_1$ and $K_3$ (Exercise 5.1).

A further issue of interest concerns the relation between $\gamma_0$ and the constants $K_1$ and $K_3$. We first consider the constant $K_3$ of Eq. (5.7). The quantity that matters in the algorithm is not $\gamma$ itself, but the product $\gamma s_i(t)$; it follows from Eq. (5.7), that only the product $\gamma K_3$ matters. This explains why $\gamma_0$ and $K_3$ are inversely proportional in Eq. (5.16).

The dependence on $K_1$ is more involved. Even for synchronous algorithms, if $K_3$ is kept constant, it is a necessary condition for convergence to pick $\gamma_0$ inversely proportional to $K_1$, consistently with Eq. (5.16). Equation (5.16), however, suggests another tradeoff: if $\gamma_0$ is held constant, then $B$ should be made smaller as $K_1$ increases. This is again natural: if $K_1$ is larger, a change in $x_i$ produces a larger change in $\nabla_j F$.

Therefore, if we want $\nabla_j F\big(x^j(t)\big) - \nabla_j F\big(x(t)\big)$ to stay bounded, a smaller change in $x_i(t)$ is tolerated before processor $j$ receives new information from processor $i$, and $B$ should be smaller.

Suppose now that there are only two processors and that $F(x) = F^1(x_1) + F^2(x_2)$; we thus have two decoupled optimization problems. Clearly, convergence is obtained even if the processors never communicate; in particular, $B$ can be arbitrarily large. Suppose that we add a coupling term $F^3(x_1, x_2)$ to the cost function. The presence of this term now requires that the two processors communicate to each other. Intuition suggests that the frequency of communication should depend primarily on the properties of the coupling term $F^3$. To formalize this effect, we refine Assumption 5.1(b) in a way that allows us to quantify the strength of coupling between different components. Assumption 5.4 that follows is a diagonal dominance assumption, similar to Assumptions 3.1 and 3.2, introduced in Section 6.3.

**Assumption 5.4.** There exists a constant $K_4$ such that $|\nabla_i F(x) - \nabla_i F(y)| \leq K_4 \|x - y\|$ for every $i$ and for all $x, y \in \Re^n$ satisfying $x_i = y_i$.

In the case of decoupled problems, we have $K_4 = 0$; the case where $K_4$ is small can be taken as the definition of a weakly coupled problem. Using Assumption 5.4, the inequalities in the proof of Prop. 5.1 can be somewhat improved. In particular, consider the term $\gamma K_1 \sum_{i=1}^n |s_i(t)| \cdot \|x(t) - x^i(t)\|$ arising in the right–hand side of inequality (5.8); if we impose the additional assumption that $\tau_i^i(t) = t$, we obtain $x_i(t) = x_i^i(t)$, and $K_1$ can be replaced by $K_4$. Tracing the steps in the proof, we are finally led to the following sufficient condition for convergence:

$$0 < \gamma < \gamma_0 = \frac{1}{K_3(K_1 + K_4 B + nK_4 B)}.$$

Thus, the asynchronism measure $B$ must be inversely proportional to the strength of coupling between different components, as quantified by the constant $K_4$. However, even this prescription on the allowed delay is too restrictive when $K_4$ becomes very small. To see this, assume that $F$ is twice continuously differentiable and that there exists some $\alpha > 0$ such that $\nabla_{ii}^2 F(x) \geq \alpha$ for every $x \in \Re^n$ and for every $i$. It is easily shown that if $K_4$ is sufficiently small, then the matrix $\nabla^2 F$ is diagonally dominant, the iteration $x := x - \gamma \nabla F(x)$ is a maximum norm contraction mapping for sufficiently small $\gamma$ (Prop. 1.11 in Section 3.1), the totally asynchronous gradient algorithm converges (Section 6.3), and no bound on $B$ is needed. Such a diagonal dominance condition cannot be exploited sufficiently well in the proof of Prop. 5.1 because this proof relies on cost reduction rather than reduction of the error $\|x(t) - x^*\|$, where $x^*$ is a minimizer of $F$.

Finally, let us remark that the dependence of $\gamma$ on $n$ in Eq. (5.16) is fairly conservative and should not be taken at face value.

### 7.5.3 Block–Iterative Algorithms

Proposition 5.1 can be extended to the case where each component $x_i$ is not one–dimensional, but rather a vector of dimension $n_i$. Thus, $x$ is a vector of dimension

$\sum_{i=1}^m n_i$, which has been decomposed into block–components $x_1, \ldots, x_m$, and $m$ is the number of processors. Accordingly, we let $s_i$ be a vector of dimension $n_i$. In this context, we can relax the requirement that each one–dimensional component of $x$ is updated in a descent direction. Rather, it is sufficient to require that each block–component be updated in a descent direction. In particular, we replace Assumption 5.2 with the following.

**Assumption 5.5.** There exist positive constants $K_2$ and $K_3$ such that:

**(a)** (*Block–Descent*) There holds $s_i(t)'\nabla_i F\big(x^i(t)\big) \leq -\|s_i(t)\|^2/K_3$ for all $i$ and all $t \in T^i$.

**(b)** There holds $\|s_i(t)\| \geq K_2\big\|\nabla_i F\big(x^i(t)\big)\big\|$ for all $i$ and all $t \in T^i$.

In the above assumption, the notation $\nabla_i F$ stands for the vector of dimension $n_i$ with the partial derivatives of $F$ with respect to the (scalar) components of $x$ corresponding to the subvector $x_i$. We have the following extension of Prop. 5.1.

**Proposition 5.2.** Under Assumptions 5.1, 5.3, and 5.5, there exists some $\gamma_0 > 0$ (depending on $n$, $B$, $K_1$, and $K_3$) such that, if $0 < \gamma < \gamma_0$, then $\lim_{t\to\infty} \nabla F\big(x(t)\big) = 0$.

The proof of Prop. 5.2 is almost the same as the proof of Prop. 5.1 and is left as an exercise.

### 7.5.4 Gradient Projection Algorithms

We consider a partially asynchronous implementation of the gradient projection algorithm for constrained optimization. For simplicity, we only consider the case of identity scaling, but the discussion applies to more general scaling rules as well. Let $X \subset \Re^n$ be nonempty, closed, and convex, and let $F : \Re^n \mapsto \Re$ be a convex continuously differentiable cost function. We assume that $X$ is a Cartesian product $X = \prod_{i=1}^m X_i$ of lower dimensional sets, where $X_i \subset \Re^{n_i}$ and $\sum_{i=1}^m n_i = n$. The $i$th processor updates the $i$th block–component $x_i$ according to

$$x_i(t+1) = \Big[x_i(t) - \gamma\nabla_i F\big(x^i(t)\big)\Big]^+, \qquad t \in T^i, \tag{5.17}$$

where $[\cdot]^+$ denotes projection on the set $X_i$ and where the vector $x^i(t)$ is defined as in Eq. (5.5). Naturally, we let $x_i(t+1) = x_i(t)$ if $t \notin T^i$. We represent this algorithm in the form of Eq. (5.3) by defining $s_i(t) = 0$, if $t \notin T^i$, and

$$s_i(t) = \frac{1}{\gamma}\big(x_i(t+1) - x_i(t)\big) = \frac{1}{\gamma}\Big(\Big[x_i(t) - \gamma\nabla_i F\big(x^i(t)\big)\Big]^+ - x_i(t)\Big), \qquad t \in T^i.$$

**Lemma 5.1.** For any $i$ and $t$, we have

$$s_i(t)'\nabla_i F\big(x^i(t)\big) \leq -\|s_i(t)\|^2. \tag{5.18}$$

***Proof.*** If $t \notin T^i$, then inequality (5.18) is trivially true since both sides are zero. If $t \in T^i$, the Projection Theorem (Prop. 3.2 in Section 3.3) yields

$$\Big(\Big[x_i(t)-\gamma\nabla_i F\big(x^i(t)\big)\Big]^+ - x_i(t)\Big)'\Big(\Big[x_i(t)-\gamma\nabla_i F\big(x^i(t)\big)\Big]^+ - x_i(t)+\gamma\nabla_i F\big(x^i(t)\big)\Big) \leq 0.$$

Equivalently,

$$\gamma s_i(t)'\Big(\gamma s_i(t) + \gamma\nabla_i F\big(x_i(t)\big)\Big) \leq 0, \qquad t \in T^i,$$

from which inequality (5.18) follows. **Q.E.D.**

Lemma 5.1 establishes a block–descent property for the gradient projection algorithm [cf. Assumption 5.5(a)]. This property can be exploited to obtain the following result.

**Proposition 5.3.** Let $X \subset \Re^n$ be as before. Suppose that $F : \Re^n \mapsto \Re$ is convex, nonnegative, and its gradient satisfies the Lipschitz continuity condition (5.1). Let Assumption 5.3 (partial asynchronism) hold. Then there exists some $\gamma_0 > 0$ such that if $0 < \gamma < \gamma_0$, then any limit point $x^*$ of the sequence $\{x(t)\}$ generated by the partially asynchronous gradient projection algorithm minimizes $F$ over $X$.

***Proof.*** We follow the proof of Prop. 5.1. The only differences are the following. The term $s_i(t)\nabla_i F\big(x^i(t)\big)$ is replaced by the inner product $s_i(t)'\nabla_i F\big(x^i(t)\big)$ and the term $|s_i(t)|$ is replaced by $\|s_i(t)\|$. Also, $K_3$ is replaced by 1. All the steps up to Eq. (5.15) remain valid, and we obtain $\lim_{t\to\infty} s_i(t) = 0$ for each $i$. Let $x^*$ be a limit point of $x(t)$, let $\{t_k\}$ be a sequence such that $\lim_{k\to\infty} x(t_k) = x^*$, and let $\tau_k$ be such that $|t_k - \tau_k| \leq B$ and $\tau_k \in T^i$. Then, Eqs. (5.14) and (5.15) imply that $x(\tau_k)$ and $x^i(\tau_k)$ converge to $x^*$. We then have

$$\Big[x_i^*-\gamma\nabla_i F(x^*)\Big]^+ - x_i^* = \lim_{k\to\infty}\Big(\Big[x_i(\tau_k)-\gamma\nabla_i F\big(x^i(\tau_k)\big)\Big]^+ - x_i(\tau_k)\Big) = \lim_{k\to\infty}\gamma s_i(\tau_k) = 0.$$

Since this is true for each $i$, it follows that $x^*$ minimizes $F$ (Prop. 3.3 in Section 3.3). **Q.E.D.**

## EXERCISES

**5.1.** Consider Example 1.3 of Section 7.1 for which the totally asynchronous gradient algorithm diverges. Show that a necessary condition for convergence of the partially asynchronous gradient algorithm is $0 < \gamma \leq c/B$, where $c$ is a suitable constant independent of $B$ or $\gamma$.

**5.2.** Identify the modifications needed in the proof of Prop. 5.1 in order to prove Prop. 5.2.

## 7.6 DISTRIBUTED ASYNCHRONOUS ROUTING IN DATA NETWORKS

We consider here and prove convergence of a distributed asynchronous implementation of the gradient projection method for solving an optimal routing problem in a data communication network, assuming that the routing problem has been formulated as a nonlinear multicommodity network flow problem, as in Section 5.6. Our model incorporates a number of aspects of actual data networks, such as asynchronism and the transients that occur when a routing strategy is changed. Our model can also serve as a prototype for the analysis of other asynchronous iterative models involving imperfectly known dependencies on several past iterates.

Throughout this section, $\|\cdot\|$ stands for the Euclidean vector norm $\|\cdot\|_2$.

### 7.6.1 Problem Definition

The reader may wish to review Section 5.6, where the routing problem has been defined. We repeat here the key features and notation. We are given a network (directed graph), a set $W$ of origin–destination (OD) pairs, and for each OD pair $w \in W$, a set of simple positive paths $P_w$ from the origin to the destination. Let $r_w > 0$ be the flow that has to be routed from the origin to the destination of OD pair $w$, and let $x_p$ denote the flow through a particular path $p \in P_w$. These path flows satisfy the constraints

$$\sum_{p \in P_w} x_p = r_w, \qquad \forall w \in W, \tag{6.1}$$

$$x_p \geq 0, \qquad \forall p \in P_w, \quad \forall w \in W. \tag{6.2}$$

Let $x_w$ be the vector consisting of the path flows $x_p$, $p \in P_w$, and let $x$ be the vector of all path flows; the latter vector is formed by arranging all the vectors $x_w$ in a single vector.

For any directed arc $(i, j)$ in the network, $F_{ij}$, the flow through that arc, is given by

$$F_{ij} = \sum_{p \in P_{ij}} x_p, \tag{6.3}$$

where $P_{ij}$ is the set of all paths traversing arc $(i, j)$. We wish to minimize a cost function of the form

$$\sum_{(i,j)} D_{ij}(F_{ij}),$$

where the summation runs over all arcs $(i, j)$ in the network. From Eq. (6.3), this cost function can be expressed in terms of the vector $x$ of path flows to yield a cost function $D(x)$ defined by

$$D(x) = \sum_{(i,j)} D_{ij}\left(\sum_{p \in P_{ij}} x_p\right). \tag{6.4}$$

We make the following assumption.

**Assumption 6.1.** For every arc $(i, j)$, the function $D_{ij} : \Re \mapsto \Re$ is convex, twice continuously differentiable, and its second derivative is bounded.

Assumption 6.1 implies that $D$ is also convex, twice continuously differentiable, and its Hessian matrix is bounded. In particular, the boundedness of the Hessian matrix implies that there exists a constant $K$ such that

$$\|\nabla D(x) - \nabla D(y)\| \leq K\|x - y\|, \qquad \forall x, y. \tag{6.5}$$

We are faced here with the problem of minimizing a twice differentiable convex function over the closed convex constraint set defined by Eqs. (6.1) and (6.2), and any variant of the gradient projection algorithm can be used. However, when such an algorithm is implemented in real time in an operating data network, the network is likely to behave quite differently from the mathematical description of the algorithm. This is due to several reasons:

**(a)** Information is inexact: the gradient projection method requires the evaluation of the derivatives of the cost function with respect to the path flows. This requires knowledge of the arc flows through the arcs of interest. Such information is obtained in practice by having the end nodes of each arc measure the average flow through the arc, over a sufficiently long period of time, and then transmit this value to all other nodes that need this value. Due to communication delays, when this information reaches its destination, it can be outdated.

**(b)** Updates occur asynchronously: the reason is that it is fairly difficult to synchronize a large and geographically distributed data network.

**(c)** The above problems notwithstanding, even if a node in the network performs a computation and decides to update the path flows under its control, it may be unable to do so instantaneously. For example, if the traffic through the network consists of sessions between users (e.g., telephone conversations), it may be difficult to reroute ongoing sessions; it may be preferable to reroute the traffic by waiting for the current sessions to terminate and assigning different routes to newly generated sessions.

The mathematical model that follows takes into account all of the above considerations. The only missing element from a fully realistic model are the stochastic fluctuations in the amount of traffic that has to be routed. Still, the analysis to follow is asymptotically exact, if one assumes that there is a very large number of users with very small communication rate, in which case, the laws of large numbers take over and eliminate the stochastic fluctuations.

### 7.6.2 The Algorithm and its Convergence

Let $t$ be a discrete (integer) time variable used to index the events of interest (routing updates). If $p \in P_w$, let $x_p(t)$ be the amount of traffic for OD pair $w$ routed through path $p$ at time $t$. We define vectors $x_w(t)$ and $x(t)$ analogously. That is, the components of $x_w(t)$ are the variables $x_p(t)$, $p \in P_w$, and $x(t)$ consists of all the vectors $x_w(t)$ arranged in a single vector. Let $F_{ij}(t)$ be the flow through arc $(i,j)$ at time $t$. In particular, Eq. (6.3) holds for all times $t$. We assume that a separate processor $w$ is assigned the responsibility of updating $x_w$. Let $T^w$ be the set of times that such an update is performed. The algorithm is assumed to start at some negative time and we assume that by time 0, each processor $w$ has performed at least one update. That is, for each $w$, there exists some $t \in T^w$ such that $t \leq 0$. We assume that processor $w$ knows $x_w(t)$ and $r_w$ exactly. This is reasonable, for example, if this processor is located at the origin node of the OD pair $w$. However, this processor does not know exactly the values of the partial derivatives $(\partial D/\partial x_p)\big(x(t)\big)$, for $p \in P_w$, but has access to estimates denoted by $\lambda_p(t)$.

We now describe how each $\lambda_p(t)$ is formed. We assume that for every arc $(i,j)$, node $i$ estimates the amount of traffic through that arc by averaging a few recent values of measured traffic. Accordingly, at each time $t$, node $i$ has available an estimate

$$\tilde{F}_{ij}(t) = \sum_{\tau=t-Q}^{t} c_{ij}(t,\tau) F_{ij}(\tau). \tag{6.6}$$

Here $c_{ij}(t,\tau)$ are (generally unknown) nonnegative scalars satisfying

$$\sum_{\tau=t-Q}^{t} c_{ij}(t,\tau) = 1, \qquad \forall t.$$

The constant $Q$ is a bound on the time over which measurements are being averaged. These estimates are broadcast from time to time (asynchronously and possibly with some variable communication delay). We assume that the times between consecutive broadcasts and the communication delays are bounded, and, consequently, the information available to any other processor can be outdated by at most $R$ time units, where $R$ is some constant. That is, at any time $t$, each processor $w$ knows the value of an estimate $\tilde{F}_{ij}(\sigma)$ for some $\sigma$ satisfying $t - R \leq \sigma \leq t$. This, together with Eq. (6.6) implies that at each time $t$, each processor $w$ has access to an estimate $\hat{F}_{ij,w}(t)$ satisfying

$$\hat{F}_{ij,w}(t) = \sum_{\tau=t-C}^{t} d_{ij,w}(t,\tau) F_{ij}(\tau), \tag{6.7}$$

where $C = R + Q$, and $d_{ij,w}(t,\tau)$ are (generally unknown) nonnegative coefficients satisfying

$$\sum_{\tau=t-C}^{t} d_{ij,w}(t,\tau) = 1, \qquad \forall t. \tag{6.8}$$

We now notice that for every path $p$, we have [cf. Eq. (6.4)]

$$\frac{\partial D}{\partial x_p}\big(x(t)\big) = \sum_{\text{all arcs } (i,j) \text{ on path } p} D'_{ij}\big(F_{ij}(t)\big), \qquad \forall t \in T^w,$$

where $D'_{ij}$ denotes the derivative of the function $D_{ij}$. As the exact value of $F_{ij}(t)$ is unavailable, we assume that processor $w$ forms an estimate $\lambda_p(t)$ by letting

$$\lambda_p(t) = \sum_{\text{all arcs } (i,j) \text{ on path } p} D'_{ij}\big(\hat{F}_{ij,w}(t)\big), \qquad \forall t \in T^w. \tag{6.9}$$

We then let $\lambda_w(t)$ be a vector of the same dimension as $x_w$, whose coordinates are the estimates $\lambda_p(t)$ corresponding to the paths $p$ belonging to $P_w$.

Using the estimate $\lambda_w(t)$, processor $w$ can compute a vector of desired path flows for OD pair $w$ denoted by $\bar{x}_w(t)$, whose components are $\bar{x}_p(t)$, $p \in P_w$. The actual flows, however, do not settle instantly to their desired values. Rather, we assume that they approach the desired values geometrically. In particular, we assume that there exist scalars $\alpha > 0$, $a_p(t)$, such that

$$a_p(t) \geq \alpha, \qquad \forall p, t, \tag{6.10}$$

and such that

$$x_p(t+1) = a_p(t)\bar{x}_p(t) + \big(1 - a_p(t)\big)x_p(t), \qquad \forall p, t. \tag{6.11}$$

Thus, the flow $x_p(t)$ moves toward the desired value by a nonnegligible factor, during each time unit. The validity of this assumption depends on the particular method that the processors use for rerouting the traffic toward a preferred set of routes; it is satisfied by several rerouting methods (Exercise 6.1). In addition to Eq. (6.11), it is, of course, assumed that the equation $\sum_{p \in P_w} x_p(t) = r_w$ is valid for all $t$ and all $w$.

It remains to describe the heart of the algorithm, which is the method for computing the desired flows. If $t \notin T^w$, we simply let $\bar{x}_w(t) = \bar{x}_w(t-1)$. If $t \in T^w$, then processor $w$ determines $\bar{x}_w(t)$ by performing an iteration of a version of the gradient projection method [see Eqs. (6.21)–(6.23) of Section 5.6]. In more detail, processor $w$ finds a MFDL (minimum first derivative length) path $p_w(t)$, that is, a path with the smallest associated value of $\lambda_p(t)$. Let

$$\tilde{\lambda}_w(t) = \min_{p \in P_w} \lambda_p(t) = \lambda_{p_w(t)}(t).$$

For any $p \in P_w$, $p \neq p_w(t)$, let $H_p(t)$ be a scaling factor. Usually, $H_p(t)$ is an estimate of the second derivative length of path $p$ [see Eq. (6.23), Section 5.6], but for our purposes, we only need to assume that there exist constants $h$ and $H$ such that

$$0 < h \leq H_p(t) \leq H, \qquad \forall t, p. \tag{6.12}$$

(If $H_p(t)$ is the second derivative length [as in Eq. (6.23) of Section 5.6], and if the second derivative $D''_{ij}$ is bounded above and below by positive numbers for each arc $(i,j)$, then condition (6.12) will be satisfied.) For any $p \in P_w$, $p \neq p_w(t)$, $\tilde{x}_p(t)$ is determined by [cf. Eq. (6.21), Section 5.6]

$$\tilde{x}_p(t) = \max\Big\{0, x_p(t) - \frac{\gamma}{H_p(t)}\big(\lambda_p(t) - \tilde{\lambda}_w(t)\big)\Big\}, \qquad t \in T^w. \tag{6.13}$$

For $p = p_w(t)$, we let

$$\tilde{x}_{p_w(t)}(t) = r_w - \sum_{p \in P_w,\ p \neq p_w(t)} \tilde{x}_p(t), \qquad t \in T^w. \tag{6.14}$$

The description of the algorithm is now complete. The main equations are Eq. (6.3), describing the dependence of the arc flows on the path flows, Eq. (6.9), describing the dependence of the estimates $\lambda_p(t)$ on the arc flows, Eqs. (6.13) and (6.14), describing the dependence of the desired flows on $\lambda_p(t)$, and Eq. (6.11), which closes the loop by describing the dependence of the actual path flows on the desired ones.

The main ideas in the proof of the following result are similar with the convergence proof for partially asynchronous gradient–like algorithms presented in the preceding section (Prop. 5.1). We show again that the updates are in a descent direction. Furthermore, the errors caused by asynchronism are of second order in $\gamma$ and are inconsequential when $\gamma$ is small. Some additional complications are caused by the transients when actual flows move towards their desired values. In particular, we have to ensure that these transients do not destroy the descent properties of the algorithm.

**Proposition 6.1.** Assume that the difference between consecutive elements of $T^w$ is bounded by some constant $B$ for each OD pair $w$. Then there exists some $\gamma_0 > 0$ such that if $0 < \gamma < \gamma_0$, then $D\big(x(t)\big)$ converges to the minimum of $D(x)$ over the set of feasible path flow vectors $x$, and every limit point of the sequence $\{x(t)\}$ minimizes $D$.

***Proof.*** Throughout the proof, $K_1, K_2, \ldots$, will be constants that do not depend on $t$ or $\gamma$. Notice that for any $p \in P_w$, $\lambda_p(t)$ is defined and used only for $t \in T^w$ [Eqs. (6.9) and (6.13)]. To simplify notation, we also define $\lambda_p(t)$ for $t \notin T^w$, $t \geq 0$, by $\lambda_p(t) = \lambda_p(t_0)$, where $t_0$ is the largest element of $T^w$ satisfying $t_0 \leq t$. (Such a $t_0$ is well defined, because we have assumed that each set $T^w$ contains a nonpositive element.) Similarly, we use the convention $\tilde{\lambda}_w(t) = \tilde{\lambda}_w(t_0)$.

For any $t$ and $p \in P_w$, we define variables $s_p(t)$ and $\bar{s}_p(t)$ by

$$s_p(t) = x_p(t+1) - x_p(t), \tag{6.15}$$

$$\bar{s}_p(t) = \begin{cases} \bar{x}_p(t) - x_p(t), & t \in T^w, \\ 0, & t \notin T^w. \end{cases} \tag{6.16}$$

We define vectors $s_w(t)$, $\bar{s}_w(t)$, $s(t)$, and $\bar{s}(t)$ analogously. Our first lemma relates $s(t)$ to $\bar{s}(t)$.

**Lemma 6.1.**

**(a)** There holds

$$s_p(t) = a_p(t)\bar{s}_p(t), \qquad \forall p \in P_w, \ \forall t \in T^w, \ \forall w. \tag{6.17}$$

**(b)** Given some $w$ and $t \geq 0$, let $t_0$ be the largest element of $T^w$ satisfying $t_0 \leq t$. Then for $p \in P_w$,

$$\operatorname{sign}\big(s_p(t)\big) = \operatorname{sign}\big(\bar{s}_p(t_0)\big), \tag{6.18}$$

$$|s_p(t)| \leq |\bar{s}_p(t_0)|, \tag{6.19}$$

where we are using the convention $\operatorname{sign}(0) = 0$.

**(c)** There exist constants $K_1$ and $K_1'$ such that

$$\sum_{\tau=0}^{t} |s_p(\tau)|^2 \leq K_1 \sum_{\tau=0}^{t} |\bar{s}_p(\tau)|^2 + K_1', \qquad \forall t \geq 0, \ \forall p \in P_w, \ \forall w. \tag{6.20}$$

***Proof.***

**(a)** Immediate from Eq. (6.11) and the definitions (6.15) and (6.16).

**(b)** Let $t_1$ be the first element of $T^w$ larger than $t_0$. Suppose that $\bar{s}_p(t_0) \geq 0$. An easy induction shows that

$$x_p(t_0) \leq x_p(t) \leq x_p(t+1) \leq \bar{x}_p(t_0), \qquad \forall t \in [t_0, t_1),$$

and the result follows for this case. The argument for the case $\bar{s}_p(t_0) \leq 0$ is identical.

**(c)** Let $p \in P_w$. Part (b) yields, for any successive nonnegative elements $t_0$ and $t_1$ of $T^w$,

$$\sum_{\tau=t_0}^{t_1-1} |s_p(\tau)|^2 \leq (t_1 - t_0)|\bar{s}_p(t_0)|^2 \leq B|\bar{s}_p(t_0)|^2. \tag{6.21}$$

Let $t_w$ ($t'_w$, respectively) be the smallest element of $T^w$ such that $t_w \geq 0$ ($t'_w > t$, respectively). We sum inequality (6.21) over all $t_0 \in T^w$ such that $t_w \leq t_0 < t'_w$, and use the property $\bar{s}_p(\tau) = 0$ for $\tau \notin T^w$ to obtain

$$\sum_{\tau=t_w}^{t'_w-1} |s_p(\tau)|^2 \leq B \sum_{\{\tau \in T^w | t_w \leq \tau < t'_w\}} |\bar{s}_p(\tau)|^2 = B \sum_{\tau=0}^{t} |\bar{s}_p(\tau)|^2.$$

Since $t'_w > t$, we obtain

$$\sum_{\tau=0}^{t} |s_p(\tau)|^2 \leq \sum_{\tau=0}^{t_w-1} |s_p(\tau)|^2 + \sum_{\tau=t_w}^{t'_w-1} |s_p(\tau)|^2 \leq K'_1 + B \sum_{\tau=0}^{t} |\bar{s}_p(\tau)|^2.$$

Notice that $t_w \leq B$ and, therefore, $K'_1$ can be chosen equal to $B$ times the maximum possible value of $|s_p(t)|$, which is finite because $s_p(t)$ belongs to a bounded set. **Q.E.D.**

We next show that the steps taken by the algorithm possess a certain descent property (cf. Lemma 5.1 in Section 7.5).

**Lemma 6.2.** There exists a constant $K_2 > 0$ such that

$$\lambda_w(t)' s_w(t) \leq -\frac{K_2}{\gamma} \|\bar{s}_w(t)\|^2, \qquad \forall t \geq 0, w. \tag{6.22}$$

***Proof.*** Notice that

$$\sum_{p \in P_w} s_p(t) = \sum_{p \in P_w} x_p(t+1) - \sum_{p \in P_w} x_p(t) = r_w - r_w = 0, \qquad \forall t, w. \tag{6.23}$$

Consequently,

$$\lambda_w(t)' s_w(t) = \sum_{p \in P_w} \lambda_p(t) s_p(t) = \sum_{p \in P_w} s_p(t) \big(\lambda_p(t) - \tilde{\lambda}_w(t)\big). \tag{6.24}$$

We first consider the case $t \notin T^w$, in which the right–hand side of inequality (6.22) is zero. Let $t_0$ be again the largest element of $T^w$ satisfying $t_0 \leq t$, as in Lemma 6.1(b). We have, by definition, $\lambda_p(t) - \tilde{\lambda}_w(t) = \lambda_p(t_0) - \tilde{\lambda}_w(t_0) \geq 0$ for every $p \in P_w$. Furthermore, Eq. (6.13) shows that $\bar{s}_p(t_0) \leq 0$ for every $p \neq p_w(t)$. Using Eq. (6.18) we obtain $s_p(t) \leq 0$ for every $p \neq p_w(t)$, $p \in P_w$. Therefore, each summand in Eq. (6.24), with $p \neq p_w(t)$, is less than or equal to zero. The summand corresponding to $p_w(t)$ is equal to zero, because $\tilde{\lambda}_w(t) = \lambda_{p_w(t)}(t)$, by definition. This proves inequality (6.22) for the case $t \notin T^w$.

We now assume that $t \in T^w$. Equation (6.24) becomes

$$\begin{aligned}\lambda_w(t)'s_w(t) &= \sum_{p\in P_w} a_p(t)\bar{s}_p(t)\big(\lambda_p(t)-\tilde{\lambda}_w(t)\big)\\ &= -\sum_{p\in P_w} a_p(t)|\bar{s}_p(t)|\cdot|\lambda_p(t)-\tilde{\lambda}_w(t)|\\ &\le -\alpha\sum_{p\in P_w}|\bar{s}_p(t)|\cdot|\lambda_p(t)-\tilde{\lambda}_w(t)|\\ &\le -\frac{\alpha h}{\gamma}\sum_{p\in P_w,\ p\neq p_w(t)}|\bar{s}_p(t)|^2.\end{aligned} \tag{6.25}$$

[Here the first equality follows from Eq. (6.17); the second follows from the same reasoning as in the preceding paragraph; the next inequality follows from inequality (6.10); and the last inequality follows because Eqs. (6.12) and (6.13) yield $|\bar{s}_p(t)| \le (\gamma/h)|\lambda_p(t)-\tilde{\lambda}_w(t)|$ for $p \neq p_w(t)$.] Equation (6.14) leads to

$$\bar{s}_{p_w(t)}(t) = -\sum_{p\in P_w,\ p\neq p_w(t)}\bar{s}_p(t) \tag{6.26}$$

and by squaring both sides of Eq. (6.26), we obtain

$$|\bar{s}_{p_w(t)}(t)|^2 \le K_3\sum_{p\in P_w,\ p\neq p_w(t)}|\bar{s}_p(t)|^2,$$

where $K_3$ is an upper bound on the cardinality of each $P_w$. Therefore,

$$\sum_{p\in P_w,\ p\neq p_w(t)}|\bar{s}_p(t)|^2 \ge \frac{1}{2}\sum_{p\in P_w,\ p\neq p_w(t)}|\bar{s}_p(t)|^2 + \frac{1}{2K_3}|\bar{s}_{p_w(t)}(t)|^2 \ge \frac{1}{2K_3}\|\bar{s}_w(t)\|^2. \tag{6.27}$$

Using Eqs. (6.25) and (6.27), we obtain

$$\lambda_w(t)'s_w(t) \le -\frac{\alpha h}{2\gamma K_3}\|\bar{s}_w(t)\|^2,$$

which proves inequality (6.22) for the case $t \in T^w$ with $K_2 = \alpha h/2\gamma K_3$. **Q.E.D.**

The next lemma bounds the error in the estimated gradient, caused by asynchronism.

**Lemma 6.3.** There exists some constant $K_4$ such that

$$\big\|\lambda(t)-\nabla D\big(x(t)\big)\big\| \le K_4\sum_{\tau=t-B-C}^{t}\|s(\tau)\|, \qquad \forall t \ge 0, \tag{6.28}$$

where $C$ is the constant in Eq. (6.7).

***Proof.*** Let us fix some $p \in P_w$ and some $t \geq 0$. Let $t_0$ be the largest element of $T^w$ satisfying $t_0 \leq t$. Then, using Eq. (6.9),

$$\left|\lambda_p(t) - \frac{\partial D}{\partial x_p}\big(x(t)\big)\right| = \left|\lambda_p(t_0) - \frac{\partial D}{\partial x_p}\big(x(t)\big)\right| \leq \sum_{(i,j)} \left|D'_{ij}\big(\hat{F}_{ij,w}(t_0)\big) - D'_{ij}\big(F_{ij}(t)\big)\right|, \tag{6.29}$$

where the last summation is over all arcs traversed by path $p$. Let $K_5$ be the maximum of the number of arcs in any given path. Let $K_6$ be a bound on the second derivatives $D''_{ij}$. Then inequality (6.29) becomes

$$\begin{aligned}
\left|\lambda_p(t) - \frac{\partial D}{\partial x_p}\big(x(t)\big)\right| &\leq K_5 K_6 \max_{(i,j)} \left|\hat{F}_{ij,w}(t_0) - F_{ij}(t)\right| \\
&\leq K_5 K_6 \max_{(i,j)} \sum_{\tau=t_0-C}^{t_0} d_{ij,w}(t_0,\tau)\left|F_{ij}(\tau) - F_{ij}(t)\right| \\
&\leq K_5 K_6 \max_{(i,j)} \max_{t_0-C\leq\tau\leq t_0} \left|F_{ij}(\tau) - F_{ij}(t)\right| \\
&\leq K_5 K_6 \max_{(i,j)} \max_{t-B-C\leq\tau\leq t} \left|F_{ij}(\tau) - F_{ij}(t)\right| \\
&\leq K_5 K_6 K_7 \max_{t-B-C\leq\tau\leq t} \left\|x(\tau) - x(t)\right\| \\
&\leq K_8 \sum_{\tau=t-B-C}^{t} \|s(\tau)\|.
\end{aligned} \tag{6.30}$$

Here $K_8 = K_5 K_6 K_7$. For the constant $K_7$, we can use the induced norm of the linear mapping in Eq. (6.3), which determines the arc flows $F_{ij}$ in terms of the path flows $x_p$. In the previous inequalities, we have also made use of Eqs. (6.7) and (6.8).

The desired result follows from inequality (6.30) and the triangle inequality. **Q.E.D.**

We can now carry out the main part of the proof, which follows the same steps as the proof of Prop. 5.1 in the preceding section. Using the descent lemma (Prop. A.32 in Appendix A) and Lemmas 6.2 and 6.3, we obtain

$$\begin{aligned}
D\big(x(t+1)\big) &\leq D\big(x(t)\big) + s(t)'\nabla D\big(x(t)\big) + \frac{K}{2}\|s(t)\|^2 \\
&\leq D\big(x(t)\big) + s(t)'\lambda(t) + \left\|\lambda(t) - \nabla D\big(x(t)\big)\right\| \cdot \left\|s(t)\right\| + \frac{K}{2}\left\|s(t)\right\|^2 \\
&\leq D\big(x(t)\big) - \frac{K_2}{\gamma}\left\|\bar{s}(t)\right\|^2 + K_4 \sum_{\tau=t-B-C}^{t} \left\|s(\tau)\right\| \cdot \left\|s(t)\right\| + \frac{K}{2}\left\|s(t)\right\|^2
\end{aligned}$$

$$\leq D\big(x(t)\big) - \frac{K_2}{\gamma}\big\|\bar{s}(t)\big\|^2 + K_4 \sum_{\tau=t-B-C}^{t} \Big(\big\|s(\tau)\big\|^2 + \big\|s(t)\big\|^2\Big) + \frac{K}{2}\big\|s(t)\big\|^2$$

$$\leq D\big(x(t)\big) - \frac{K_2}{\gamma}\big\|\bar{s}(t)\big\|^2 + K_9 \sum_{\tau=t-B-C}^{t} \big\|s(\tau)\big\|^2, \tag{6.31}$$

where $K_9 = K_4(B + C + 1) + K/2$. By adding inequality (6.31) for different values of $t$, and rearranging terms, we obtain

$$D\big(x(t+1)\big) \leq D\big(x(0)\big) - \frac{K_2}{\gamma}\sum_{\tau=0}^{t}\big\|\bar{s}(t)\big\|^2 + K_9(B + C + 1)\sum_{\tau=-B-C}^{t}\big\|s(\tau)\big\|^2$$

$$\leq D\big(x(0)\big) - \Big(\frac{K_2}{\gamma} - K_1K_9(B + C + 1)\Big)\sum_{\tau=0}^{t}\big\|\bar{s}(\tau)\big\|^2$$

$$+ K_1'K_9(B + C + 1) + \sum_{\tau=-B-C}^{-1}\big\|s(\tau)\big\|^2,$$

where in the last step we made use of inequality (6.20). Let $\gamma$ be small enough so that $K_2 > \gamma K_1 K_9(B + C + 1)$. Notice that $D\big(x(t+1)\big)$ is bounded from below [because $D$ is continuous and $x(t+1)$ belongs to a compact set]. Since the sum $\sum_{\tau=-B-C}^{-1}\big\|s(\tau)\big\|^2$ is finite, we conclude that the series $\sum_{\tau=0}^{t}\|\bar{s}(\tau)\|^2$ is bounded. Therefore,

$$\lim_{t\to\infty}\|\bar{s}(t)\| = 0. \tag{6.32}$$

Using inequality (6.19), we obtain

$$\lim_{t\to\infty}\|s(t)\| = 0. \tag{6.33}$$

Then Lemma 6.3 yields

$$\lim_{t\to\infty}\Big(\lambda(t) - \nabla D\big(x(t)\big)\Big) = 0. \tag{6.34}$$

Let $x^*$ be a limit point of the sequence $\{x(t)\}$. Such a limit point exists because $x(t)$ is constrained to lie in a compact set. Let us fix some $w \in W$. Since $x(t+1) - x(t) = s(t)$ converges to zero and since the difference between consecutive elements of $T^w$ is bounded, it follows that there exists a sequence of elements of $T^w$ along which $x(t)$ converges to $x^*$. Let $\{t_k\}$ be a subsequence of that sequence such that $p_w(t_k)$ is the same and equal to some $p^* \in P_w$ for all $t_k$. Let us also fix some $p \in P_w$ for which $x_p^* > 0$. Since $\bar{s}_p(t)$ converges to zero, the update equation (6.13), implies that

$$\lim_{k\to\infty}\big(\lambda_p(t_k) - \lambda_{p^*}(t_k)\big) = 0.$$

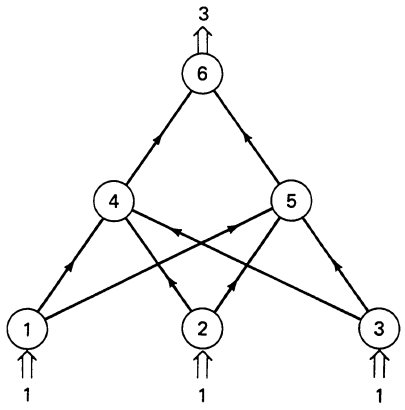



**Figure 7.6.1** The routing problem of Example 6.1. If the communication delays are excessively large, the flows can oscillate between the two available paths for each OD pair.

By the definition of $p^*$, $\lambda_{p^*}(t_k) \leq \lambda_{p'}(t_k)$ for every $k$ and for every $p' \in P_w$. Thus,

$$\limsup_{k\to\infty}\bigl(\lambda_p(t_k) - \lambda_{p'}(t_k)\bigr) \leq 0, \qquad \forall p' \in P_w.$$

Using Eq. (6.34), we obtain

$$\limsup_{k\to\infty}\left[\frac{\partial D}{\partial x_p}\bigl(x(t_k)\bigr) - \frac{\partial D}{\partial x_{p'}}\bigl(x(t_k)\bigr)\right] \leq 0, \qquad \forall p' \in P_w.$$

Since $x(t_k)$ converges to $x^*$, we use the continuity of $\partial D/\partial x_p$ to conclude that

$$\frac{\partial D}{\partial x_p}(x^*) \leq \frac{\partial D}{\partial x_{p'}}(x^*), \qquad \forall p' \in P_w.$$

This inequality has been shown to be true for every $w$ and for every $p \in P_w$ such that $x_p^* > 0$. But these are precisely the sufficient conditions for optimality of Section 5.6 [Eq. (6.11) in Section 5.6]. Therefore, $x^*$ is an optimal vector of path flows.

Consider now any sequence $\{t_k\}$ such that $D\bigl(x(t_k)\bigr)$ converges. By restricting to a subsequence, we can assume that $x(t_k)$ also converges to some $x^*$. As shown already, $x^*$ is an optimal vector and, therefore, $D\bigl(x(t_k)\bigr)$ converges to the optimal value of $D$, denoted by $D^*$. The sequence $\bigl\{D\bigl(x(t)\bigr)\bigr\}$ is bounded and we have just shown that all of its limit points are equal to $D^*$. Thus, $D\bigl(x(t)\bigr)$ converges to $D^*$, which completes the proof of the proposition. **Q.E.D.**

### 7.6.3 Discussion

The proof of Prop. 6.1 made essential use of the assumption that the information available to the processors cannot be arbitrarily out of date (see Lemma 6.3, for example). In the absence of such an assumption, the flows through the network can oscillate back and forth to what the nodes incorrectly perceive as underutilized paths and the algorithm does not converge. A simple example follows.

**Example 6.1.**

Consider the network of Fig. 7.6.1. There are three origin nodes (nodes 1, 2, and 3), whose input arrival rates are all 1, and a single destination node (node 6). For each OD pair, there are two paths. For each origin node $i$, let $x_i$ denote the flow routed through the path containing node 4. Let $D_{ij}(F_{ij}) = F_{ij}^2$ for $(i,j) = (4,6)$ or $(5,6)$, and $D_{ij}(F_{ij}) = 0$, for all other arcs. In terms of the variables $x_1$, $x_2$, and $x_3$, the cost becomes

$$D(x_1, x_2, x_3) = (x_1 + x_2 + x_3)^2 + (3 - x_1 - x_2 - x_3)^2.$$

We assume that the settling time is zero, so that we need not distinguish between actual and desired flows, and that each node $i$, $i = 1, 2, 3$, knows $x_i$ exactly and is able to transmit it instantaneously to the remaining nodes. Suppose that initially $x_1 = x_2 = x_3 = 1$ and that each node executes a large number of gradient projection iterations with a small stepsize before communicating the current value of $x_i$ to the other nodes. Then, effectively, each node $i$ solves the problem

$$\min_{0 \le x_i \le 1} \{(x_i + 2)^2 + (1 - x_i)^2\},$$

thereby obtaining the value $x_i = 0$. At that point, the processors broadcast their current values of $x_i$. If that sequence of events is repeated, a symmetrical argument shows that each $x_i$ will become again equal to 1. So, $(x_1, x_2, x_3)$ oscillates between $(1, 1, 1)$ and $(0, 0, 0)$, and does not converge to an optimal routing. The same behavior is observed if the cost function is modified by adding a term $\epsilon(x_1^2 + x_2^2 + x_3^2)$, which makes it strictly convex, as long as $\epsilon$ is positive and very small.

An important virtue of the algorithm of this section is that it is able to adapt to changes in the input traffic $r_w$ without having to be restarted. Simply, whenever $r_w$ changes, the processor in charge of OD pair $w$ continues performing gradient projection iterations, using the new value of $r_w$. As long as the time variations in $r_w$ are slow compared to the speed of convergence of the algorithm, the algorithm will keep generating flows that at any time are close to the optimal ones, given the current value of $r_w$.

There is an alternative distributed implementation of the algorithm, in which the estimates $\lambda_p(t)$ are formed somewhat differently. In particular, let node $i$ evaluate the derivative $D'_{ij}(\tilde{F}_{ij})$, where $\tilde{F}_{ij}$ is defined by Eq. (6.6), and then transmit this value to all other nodes. Then the node in control of the variable $x_p$ evaluates $\lambda_p$ by adding these derivatives along the path $p$. Essentially, this amounts to substituting $\tilde{F}_{ij}$ for $\hat{F}_{ij}$ in Eq. (6.9). With this modification, the proof of Prop. 6.1 remains valid, with a minor change in the proof of Lemma 6.4.

We finally mention another form of the algorithm in which the basic iteration (6.13)–(6.14) is replaced by the scaled gradient projection iteration

$$\bar{x}_w(t+1) = \Big[\bar{x}_w(t) - \gamma M_w^{-1}(t)\lambda_w(t)\Big]^+_{M_w(t)}, \qquad t \in T^w. \tag{6.35}$$

Here $M_w(t)$ is a positive definite symmetric matrix, and $[\cdot]^+_{M_w(t)}$ denotes projection on the set

$$\Big\{x_w \mid \sum_{p\in P_w} x_p = r_w \text{ and } x_p \geq 0,\ \forall p \in P_w\Big\}$$

[cf. Eqs. (6.1) and (6.2)], with respect to the norm $\|x_w\|_{M_w(t)} = \big(x_w' M_w(t) x_w\big)^{1/2}$ (see Subsection 3.3.3). This algorithm also converges (Exercise 6.2); however, it has the undesirable feature that the next value $\bar{x}_w(t+1)$ is determined in terms of the current value $\bar{x}_w(t)$ of the desired flows, as opposed to the current value $x_w(t)$ of the actual path flows, and this can slow down the adaptation to sudden changes in the problem data $r_w$. A more reasonable alternative, in which $x_w(t)$ replaces $\bar{x}_w(t)$ in Eq. (6.35), might be

$$\bar{x}_w(t) = \Big[x_w(t) - \gamma M_w^{-1}(t)\lambda_w(t)\Big]^+_{M_w(t)}, \qquad t \in T^w. \tag{6.36}$$

Surprisingly, however, this algorithm need not have the descent property and is not guaranteed to converge (see Exercise 6.3).

## EXERCISES

**6.1.** This exercise provides an example of a rerouting method for which the model of Eqs. (6.10) and (6.11) is valid. Consider a particular OD pair $w$ and the associated paths $P_w$. We assume that at any given time $t$, the traffic of such an OD pair consists of a number $n(t)$ of sessions that are simultaneously active. Each active session generates a constant flow of $\delta$ bits per second, which has to be routed from the origin to the destination. We assume that there is a very large number of sessions, each generating a very small flow, which allows us to view the total traffic $r(t) = \delta n(t)$ as a continuous variable. We assume that new sessions are "born" at a rate of $\lambda$ per second, and that existing sessions terminate at a rate of $\mu n(t)$ per second, where $\lambda$ and $\mu$ are positive constants. Thus, the total traffic $r(t)$ satisfies the differential equation

$$\frac{dr}{dt}(t) = \delta\lambda - \mu r(t).$$

In the long run, equilibrium is reached, we have $(dr/dt)(t) = 0$, and the traffic $r(t)$ is constant and equal to $r^* = \delta\lambda/\mu$.

Let $x_p(t)$ be the traffic routed through a particular path $p \in P_w$, and let $\bar{x}_p(t)$ be the corresponding desired flow. Suppose that once a session is born, it is assigned to some path and it cannot be reassigned before it terminates. Thus, routing can be only controlled by assigning paths to new sessions. Let $\rho_p(t)$ be the proportion of new sessions generated at time $t$ that are assigned to path $p$. Then the path flow $x_p(t)$ satisfies the equation

$$\frac{dx_p}{dt}(t) = \delta\lambda\rho_p(t) - \mu x_p(t).$$

Suppose that $\rho_p(t) = \bar{x}_p(t)/r^*$. Furthermore, suppose that the desired path flows $\bar{x}_p(t)$ can only change at integer times. Show that Eqs. (6.10) and (6.11) hold for every integer time $t$.

**6.2.** Suppose that the update equations (6.13) and (6.14) are replaced by Eq. (6.35). We assume that the matrices $M_w(t)$ are symmetric, bounded, and that there exists some $h > 0$ such that $M_w(t) - hI$ is nonnegative definite for each $w$ and $t$. Show that Prop. 6.1 remains valid. Assume for simplicity, that $x(0) = \bar{x}(0)$.
*Hints:* Let $s(t) = \bar{x}(t+1) - \bar{x}(t)$. Use the Scaled Projection Theorem of Section 3.3 to show that

$$\lambda_w(t)'s_w(t) \leq -\frac{A}{\gamma}\|s_w(t)\|^2, \qquad \forall t,$$

where $A$ is a positive constant (this replaces Lemma 6.2). Then show that there exist constants $A > 0$ and $\rho \in (0,1)$ such that

$$\big\|\lambda(t) - \nabla D\big(x(t)\big)\big\| \leq A\sum_{\tau=0}^{t}\rho^{t-\tau}\big\|s(\tau)\big\|, \qquad \forall t.$$

(This replaces Lemma 6.3.) Next, use the inequality

$$D\big(\bar{x}(t+1)\big) \leq D\big(\bar{x}(t)\big) + s(t)'\nabla D\big(\bar{x}(t)\big) + \frac{K}{2}\|s(t)\|^2$$

and proceed as in inequality (6.31) to show that $s(t)$ converges to zero.

**6.3.** Suppose that the update equations (6.13) and (6.14) are replaced by Eq. (6.36). Consider the simple case where there is a single OD pair (we accordingly suppress the superscript or subscript $w$ from our notation). Furthermore, consider the synchronous scenario where the set $T$ of update times is the set of all nonnegative integers, $\lambda(t) = \nabla D\big(x(t)\big)$ for all $t$, and the scaling matrix $M(t)$ is equal to a fixed symmetric positive definite matrix $M$. Let $A(t)$ be a diagonal matrix with $a_p(t)$ being the $p$th diagonal entry. Then the iteration (6.36) and the flow adjustment equation (6.11) become

$$\bar{x}(t) = \Big[x(t) - \gamma M^{-1}\nabla D\big(x(t)\big)\Big]^+_M, \tag{6.37}$$

$$x(t+1) = x(t) + A(t)\big(\bar{x}(t) - x(t)\big). \tag{6.38}$$

It is assumed, of course, that the coefficients $a_p(t)$ satisfy the conditions $0 < \alpha \leq a_p(t) \leq 1$ and

$$\sum_p a_p(t)\big(\bar{x}_p(t) - x_p(t)\big) = 0$$

so that Eq. (6.38) keeps $x(t+1)$ in the feasible set. Show (possibly by means of a figure) that the iteration (6.37)–(6.38) is not guaranteed to have the descent property $D\big(x(t+1)\big) \leq D\big(x(t)\big)$ no matter how small $\gamma$ is chosen. *Hint:* The matrix $A(t)M^{-1}$ is not necessarily positive definite.

## 7.7 A MODEL IN WHICH SEVERAL PROCESSORS UPDATE THE SAME VARIABLES

We now present a model of distributed iterative computation in which several processors are allowed to update the same component of the vector being iterated. As the updates of different processors will be generally different, the agreement algorithm of Section 7.3 will be used to reconcile the individual updates. This model will be used in the next section in the context of a stochastic gradient–like algorithm.

We motivate the model of this section by means of a simple example. Let $y$ be a random variable with unknown finite mean $m$ and variance $\sigma^2$. Suppose that there are $p$ processors and that the $i$th processor observes, at times $t = 1, 2, \ldots$, an independent realization (sample value) of $y$, to be denoted by $y_i(t)$. [For instance, the processors might be generating $y_i(t)$ by Monte Carlo simulation; alternatively, the observations $y_i(t)$ could be sensor data obtained at geographically distributed sensor sites.] The objective of the processors is to estimate the mean of $y$. At any time $t \geq 1$, each processor can compute the estimate $m_i(t) = \big(\sum_{\tau=1}^{t} y_i(\tau)\big)/t$ based on its own data. Furthermore, the computation of $m_i(t)$ can be carried out recursively by means of the formula

$$m_i(t+1) = m_i(t) + \frac{1}{t+1}\big(y_i(t+1) - m_i(t)\big),$$

initialized with $m_i(0) = 0$. This equation resembles the equation $x(t+1) = x(t) + \gamma s(t)$ of Section 7.5 [cf. Eq. (5.3)]; here $1/(t+1)$ plays the role of a (time–varying and decreasing) stepsize $\gamma$ and $y_i(t+1) - m_i(t)$ plays the role of $s(t)$.

The variance $E\big[(m_i(t) - m)^2\big]$ of $m_i(t)$ is equal to $\sigma^2/t$. A better estimate, with variance $\sigma^2/(tp)$, is obtained if the processors were to exchange their individual estimates at time $t$ and compute their average. This averaging could be performed once and for all at the end of the algorithm. However, it may be desirable that the processors obtain good estimates of $m$ while the generation of new observations $y_i(t)$ is in progress. This necessitates that the averaging of their individual estimates be performed more frequently. One possibility is that at each time $t$, once the processors obtain the new realizations $y_i(t)$ and compute their new individual estimates, they exchange and average their estimates. In this case, the computation consists of interleaved phases involving computation of $y_i(t)$ and averaging. This is a synchronous algorithm, which may be undesirable if, say, some processors are faster than others or if communication delays are too long.

An alternative approach is to overlap the individual computations and the averaging process. This can be done by letting the processors execute the agreement algorithm of Section 7.3, trying to agree on a common estimate, while they keep obtaining new samples $y_i(t)$, which they immediately incorporate into their estimates. We are faced here with two opposing effects: the agreement algorithm tends to bring the estimates of the processors closer together, whereas the generation of new samples has the potential of increasing the difference of their estimates. The interaction of these two effects substantially complicates the question of asynchronous convergence.

We now proceed to the formal description of the model to be employed. The reader is advised at this point that the notation to be introduced is somewhat different from earlier sections. This is because components will no more be associated with a unique

processor; an additional superscript will be used to specify the processor transmitting a message with the value of some component. The general notational convention to be followed here and in the next section is that superscripts denote processors and subscripts denote components.

We consider an algorithm that iterates on a vector $x$ belonging to the vector space $\Re^n$. Furthermore, we assume that $\Re^n$ is expressed as a Cartesian product of $m$ subspaces of lower dimensions, that is, $\Re^n = \Re^{n_1} \times \cdots \times \Re^{n_m}$, where $n_1 + \cdots + n_m = n$. Accordingly, any vector $x \in \Re^n$ is decomposed as $x = (x_1, \ldots, x_m)$, with $x_\ell$ belonging to $\Re^{n_\ell}$. We refer to $x_\ell$ as the $\ell$th component of $x$.

Let there be $p$ processors. Each processor $i$ has available at each time $t$ a vector $x^i(t) \in \Re^n$, with components $x^i_\ell(t) \in \Re^{n_\ell}$, $\ell = 1, \ldots, m$. For every component index $\ell$, we let $C_\ell \subset \{1, \ldots, p\}$ be the set of processors who are in charge of updating the $\ell$th component $x_\ell$, based on their own measurements or computations. We call $C_\ell$ the set of *computing processors* for component $\ell$ and we assume that it is nonempty for each $\ell$. If $i \in C_\ell$, we let $T^i_\ell$ be the set of times at which processor $i$ updates $x_\ell$.

For every processor $i$, any component $x_\ell$, and any time $t$, let $\gamma^i_\ell(t)s^i_\ell(t)$ be the update in $x_\ell$ due to a computation by processor $i$. Here $\gamma^i_\ell(t)$ is a positive stepsize and $s^i_\ell(t) \in \Re^{n_\ell}$ is an update direction. We could, without loss of generality, absorb $\gamma^i_\ell(t)$ into $s^i_\ell(t)$; our choice of notation is dictated by the application to be considered in the next section. Naturally, we assume that if $i \notin C_\ell$ or if $t \notin T^i_\ell$, then no computation is performed and $s^i_\ell(t) = 0$.

For every component index $\ell$, there is a directed graph $G_\ell = (N, A_\ell)$ used to model the exchange of messages carrying a value of $x_\ell$. The node set $N$ is the set $\{1, \ldots, p\}$ of processors and the arc set $A_\ell$ is the set of all pairs $(j, i)$ such that processor $j$ keeps sending messages $x_\ell$ to processor $i$. For $(j, i) \in A_\ell$, we let $T^{ij}_\ell$ be the set of times that such a message is received by processor $i$ and we assume that this set is infinite. For any $t \in T^{ij}_\ell$, we use $\tau^{ij}_\ell(t)$ to denote the time at which the message $x^j_\ell$ (received by $i$ at time $t$) was transmitted by processor $j$. Thus, the message received by processor $i$ at time $t$ is equal to $x^j_\ell\big(\tau^{ij}_\ell(t)\big)$. We assume that $1 \leq \tau^{ij}_\ell(t) \leq t$. We also use the convention that $T^{ii}_\ell$ is the set of all nonnegative integers and that $\tau^{ii}_\ell(t) = t$, for all $t$.

The algorithm is initialized at time 1 with each processor $i$ having a vector $x^i(1) \in \Re^n$ in its memory. Processor $i$ can, at any time $t$, receive messages $x^j_\ell\big(\tau^{ij}_\ell(t)\big)$ from other processors, incorporate these messages into its memory by forming a convex combination with its own value $x^i_\ell(t)$, and finally incorporate the result $\gamma^i_\ell(t)s^i_\ell(t)$ of its own computations. We thus postulate that for every $i$ and $\ell$, the variable $x^i_\ell(t)$ is updated according to the formula

$$x^i_\ell(t+1) = \sum_{\{j \mid t \in T^{ij}_\ell\}} a^{ij}_\ell(t) x^j_\ell\big(\tau^{ij}_\ell(t)\big) + \gamma^i_\ell(t)s^i_\ell(t), \tag{7.1}$$

where the coefficients $a^{ij}_\ell(t)$ are nonnegative scalars satisfying

$$\sum_{\{j \mid t \in T^{ij}_\ell\}} a^{ij}_\ell(t) = 1, \qquad \forall \ell, t, i.$$

Equation (7.1) is the general description of the algorithm but it takes somewhat simpler forms in special cases. For example, if processor $i$ receives no messages $x_\ell^j$ at time $t$, then $\{j \mid t \in T_\ell^{ij}\} = \{i\}$ and Eq. (7.1) simplifies to

$$x_\ell^i(t+1) = x_\ell^i(t) + \gamma_\ell^i(t)s_\ell^i(t).$$

Similarly, if $i \notin C_\ell$ or if $t \notin T_\ell^i$, then $s_\ell^i(t) = 0$ and Eq. (7.1) becomes

$$x_\ell^i(t+1) = \sum_{\{j \mid t \in T_\ell^{ij}\}} a_\ell^{ij}(t)x_\ell^j\big(\tau_\ell^{ij}(t)\big). \tag{7.2}$$

Equation (7.2) resembles the equations defining the agreement algorithm of Section 7.3, although it is somewhat more general, due to the dependence of the coefficients $a_\ell^{ij}(t)$ on time. In any case, the iteration (7.2) will be referred to as the *underlying agreement algorithm.* Notice that we essentially have a decoupled set of agreement algorithms, one for each component $x_\ell$. Coupling between components can arise, however, in Eq. (7.1) because $s_\ell^i(t)$ can depend on the entire vector $x^i(t)$.

Given our motivation of the model (7.1), we want to ensure that the underlying agreement algorithm (7.2) works properly. That is, we would like the values of the variables $x_\ell^i$, possessed by different processors, to converge to a common value in the absence of further computations. This turns out to be the case under some reasonable assumptions.

**Example 7.1.**

We show that the agreement algorithm of Section 7.3 is a special case of Eq. (7.2). Suppose that for each processor $i$ and for every $j$ who communicates $x_\ell$ to $i$ [i.e., if $(j,i) \in A_\ell$], the set $T_\ell^{ij}$ does not depend on $j$. We are thus assuming that $T_\ell^{ij} = R_\ell^i$ for all $j$ such that $(j,i) \in A_\ell$, where $R_\ell^i$ is some set. Accordingly, at any time $t$, either $t \notin R_\ell^i$ and processor $i$ receives no message $x_\ell^j$ or $t \in R_\ell^i$ and processor $i$ receives simultaneously messages $x_\ell^j$ from all processors $j$ that are supposed to send such messages. This is not as unrealistic as it sounds. For example, processor $i$ might physically receive these messages at different times, store them in a buffer, and then read them all at the same time $t$. As far as the mathematical description of the algorithm is concerned, this situation is identical to the case of simultaneous receptions.

We now assume for each $(j,i) \in A_\ell$ and $t \in R_\ell^i$, that the value of the coefficient $a_\ell^{ij}(t)$ is positive and independent of $t$, and will be denoted by $a_\ell^{ij}$. Let us define for convenience $a_\ell^{ij} = 0$ if $j \neq i$ and $(j,i) \notin A_\ell$. Then Eq. (7.2) can be rewritten as

$$x_\ell^i(t+1) = \sum_{j=1}^{p} a_\ell^{ij}x_\ell^j\big(\tau_\ell^{ij}(t)\big), \qquad \text{if } t \in R_\ell^i,$$

$$x_\ell^i(t+1) = x_\ell^i(t), \qquad \text{if } t \notin R_\ell^i.$$

These two equations are identical to those describing the agreement algorithm of Section 7.3 [cf. Eqs. (3.2) and (3.3)], except for somewhat different notation. If we now introduce the partial asynchronism assumption and assume that there exists some $i$ such that $a_\ell^{ii} > 0$ and

such that there exists a positive path (in the graph $G_\ell$) from $i$ to all other processors, then the sequence generated by Eq. (7.2) is guaranteed to converge to agreement, geometrically.

**Example 7.2.**

We now consider an alternative to Example 7.1, in which received messages are immediately incorporated in the memory of the receiving processor. For any $i$ and $\ell$ and any $j \neq i$ such that $(j,i) \in A_\ell$, we are given a constant $a_\ell^{ij} \in (0,1)$. Furthermore, we assume that each processor $i$ receives at most one message $x_\ell^j$ at each time unit; equivalently, the sets $T_\ell^{ij}$, with $j \neq i$, are disjoint for any fixed $i$. In practice, physical message receptions could be simultaneous, but a processor can always place incoming messages in a buffer and read them one at a time. Thus, our assumption is not entirely unrealistic. Suppose that for every $t \in T_\ell^{ij}$, the incoming message $x_\ell^j\big(\tau_\ell^{ij}(t)\big)$ is taken into account by processor $i$ by letting

$$x_\ell^i(t+1) = a_\ell^{ij} x_\ell^j\big(\tau_\ell^{ij}(t)\big) + (1 - a_\ell^{ij}) x_\ell^i(t) + \gamma_\ell^i(t) s_\ell^i(t), \qquad t \in T_\ell^{ij}.$$

We also let $x_\ell^i(t+1) = x_\ell^i(t) + \gamma_\ell^i(t) s_\ell^i(t)$ if $t$ does not belong to any $T_\ell^{ij}$. It is easily seen that we are dealing with a special case of Eq. (7.1). The underlying agreement algorithm is identical to the one considered in Exercise 3.2 of Section 7.3. According to the result of that exercise, such an agreement algorithm is guaranteed to converge geometrically if the partial asynchronism assumption holds and provided that for some $i$, there exists a positive path (in the graph $G_\ell$) from $i$ to every other processor $j$.

Examples 7.1 and 7.2 demonstrate that we can realistically assume that the underlying agreement algorithm (7.2) converges geometrically and such an assumption will be introduced shortly. Before doing so, we shall develop an alternative representation that is equivalent to Eq. (7.1) but easier to work with.

Suppose that we have fixed the sets $T_\ell^{ij}$ and the values of the coefficients $a_\ell^{ij}(t)$ and $\tau_\ell^{ij}(t)$ for all $\ell, i, j, t \in T_\ell^{ij}$. (We refer to any fixed choice of these sets and variables as a *realization* of the underlying agreement algorithm.) Let us now fix some component index $\ell$. It is seen from Eq. (7.1) that for any fixed $t$ and $i$, the value of $x_\ell^i(t)$ is a linear function of the variables $\{x^j(1) \mid j = 1, \ldots, p\}$ and $\{\gamma_\ell^j(\tau) s_\ell^j(\tau) \mid j = 1, \ldots, p,\ \tau = 1, \ldots, t-1\}$. (This is easily proved by induction, using the fact that the composition of two linear functions is linear.) It follows that there exist scalar coefficients $\Phi_\ell^{ij}(t,\tau)$, $t > \tau \geq 0$, such that

$$x_\ell^i(t) = \sum_{j=1}^{p} \Phi_\ell^{ij}(t,0) x_\ell^j(1) + \sum_{\tau=1}^{t-1} \sum_{j=1}^{p} \Phi_\ell^{ij}(t,\tau) \gamma_\ell^j(\tau) s_\ell^j(\tau). \tag{7.3}$$

Equation (7.3) is more explicit than the original Eq. (7.1). On the other hand, the coefficients $\Phi_\ell^{ij}(t,\tau)$ depend on the particular realization of the underlying agreement algorithm and are therefore usually unknown.

**Example 7.3.**

The best way of visualizing the coefficient $\Phi_\ell^{ij}(t,\tau)$ is by considering the following sequence of events. Let us fix $j$, $\ell$, and $\tau$. For simplicity, suppose that each component is one-dimensional ($n_\ell = 1$) and that $x_\ell^k(1) = 0$ for all $k$. Suppose that processor $j$ performs a

computation at time $\tau$ and obtains $\gamma_\ell^j(\tau)s_\ell^j(\tau) = 1$. Furthermore, suppose that this is the only computation ever to be performed by any processor, that is $s_\ell^k(\sigma) = 0$, unless $k = j$ and $\sigma = \tau$. For such a sequence of events, Eq. (7.3) shows that for $t > \tau$, $\Phi_\ell^{ij}(t,\tau) = x_\ell^i(t)$. In other words, $\Phi_\ell^{ij}(t,\tau)$ is the value (at time $t$ and at processor $i$) generated by the underlying agreement algorithm, initialized at time $\tau+1$ with $x^j(\tau+1) = 1$, $x^k(\tau+1) = 0$ for $k \neq j$, and with all messages in transit at time $\tau+1$ equal to zero.

There are several conclusions that can be drawn from Example 7.3. First, we must have

$$0 \leq \Phi_\ell^{ij}(t,\tau) \leq 1, \qquad \forall i,j,\ell,t > \tau.$$

Furthermore, if the underlying agreement algorithm is geometrically convergent (as is the case in Examples 7.1 and 7.2), then the coefficients $\Phi_\ell^{ij}(t,\tau)$ converge geometrically, as $t$ tends to infinity, to a limit independent of $i$. We introduce the notation $\Phi_\ell^j(\tau)$ to denote this common limit. The variable $\Phi_\ell^j(\tau)$ is interpreted as the value of the component $x_\ell$ on which all processors would agree under the sequence of events specified in Example 7.3.

Finally, let us notice that if $i$ is a computing processor for component $\ell$ (that is, if $i \in C_\ell$), then it is desirable that each update $\gamma_\ell^i(t)s_\ell^i(t)$ of processor $i$ can influence the value of $x_\ell^j$ for all other processors $j$ by a nonnegligible amount in the long run. This can be expressed mathematically by requiring that there exists a positive constant $\eta$ such that $\Phi_\ell^i(\tau) \geq \eta$ for all $i \in C_\ell$ and all $\tau \geq 0$.

**Example 7.1.** ( *cont.* )

Suppose that for every computing processor $i$ for component $\ell$ (i.e., $i \in C_\ell$), we have $a_\ell^{ii} > 0$ and that there exists a positive path in the graph $G_\ell$ from processor $i$ to every other processor $j$. Then processor $i$ is a "distinguished" processor in the terminology of Section 7.3 and part (c) of Prop. 3.1 applies. In particular, if $x_\ell^k(1) = 0$ for every $k$ and $\ell$, and if $\gamma_\ell^i(\tau)s_\ell^i(\tau) = 1$ is the only nonzero update in the course of the algorithm (as in Example 7.3), then the value on which the processors will eventually agree is no smaller than the positive constant $\eta$ of Prop. 3.1(c). We have already shown that the agreed upon value is equal to $\Phi_\ell^i(\tau)$ (Example 7.3). We conclude that the inequality $\Phi_\ell^i(\tau) \geq \eta > 0$ holds for all $\tau \geq 0$. A similar argument can be made for the agreement algorithm of Example 7.2.

We summarize the above discussion in Assumption 7.1 to follow. In particular, we have shown that Assumption 7.1 is satisfied when the underlying agreement algorithm is as in Example 7.1 or Example 7.2, provided that for each component $\ell$, there is a communication path from every computing processor to every other processor. This assumption can be also shown to hold for more general choices of the underlying agreement algorithm [Tsi84].

**Assumption 7.1.** The sets $T_\ell^{ij}$ and the variables $a_\ell^{ij}(t)$, $\tau_\ell^{ij}(t)$, defining a realization of the underlying agreement algorithm [cf. Eqs. (7.1) and (7.2)], are such that the following hold:

**(a)** For all $i$, $j$, and $t > \tau \geq 0$, we have $0 \leq \Phi_\ell^{ij}(t,\tau) \leq 1$.

**(b)** For any $i$, $j$, and $\tau \geq 0$, the limit of $\Phi_\ell^{ij}(t,\tau)$, as $t$ tends to infinity, exists, is the same for all $i$, and is denoted by $\Phi_\ell^j(\tau)$.

**(c)** There exists some $\eta > 0$ (independent of the particular realization) such that $\Phi_\ell^j(\tau) \geq \eta$ for all $j \in C_\ell$ and $\tau \geq 0$.

**(d)** There exist constants $A > 0$ and $\rho \in (0,1)$ (independent of the particular realization) such that

$$|\Phi_\ell^{ij}(t,\tau) - \Phi_\ell^j(\tau)| \leq A\rho^{t-\tau}, \qquad \forall t > \tau \geq 0.$$

Equation (7.3) has a simple structure, but is still difficult to manipulate when it comes to the analysis of specific algorithms. The reason is that we have one such equation for each $\ell$ and $i$, and all of these equations are, in general, coupled. Thus, we need to keep track of all the vectors $x^1(t), \ldots, x^p(t)$, simultaneously. Analysis would be easier if we could associate with the algorithm a single vector $y(t)$ that summarizes the information contained in the vectors $x^i(t)$. It turns out that this is possible and the following choice of $y(t)$ is particularly useful. We define the $\ell$th component of $y(t)$ by

$$y_\ell(t) = \sum_{i=1}^{p} \Phi_\ell^i(0) x_\ell^i(1) + \sum_{\tau=1}^{t-1} \sum_{i=1}^{p} \Phi_\ell^i(\tau)\gamma_\ell^i(\tau)s_\ell^i(\tau), \qquad \ell = 1, \ldots, m. \tag{7.4}$$

The interpretation of $y(t)$ is quite interesting. Let us fix some time $\bar{t}$ and suppose that $\gamma_\ell^i(\tau)s_\ell^i(\tau) = 0$ for all $\tau \geq \bar{t}$. Consider Eq. (7.3) and take the limit as $t \to \infty$. We only need to keep the summands for $\tau < \bar{t}$ and we are left with the defining expression for $y_\ell(\bar{t})$. In other words, if the processors stop performing any updates at time $\bar{t}$, but keep communicating and forming convex combinations of their states, as in the underlying agreement algorithm, they will asymptotically agree and $y(\bar{t})$ is precisely the vector on which they will agree.

The vector $y(t)$ is a very convenient tool for analysis, primarily because it is generated by the recursion

$$y_\ell(t+1) = y_\ell(t) + \sum_{i=1}^{p} \Phi_\ell^i(t)\gamma_\ell^i(t)s_\ell^i(t), \qquad \ell = 1, \ldots, m, \tag{7.5}$$

which is an immediate consequence of Eq. (7.4)

**Example 7.4.** *Specialized Computation*

We now show that our earlier model of asynchronous computation in which each processor is associated with a different component (e.g., as in Section 7.5) is a special case of the present model, and we evaluate the coefficients $\Phi_\ell^i(t)$ for this particular context. Let the number $m$ of components be the same as the number $p$ of processors, and let processor $j$ be the only computing processor for component $j$, that is, $C_j = \{j\}$. Each processor $j$ broadcasts its value of $x_j$ from time to time to all other processors and these values are received after some bounded delay. Processor $i$, upon receiving at time $t$ a value of $x_j$ that

has been sent at some time $\tau_j^{ij}(t) \leq t$ by some processor $j \neq i$ [thus, processor $i$ receives the message $x_j^j\big(\tau_j^{ij}(t)\big)$], lets

$$x_j^i(t+1) = x_j^j\big(\tau_j^{ij}(t)\big). \tag{7.6}$$

Also, processor $i$ lets

$$x_j^i(t+1) = x_j^i(t), \tag{7.7}$$

if no message is received from processor $j \neq i$ at time $t$. Equations (7.6) and (7.7) are a trivial special case of an agreement algorithm. Comparing with Eq. (7.1), we have $a_j^{ij}(t) = 1$, if processor $i$ receives a message $x_j^j$ from $j$ at time $t$. In this example, if processor $j$ stops performing any computations, the values $x_j^i(t)$ possessed by different processors all agree, within finite time. (Finite time convergence can be viewed as a special case of geometric convergence.) Furthermore, the value on which they agree is the value possessed by the computing processor $j$. Using the interpretation of the coefficients $\Phi_i^j(t)$ provided by Example 7.3, we see that $\Phi_j^j(t) = 1$ for all $j$ and $t$, and $\Phi_i^j(t) = 0$ if $i \neq j$.

The model we have developed will be used in the next section, in the context of a stochastic gradient algorithm.

## 7.8 STOCHASTIC GRADIENT ALGORITHMS

Let $F : \Re^n \mapsto \Re$ be a differentiable cost function to be minimized, subject to no constraints. Suppose, however, that we only have access to noisy measurements of the gradient. That is, for any $x \in \Re^n$, we cannot compute $\nabla F(x)$, but we have access to $\nabla F(x) + w$, where $w$ is a random variable representing measurement or observation noise. Such a situation often arises in statistics or in system identification ([RoM51], [Lju77], [LjS83], and [PoT73]).

One could still try to use the gradient algorithm, as if no noise was present, which gives rise to the equation

$$x(t+1) = x(t) - \gamma\Big(\nabla F\big(x(t)\big) + w(t)\Big), \tag{8.1}$$

where $w(t)$ is the noise in the measurement of $\nabla F\big(x(t)\big)$. This algorithm does not converge, in general, to the minimum of $F$. For example, suppose that for each $t$, the random variable $w(t)$ is independent of $x(t)$ and has variance $\sigma^2 > 0$. It follows from Eq. (8.1) that the variance of $x(t)$ is at least $\gamma^2\sigma^2$ and, therefore, $x(t)$ fails to converge (in the mean square sense). What may happen, at best, is that $x(t)$ reaches a neighborhood of a (local) minimum $x^*$ of $F$ and moves randomly around $x^*$. The mean square distance of $x(t)$ from $x^*$ increases with the variance of $x(t)$ and can be made small only if $\gamma$ is chosen small. On the other hand, if $\gamma$ is excessively small, it can take a very large number of steps to reach a neighborhood of a local minimum. We can strike a balance

between these two effects by employing a time–varying stepsize $\gamma(t)$ and replacing Eq. (8.1) by

$$x(t+1) = x(t) - \gamma(t)\Big(\nabla F\big(x(t)\big) + w(t)\Big). \tag{8.2}$$

The stepsize $\gamma(t)$ is initially large, which allows a rapid approach to the vicinity of a minimizing point, and then is decreased to zero so that, hopefully, the variance of $x(t)$ also decreases to zero.

Some typical conditions on the choice of the stepsize are

$$\sum_{t=1}^{\infty} \gamma(t) = \infty, \tag{8.3}$$

$$\sum_{t=1}^{\infty} \gamma^2(t) < \infty. \tag{8.4}$$

Condition (8.3) is needed because, otherwise, $x(t)$ might not be able to travel far enough to get to a minimum of $F$; this condition would be required even if no noise was present. Condition (8.4) is used to ensure that the variance of $x(t)$ converges to zero. A choice for $\gamma(t)$, satisfying conditions (8.3) and (8.4) and commonly used in analytical studies, is $\gamma(t) = 1/t$.

**Example 8.1.**

Let $x$ be one–dimensional and $F(x) = \frac{1}{2}x^2$. Then Eq. (8.2) becomes $x(t+1) = \big(1 - \gamma(t)\big)x(t) + \gamma(t)w(t)$. Let $V(t)$ be the variance of $x(t)$. If $w(t)$ is independent of $x(t)$ and has variance $\sigma^2$, then the variance of $x(t+1)$ is the sum of the variances of $\big(1-\gamma(t)\big)x(t)$ and $\gamma(t)w(t)$. Thus,

$$V(t+1) = \big(1-\gamma(t)\big)^2 V(t) + \gamma^2(t)\sigma^2.$$

Assuming that conditions (8.3) and (8.4) hold, it can be shown that $V(t)$ converges to zero (Exercise 8.1).

The algorithm (8.2) admits a straightforward distributed implementation, similar to the distributed deterministic gradient algorithm of Section 7.5, whereby each processor is assigned the task of updating a particular component of $x$. However, in the presence of noise, it makes sense to have several processors work on the same component of $x$. In particular, if the noise in the measured value of the gradient is independent for different processors, then the different processors can average their individual updates and come up with a common update. This is expected to improve the speed of convergence because the averaging of independent identically distributed random variables always reduces the variance. Nevertheless, we might wish to avoid a synchronous implementation of this updating and averaging procedure, and this leads us to the model introduced in Section 7.7.

### 7.8.1 Description of the Algorithm and Assumptions

Throughout this section, $\|\cdot\|$ will stand for the Euclidean norm $\|\cdot\|_2$. We make the same assumptions on $F$ as in Section 7.5.

**Assumption 8.1.**

**(a)** There holds $F(x) \geq 0$ for every $x \in \Re^n$.

**(b)** (*Lipschitz Continuity of* $\nabla F$) The function $F$ is continuously differentiable and there exists a constant $K_1$ such that

$$\|\nabla F(x) - \nabla F(y)\| \leq K_1 \|x - y\|, \qquad \forall x, y \in \Re^n. \tag{8.5}$$

We assume that vectors in $\Re^n$ are decomposed into $m$ components. As in Section 7.7, we assume that the algorithm is described by the equation [cf. Eq. (7.1)]

$$x_\ell^i(t+1) = \sum_{\{j \mid t \in T_\ell^{ij}\}} a_\ell^{ij}(t) x_\ell^j\big(\tau_\ell^{ij}(t)\big) + \gamma_\ell^i(t) s_\ell^i(t).$$

Furthermore, the underlying agreement algorithm is described by [cf. Eq. (7.2)]

$$x_\ell^i(t+1) = \sum_{\{j \mid t \in T_\ell^{ij}\}} a_\ell^{ij}(t) x_\ell^j\big(\tau_\ell^{ij}(t)\big),$$

and will be assumed to satisfy Assumption 7.1. We assume that the sets $T_\ell^{ij}$ and the variables $a_\ell^{ij}(t)$, $\tau_\ell^{ij}(t)$, defining the underlying agreement algorithm, are *deterministic*. (This does not imply that they are known ahead of time by the processors; it only means that they are not modeled as random variables.) We also assume that the initializations $x_\ell^i(1)$ are deterministic.

Concerning the stepsizes, we assume that there exist positive constants $K_2$ and $K_3$ such that

$$\frac{K_2}{t} \leq \gamma_\ell^i(t) \leq \frac{K_3}{t}, \quad \forall i, \ell, t. \tag{8.6}$$

As a practical matter, a stepsize satisfying condition (8.6) can be enforced without assuming that the processors know the current value of $t$. For example, suppose that each processor $i$ uses the following rule: let the current value of $\gamma_\ell^i$ be the reciprocal of the number of times that $i$ has performed an update. Assuming that the time between consecutive updates by each processor is bounded above and below by positive constants, it is easily verified that the resulting stepsizes satisfy inequality (8.6).

It remains to specify the update directions $s_\ell^i(t)$. An interesting special case suggested by Eq. (8.2) is to let $s_\ell^i(t) = -\big(\nabla_\ell F\big(x^i(t)\big) + w_\ell^i(t)\big)$ for $t \in T_\ell^i$, where the random variables $w_\ell^i(t)$ are independent identically distributed with finite variance. We shall actually introduce a somewhat more general set of assumptions.

We consider the set $\mathcal{F}(t)$ of random variables defined by

$$\mathcal{F}(t) = \Big\{ s_\ell^i(\tau) \mid i \in \{1,...,p\},\ \ell \in \{1,...,m\},\ \tau < t \Big\}.$$

Given our earlier assumptions on the deterministic nature of the underlying agreement algorithm and the initial conditions, the variables in the set $\mathcal{F}(t)$ are the only sources of randomness up to time $t$. We may therefore view $\mathcal{F}(t)$ as a representation of the entire history of the algorithm up to the moment that the update directions $s_\ell^i(t)$ are to be generated. Furthermore, for any choice of initial conditions and any fixed realization of the underlying agreement algorithm, it is seen that $x_\ell^i(t)$ is a uniquely determined function of the random variables in the set $\mathcal{F}(t)$.

**Assumption 8.2.** (*Stochastic Descent*) There exist positive constants $K_4$, $K_5$ and $K_6$ such that

$$\nabla_\ell F\big(x^i(t)\big)' E\big[s_\ell^i(t) \mid \mathcal{F}(t)\big] \le -K_4 \big\|\nabla_\ell F\big(x^i(t)\big)\big\|^2, \quad \forall t \in T_\ell^i, \tag{8.7}$$

$$E\Big[\big\|s_\ell^i(t)\big\|^2 \mid \mathcal{F}(t)\Big] \le K_5 \big\|\nabla_\ell F\big(x^i(t)\big)\big\|^2 + K_6, \quad \forall t \in T_\ell^i. \tag{8.8}$$

Furthermore $s_\ell^i(t) = 0$, for all $t \notin T_\ell^i$.

Assumption 8.2 is similar to Assumptions 5.2 and 5.5 of Section 7.5. Inequality (8.7) requires that the expected direction of the update $s_\ell^i(t)$, given the past history $\mathcal{F}(t)$ of the algorithm, is a descent direction. Notice the presence of the constant $K_6$ in inequality (8.8). This constant allows the algorithm to make nonzero updates even if a minimum has been reached. This is natural because a noisy measurement of the gradient at the minimum may be nonzero.

### 7.8.2 A Convergence Result

The synchronous version of the algorithm considered in this section is known to be convergent [PoT73]. The proof we provide here follows the same steps as the corresponding proof in [PoT73] and establishes that convergence is preserved in the presence of partial asynchronism. The main idea is that we use the vector $y(t)$ [cf. Eq. (7.4)] as a summary of the state of the algorithm at time $t$, we monitor the progress of the cost $F\big(y(t)\big)$, and we obtain an inequality of the form [cf. Eq. (8.16)]

$$F\big(y(t+1)\big) \le F\big(y(t)\big) - \gamma(t)G(t) + O\big(\gamma(t)^2\big).$$

Here the term $G(t)$ is nonnegative in a probabilistic sense, meaning that $E[G(t) \mid \mathcal{F}(t)] \ge 0$. The errors due to asynchronism are all incorporated into the $O\big(\gamma(t)^2\big)$ term. The proof is then completed using reasoning analogous to the deterministic case (Section 7.5) although some of the technical issues are more difficult.

**Proposition 8.1.** We assume the model of Section 7.7. We assume that $T_\ell^i$ is infinite for every computing processor $i$ for component $\ell$, and that the difference between consecutive elements of $T_\ell^i$ is bounded by some $B > 0$. We also assume that the underlying agreement algorithm satisfies Assumption 7.1 of Section 7.7. Finally, we suppose that Assumptions 8.1 and 8.2 hold and that the stepsizes satisfy condition (8.6). Then the following are true:

**(a)** The limit of $F\big(x^i(t)\big)$, as $t \to \infty$, exists and is the same for all $i$, with probability 1.

**(b)** The limit of $x^i(t) - x^j(t)$, as $t \to \infty$, is equal to zero with probability 1 and in the mean square sense.

**(c)** For every $i$, $\liminf_{t\to\infty} \nabla F\big(x^i(t)\big) = 0$.

**(d)** Suppose that the set $\{x \mid F(x) \leq c\}$ is bounded for every $c \in \Re$, that there exists a unique vector $x^*$ at which $F$ is minimized, and that this is the unique vector at which $\nabla F$ vanishes. Then $x^i(t)$ converges to $x^*$ for each $i$ with probability 1.

***Proof.*** We only prove the result under the assumption $x_\ell^i(1) = 0$ for all $i$ and $\ell$. There are only a few minor modifications needed to cover the general case.

Let us define $\bar{s}_\ell^i(t) = \big(t\gamma_\ell^i(t)\big)s_\ell^i(t)$ and view $\bar{s}_\ell^i(t)$ as the update direction, with stepsize $1/t$. Using inequality (8.6), we see that conditions (8.7) and (8.8) also hold for $\bar{s}_\ell^i(t)$, possibly with a different choice of constants $K_4$, $K_5$, $K_6$. Thus, without loss of generality, we can and will assume that $\gamma_\ell^i(t) = 1/t$ for all $i$, $\ell$, and $t$.

We use the notations $\Phi_\ell^{ij}(t,\tau)$, $\Phi_\ell^i(t)$, and $y(t)$ as in Section 7.7. We also define

$$b(t) = \sum_{i=1}^{p}\sum_{\ell=1}^{m} \|s_\ell^i(t)\|, \quad t \geq 1, \tag{8.9}$$

$$G(t) = -\sum_{i=1}^{p}\sum_{\ell=1}^{m} \Big(\nabla_\ell F\big(x^i(t)\big)\Big)' \Phi_\ell^i(t) s_\ell^i(t), \quad t \geq 1. \tag{8.10}$$

**Lemma 8.1.**

**(a)** If $t \in T_\ell^i$, then

$$E\big[G(t) \mid \mathcal{F}(t)\big] \geq K_4\eta \sum_{\{(i,\ell)\mid t\in T_\ell^i\}} \big\|\nabla_\ell F\big(x^i(t)\big)\big\|^2 \geq 0,$$

where $\eta > 0$ is the constant in Assumption 7.1(c).

**(b)** If $t \geq 1$, then

$$E\big[b^2(t) \mid \mathcal{F}(t)\big] \leq A_1 E\big[G(t) \mid \mathcal{F}(t)\big] + A_2,$$

where $A_1 = (mpK_5)/(\eta K_4)$ and $A_2 = m^2p^2K_6$.

***Proof.*** Using inequality (8.7) and the fact that $s_\ell^i(t) = 0$ for $t \notin T_\ell^i$, we obtain

$$\begin{aligned} E\big[G(t) \mid \mathcal{F}(t)\big] &= -\sum_{\{(i,\ell)\mid t\in T_\ell^i\}} \nabla_\ell F\big(x^i(t)\big)' \Phi_\ell^i(t) E\big[s_\ell^i(t) \mid \mathcal{F}(t)\big] \\ &\geq \sum_{\{(i,\ell)\mid t\in T_\ell^i\}} K_4 \big\|\nabla_\ell F\big(x^i(t)\big)\big\|^2 \Phi_\ell^i(t) \\ &\geq \eta K_4 \sum_{\{(i,\ell)\mid t\in T_\ell^i\}} \big\|\nabla_\ell F\big(x^i(t)\big)\big\|^2. \end{aligned} \tag{8.11}$$

This proves part (a) of the lemma. For part (b), we use inequality (8.8) to obtain

$$\begin{aligned} E\big[b^2(t) \mid \mathcal{F}(t)\big] &= E\left[\left(\sum_{i=1}^p \sum_{\ell=1}^m \big\|s_\ell^i(t)\big\|\right)^2 \;\Big|\; \mathcal{F}(t)\right] \leq pm \sum_{i=1}^p \sum_{\ell=1}^m E\Big[\big\|s_\ell^i(t)\big\|^2 \mid \mathcal{F}(t)\Big] \\ &\leq pm \sum_{\{(i,\ell)\mid t\in T_\ell^i\}} \Big(K_5 \big\|\nabla_\ell F\big(x^i(t)\big)\big\|^2 + K_6\Big) \\ &\leq \frac{pmK_5}{\eta K_4} E\big[G(t) \mid \mathcal{F}(t)\big] + p^2 m^2 K_6, \end{aligned} \tag{8.12}$$

where the last inequality follows from part (a). **Q.E.D.**

**Lemma 8.2.** For every $t \geq 1$, we have

$$\big\|y(t) - x^i(t)\big\| \leq A \sum_{\tau=1}^{t-1} \frac{1}{\tau} \rho^{t-\tau} b(\tau), \tag{8.13}$$

where $A > 0$ and $\rho \in (0, 1)$ are the constants of Assumption 7.1(d).

***Proof.*** We subtract Eq. (7.3) from Eq. (7.4) to obtain

$$y_\ell(t) - x_\ell^i(t) = \sum_{\tau=1}^{t-1} \sum_{j=1}^p \frac{1}{\tau} \Big[\Phi_\ell^j(\tau) - \Phi_\ell^{ij}(t,\tau)\Big] s_\ell^j(\tau). \tag{8.14}$$

We then use Assumption 7.1(d) to obtain

$$\big\|y_\ell(t) - x_\ell^i(t)\big\| \leq \sum_{\tau=1}^{t-1} \frac{1}{\tau} \sum_{j=1}^p A\rho^{t-\tau} \big\|s_\ell^j(\tau)\big\|. \tag{8.15}$$

We then sum inequality (8.15) over all components $\ell$ and use the definition of $b(\tau)$ to obtain inequality (8.13). **Q.E.D.**

We use Eq. (7.5) and the descent lemma (Prop. A.32 in Appendix A) to obtain

$$
\begin{aligned}
F\big(y(t+1)\big) &\le F\big(y(t)\big) + \frac{1}{t}\sum_{i=1}^{p}\sum_{\ell=1}^{m}\nabla_\ell F\big(y(t)\big)'\Phi_\ell^i(t)s_\ell^i(t) + \frac{K_1}{2t^2}\sum_{\ell=1}^{m}\Big\|\sum_{i=1}^{p}\Phi_\ell^i(t)s_\ell^i(t)\Big\|^2 \\
&\le F\big(y(t)\big) + \frac{1}{t}\sum_{i=1}^{p}\sum_{\ell=1}^{m}\nabla_\ell F\big(x^i(t)\big)'\Phi_\ell^i(t)s_\ell^i(t) \\
&\quad + \frac{1}{t}\sum_{i=1}^{p}\sum_{\ell=1}^{m}\Big\|\nabla_\ell F\big(x^i(t)\big) - \nabla_\ell F\big(y(t)\big)\Big\|\cdot\big\|\Phi_\ell^i(t)s_\ell^i(t)\big\| + \frac{K_1}{2t^2}b^2(t) \\
&\le F\big(y(t)\big) - \frac{1}{t}G(t) + \frac{1}{t}\sum_{i=1}^{p}\sum_{\ell=1}^{m}K_1\big\|x^i(t) - y(t)\big\|\cdot\big\|s_\ell^i(t)\big\| + \frac{K_1}{2t^2}b^2(t) \\
&\le F\big(y(t)\big) - \frac{1}{t}G(t) + \frac{K_1}{t}\sum_{i=1}^{p}\sum_{\ell=1}^{m}A\sum_{\tau=1}^{t-1}\frac{1}{\tau}\rho^{t-\tau}b(\tau)\big\|s_\ell^i(\tau)\big\| + \frac{K_1}{2t^2}b^2(t) \\
&= F\big(y(t)\big) - \frac{1}{t}G(t) + K_1A\sum_{\tau=1}^{t-1}\frac{1}{t\tau}\rho^{t-\tau}b(\tau)b(t) + \frac{K_1}{2t^2}b^2(t) \\
&\le F\big(y(t)\big) - \frac{1}{t}G(t) + K_1A\sum_{\tau=1}^{t-1}\rho^{t-\tau}\left(\frac{b^2(\tau)}{\tau^2} + \frac{b^2(t)}{t^2}\right) + \frac{K_1}{2t^2}b^2(t) \\
&\le F\big(y(t)\big) - \frac{1}{t}G(t) + A_3\sum_{\tau=1}^{t}\rho^{t-\tau}\frac{b^2(\tau)}{\tau^2},
\end{aligned}
\tag{8.16}
$$

where $A_3 = K_1A/(1-\rho) + K_1/2$.

**Lemma 8.3.** There holds

$$
\sum_{t=1}^{\infty}\frac{1}{t}E\big[G(t)\big] < \infty. \tag{8.17}
$$

***Proof.*** We take expectations of both sides of inequality (8.16) and use Lemma 8.1(b) to bound $E\big[b^2(t)\big]$. This yields

$$
E\big[F\big(y(t+1)\big)\big] \le E\big[F\big(y(t)\big)\big] - \frac{1}{t}E\big[G(t)\big] + A_3\sum_{\tau=1}^{t}\rho^{t-\tau}\frac{1}{\tau^2}\Big(A_1E\big[G(\tau)\big] + A_2\Big). \tag{8.18}
$$

We add inequality (8.18) for $t$ running from 1 to $\bar{t}$; after cancellations, we obtain

$$E\left[F\left(y(\bar{t}+1)\right)\right] \leq F\left(y(1)\right) - \sum_{t=1}^{\bar{t}} \frac{1}{t} E\left[G(t)\right] + A_2 A_3 \sum_{t=1}^{\bar{t}} \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2}$$

$$+ A_1 A_3 \sum_{t=1}^{\bar{t}} \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2} E\left[G(\tau)\right]$$

$$= F\left(y(1)\right) - \sum_{t=1}^{\bar{t}} \frac{1}{t} E\left[G(t)\right] \left(1 - A_1 A_3 \frac{1}{t} \sum_{\tau=t}^{\bar{t}} \rho^{\tau-t}\right) \tag{8.19}$$

$$+ A_2 A_3 \sum_{t=1}^{\bar{t}} \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2}$$

$$\leq F\left(y(1)\right) - \sum_{t=1}^{\bar{t}} \frac{1}{t} E\left[G(t)\right] \left(1 - \frac{A_1 A_3}{t(1-\rho)}\right) + A_2 A_3 \sum_{\tau=1}^{\bar{t}} \frac{1}{\tau^2(1-\rho)}.$$

Notice that $A_2 A_3 \sum_{\tau=1}^{\bar{t}} 1/\left(\tau^2(1-\rho)\right)$ is bounded above by some finite constant $A_4$, because the infinite sum $\sum_{\tau=1}^{\infty} 1/\tau^2$ is finite. Furthermore, $1 - A_1 A_3/\left(t(1-\rho)\right)$ is larger than $1/2$ when $t$ is sufficiently large. If $\sum_{t=1}^{\infty} E[G(t)]/t$ was equal to infinity, then the right–hand side of inequality (8.19) would be equal to minus infinity. However, the left–hand side of inequality (8.19) is nonnegative [Assumption 8.1(a)]. This proves the lemma. **Q.E.D.**

**Lemma 8.4.** The sequence $\left\{F\left(y(t)\right)\right\}$ converges, with probability 1.

***Proof.*** We take the conditional expectation of both sides of inequality (8.16), conditioned on $\mathcal{F}(t)$, and use Lemma 8.1(a) to obtain

$$E\left[F\left(y(t+1)\right) \mid \mathcal{F}(t)\right] \leq F\left(y(t)\right) + A_3 \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2} E\left[b^2(\tau) \mid \mathcal{F}(t)\right]. \tag{8.20}$$

Let

$$Z(t) = \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2} E[b^2(\tau) \mid \mathcal{F}(t)].$$

Using Lemma 8.1(b) and Lemma 8.3, we have

$$\sum_{t=1}^{\infty} E\left[Z(t)\right] = \frac{1}{1-\rho} \sum_{t=1}^{\infty} \frac{1}{t^2} E\left[b^2(t)\right] \leq \frac{1}{1-\rho} \sum_{t=1}^{\infty} \frac{1}{t^2} \left(A_1 E\left[G(t)\right] + A_2\right) < \infty. \tag{8.21}$$

With inequalities (8.20) and (8.21) available, a variant of the Supermartingale Convergence Theorem (Prop. D.7 in Appendix D) applies and shows that $F\left(y(t)\right)$ converges, with probability 1. **Q.E.D.**

**Lemma 8.5.** For each $i$, we have $\lim_{t\to\infty}\|y(t)-x^i(t)\|=0$, with probability 1.

***Proof.*** Inequality (8.21) has shown that

$$\sum_{t=1}^{\infty}\frac{1}{t^2}E\left[b^2(t)\right]<\infty. \tag{8.22}$$

Using the Monotone Convergence Theorem (Prop. D.4 in Appendix D), the infinite sum $\sum_{t=1}^{\infty}b^2(t)/t^2$ is finite and, in particular, $b(t)/t$ converges to zero, with probability 1. Given some $\epsilon>0$, let $T$ be such that $b(t)/t\le\epsilon$ for all $t\ge T$. Using inequality (8.13) for $t\ge T$, we obtain

$$\|y(t)-x^i(t)\|\le A\left(\sum_{\tau=1}^{T-1}\frac{1}{\tau}\rho^{t-\tau}b(\tau)+\sum_{\tau=T}^{t-1}\frac{1}{\tau}\rho^{t-\tau}b(\tau)\right)\le A\rho^{t-T}\sum_{\tau=1}^{T-1}\frac{b(\tau)}{\tau}+\frac{A\epsilon}{1-\rho},$$

which converges to $A\epsilon/(1-\rho)$, as $t$ tends to $\infty$. Since $\epsilon$ was arbitrary, the result follows. **Q.E.D.**

**Lemma 8.6.** For every $i$, we have $\lim_{t\to\infty}E\left[\|x^i(t)-y(t)\|^2\right]=0$.

***Proof.*** From Lemma 8.1(b) and Lemma 8.3,

$$\lim_{t\to\infty}E\left[\frac{b^2(t)}{t}\right]\le\lim_{t\to\infty}\left(\frac{A_1}{t}E\left[G(t)\right]+\frac{A_2}{t}\right)=0.$$

We now square both sides of inequality (8.13), use the elementary inequality $\left(\sum_{\tau=1}^{t-1}a_i\right)^2\le(t-1)\sum_{\tau=1}^{t-1}a_i^2$, and take expectations to obtain

$$E\left[\|y(t)-x^i(t)\|^2\right]\le tA^2\sum_{\tau=1}^{t-1}\frac{1}{\tau^2}\rho^{2(t-\tau)}E\left[b^2(\tau)\right].$$

Given some $\epsilon>0$, let $T$ be such that $E\left[b^2(\tau)/\tau\right]\le\epsilon$ for all $\tau\ge T$. Let $Q=\max_{\tau\le T}E\left[b^2(\tau)/\tau\right]$. Then for $t>T$,

$$\begin{aligned}E\left[\|y(t)-x^i(t)\|^2\right]&\le tA^2\sum_{\tau=1}^{T-1}\frac{1}{\tau}\rho^{t-\tau}Q+\epsilon tA^2\sum_{\tau=T}^{t-1}\frac{1}{\tau}\rho^{t-\tau}\\&\le t\rho^{t-T}A^2TQ+\epsilon A^2\sum_{\tau=T}^{t-1}(t-\tau+1)\rho^{t-\tau}.\end{aligned}$$

[We have used the trivial inequality $t/\tau = (t - \tau + \tau)/\tau \leq t - \tau + 1$.] As $t$ tends to infinity, $t\rho^t$ converges to zero, which shows that the first term in the right–hand side of the previous inequality vanishes. The limit of the second term is bounded by $\epsilon A^2 \sum_{\tau=0}^{\infty}(\tau + 1)\rho^\tau$, which is finite. By taking $\epsilon$ arbitrarily small, the result follows. **Q.E.D.**

Part (b) of the proposition follows from Lemmas 8.5 and 8.6 and the triangle inequality $\|x^i(t) - x^j(t)\| \leq \|x^i(t) - y(t)\| + \|y(t) - x^j(t)\|$.

**Lemma 8.7.** For every $x \in \Re^n$, we have $\|\nabla F(x)\|^2 \leq 2K_1 F(x)$.

***Proof.*** Using the nonnegativity of $F$ and Prop. A.32 in Appendix A, we have

$$0 \leq F\Big(x - \frac{1}{K_1}\nabla F(x)\Big) \leq F(x) - \frac{1}{K_1}\|\nabla F(x)\|^2 + \frac{K_1}{2}\frac{1}{K_1^2}\|\nabla F(x)\|^2$$

$$= F(x) - \frac{1}{2K_1}\|\nabla F(x)\|^2.$$

**Q.E.D.**

Since $F\big(y(t)\big)$ converges (Lemma 8.4), it is bounded. By Lemma 8.7, $\|\nabla F\big(y(t)\big)\|$ is also bounded by some constant $A_5$. Therefore, using Prop. A.32 in Appendix A,

$$\Big|F\big(x^i(t)\big) - F\big(y(t)\big)\Big| \leq A_5\|x^i(t) - y(t)\| + \frac{K_1}{2}\|x^i(t) - y(t)\|^2, \tag{8.23}$$

which converges to 0, with probability 1, by Lemma 8.5. Therefore, for each $i$, $F\big(x^i(t)\big)$ converges, with probability 1, to the same limit as $F\big(y(t)\big)$, which proves part (a) of the proposition.

**Lemma 8.8.** There holds $\lim_{t\to\infty}\|y(t+1) - y(t)\| = 0$, with probability 1.

***Proof.*** Equation (7.5) yields $\|y(t+1) - y(t)\| \leq b(t)/t$. But we have already shown, in the course of the proof of Lemma 8.5, that $b(t)/t$ converges to 0, with probability 1. **Q.E.D.**

We now recall Lemma 8.3 and the fact $E\big[G(t) \mid \mathcal{F}(t)\big] \geq 0$ (Lemma 8.1). We apply the Monotone Convergence Theorem to conclude that

$$E\left[\sum_{t=1}^{\infty}\frac{1}{t}E\big[G(t) \mid \mathcal{F}(t)\big]\right] = \sum_{t=1}^{\infty}E\left[\frac{1}{t}E\big[G(t) \mid \mathcal{F}(t)\big]\right] = \sum_{t=1}^{\infty}\frac{1}{t}E\big[G(t)\big] < \infty.$$

This implies that

$$\sum_{t=1}^{\infty} \frac{1}{t} E\big[G(t) \mid \mathcal{F}(t)\big] < \infty, \tag{8.24}$$

with probability 1. Let $B$ be a bound on the difference between successive elements of $T_\ell^i$ for every $\ell$ and every computing processor $i$ for component $\ell$. Since $\sum_{t=1}^{\infty} 1/t = \infty$, inequality (8.24) implies that

$$\liminf_{t\to\infty} \sum_{\tau=t}^{t+B} E\big[G(\tau) \mid \mathcal{F}(\tau)\big] = 0.$$

Using Lemma 8.1(a), we obtain

$$\liminf_{t\to\infty} \sum_{\tau=t}^{t+B} \sum_{\{(i,\ell)\mid \tau\in T_\ell^i\}} \big\|\nabla_\ell F\big(x^i(\tau)\big)\big\|^2 = 0.$$

By taking $t$ sufficiently large, the quantity $\max\big\{\big\|y(t) - x^i(\tau)\big\| \mid t \le \tau \le t+B\big\}$ can be made arbitrarily small, because of Lemmas 8.5 and 8.8. We then use the Lipschitz continuity of $\nabla F$ to obtain

$$\liminf_{t\to\infty} \sum_{\tau=t}^{t+B} \sum_{\{(i,\ell)\mid \tau\in T_\ell^i\}} \big\|\nabla_\ell F\big(y(t)\big)\big\|^2 = 0. \tag{8.25}$$

By the definition of $B$ and the assumption that there is at least one computing processor for each component, we see that for every component $\ell$ and time $t$, there exists some processor $i$ and some time $\tau \in [t, t+B]$ such that $\tau \in T_\ell^i$. Therefore, in the sum (8.25), the summand $\big\|\nabla_\ell F\big(y(t)\big)\big\|^2$ appears at least once for each $\ell$. Therefore,

$$\liminf_{t\to\infty} \nabla F\big(y(t)\big) = 0.$$

Since $\big\|x^i(t) - y(t)\big\|$ converges to zero and $\nabla F$ is Lipschitz continuous, part (c) of the proposition follows.

We now turn to the proof of part (d). Let $x^*$ be the unique minimizer of $F$ and fix some $i$. Consider a sequence of times along which $\nabla F\big(x^i(t)\big)$ converges to zero, which exists because of part (c) of the proposition. We restrict to a subsequence $\{t_k\}$ such that the sequence $\{x^i(t_k)\}$ converges to some $y^*$. [Such a sequence exists because $F\big(x^i(t)\big)$ is bounded (since it converges) and because $F$ is assumed to have bounded level sets; this implies that $x^i(t)$ is bounded and, therefore, has a convergent subsequence.] From the triangle inequality,

$$\big\|\nabla F(y^*)\big\| \le \lim_{k\to\infty} \big\|\nabla F(y^*) - \nabla F\big(x^i(t_k)\big)\big\| + \lim_{k\to\infty} \big\|\nabla F\big(x^i(t_k)\big)\big\| = 0.$$

Therefore, $\nabla F(y^*) = 0$ and, by our assumptions, $y^* = x^*$. Since $x^*$ is a limit point of the sequence $\{x^i(t)\}$, we conclude that $F(x^*)$ is equal to the limit of $F\big(x^i(t)\big)$. Let $z$ be an arbitrary limit point of $x^i(t)$. Then $F(z)$ is a limit point of $F\big(x^i(t)\big)$ and, since the latter converges to $F(x^*)$, we conclude that $z = x^*$. Therefore, $x^*$ is the unique limit point of $x^i(t)$. Since the sequence $x^i(t)$ is bounded, it converges to $x^*$. **Q.E.D.**

### 7.8.3 Discussion and Extensions

In the assumptions preceding Prop. 8.1 and in the proof, we have treated the coefficients $\Phi_\ell^i(t)$ as unknown but deterministic. This allowed us to go through the crucial step, in inequality (8.11),

$$E\Big[\nabla_\ell F\big(x^i(t)\big)'\Phi_\ell^i(t)s_\ell^i(t) \;\Big|\; \mathcal{F}(t)\Big] = \Phi_\ell^i(t)\nabla_\ell F\big(x^i(t)\big)'E\big[s_\ell^i(t)\,|\,\mathcal{F}(t)\big]. \tag{8.26}$$

However, if $\Phi_\ell^i(t)$ is a random variable, then Eq. (8.26) is not valid, in general.

To illustrate how Eq. (8.26) can fail to hold, suppose that the processors decide when to send messages to other processors, based on the results of their recent computations. For example, a processor could decide to inform the others whenever a substantial change in its state of computation occurs. This implies that the decision whether to transmit becomes a function of the random variables $s_\ell^i(t)$. In such a context, the underlying agreement algorithm has some probabilistic aspects and the coefficient $\Phi_\ell^i(t)$ will be, in general, a random variable. Under such circumstances, neither Eq. (8.26) nor the proof of Prop. 8.1 is valid. This is not a deficiency of the proof: one can construct examples in which the dependence of the transmission times on the state of computation of a processor can cause divergence of an otherwise convergent algorithm. The essence of such examples is that the processors can choose to communicate only when their most recent update is a direction of ascent. Then the coefficient $\Phi_\ell^i(t)$ will tend to be larger if $s_\ell^i(t)$ is a direction of ascent and the inner product $\nabla_\ell F\big(x^i(t)\big)'E\big[\Phi_\ell^i(t)s_\ell^i(t)\,|\,\mathcal{F}(t)\big]$ could become positive even though $\nabla_\ell F\big(x^i(t)\big)'E\big[s_\ell^i(t)\,|\,\mathcal{F}(t)\big]$ is assumed to be negative.

Notice that in the case of a "specialized computation" (exactly one processor per component; see Example 7.4 of the preceding section), the coefficients $\Phi_\ell^i(t)$ are exactly known and are deterministic, even if the underlying process of message receptions is random. For this particular case, Eq. (8.26) and the proof of Prop. 8.1 remain valid.

We indicate a few extensions. Notice that as time goes to infinity, the changes in the state of computation of each processor are smaller due to the stepsize decrease. Intuition then suggests that processors should be allowed to communicate to each other less and less frequently. This is in fact correct: convergence can be proved under the assumption that $t - \tau_\ell^{ij}(t)$ is bounded by $t^\beta$, where $\beta$ is a positive number smaller than 1, even though the geometric convergence of $\Phi_\ell^{ij}(t,\tau)$ [Assumption 7.1(d)] is lost ([Tsi84] and [TBA86]).

A last, but important, extension concerns the case where the stochastic descent assumption (8.7) fails to hold. An alternative and substantially weaker assumption requires

that if $x^i(t)$ is kept fixed and a sequence of steps $s_\ell^i$ is generated, then the long–run average of these steps is a descent direction. When the stepsize is very small, which is the case as $t$ tends to infinity, $x^i(t)$ remains approximately constant sufficiently long. Therefore, under a suitable continuity assumption on the dependence of $s_\ell^i(t)$ on $x^i(t)$, the average value of $s_\ell^i(\tau)$ (averaged over a time interval of the form $[t, t+T]$, where $T$ is large) is approximately the same as the average value of the steps $s_\ell^i$ generated at the constant (frozen) value of $x^i(t)$. But, the latter average was assumed to be a descent direction. This argument can be made rigorous and, under a suitable set of assumptions, convergence to a stationary point of $F$ can be proved ([Tsi84], [KuY87a], and [KuY87b]).

## EXERCISES

**8.1.** Verify that $V(t)$ converges to zero in Example 8.1. *Hints:* Use condition (8.4) to establish that $V(t)$ is bounded above. Conclude that there exist $\epsilon(t)$ such that $\sum_{t=1}^{\infty} \epsilon(t) < \infty$ and $V(t+1) \leq V(t) - 2\gamma(t)V(t) + \epsilon(t)$. Finally, use Eq. (8.3) to establish that $V(t)$ converges to zero.

## NOTES AND SOURCES

**7.1.** Proposition 1.1 and the geometric convergence result for partially asynchronous linear iterations (Exercise 1.2) are new.

**7.2.** The results of this section are from [TBT88] which also contains simulation results and additional examples. Some convergence results for synchronous nonexpansive iterations can be found in [BrP67] and references therein.

**7.3.** The asynchronous agreement algorithm and its convergence proof are from [Tsi84], where a more general version is considered. A different class of distributed agreement algorithms is studied in [BoV82] and [TsA84]. The partially asynchronous algorithm for Markov chain problems (Subsection 7.3.2) is from [LuM86], where partially asynchronous convergence was established. The relation of this algorithm to the agreement algorithm and the corresponding convergence proof are new.

**7.4.** The synchronous version of the load equalization algorithm has been suggested in [Cyb87], where a convergence proof is provided. The asynchronous algorithm and the convergence proof are new.

**7.5.** This section is based on the results of [Tsi84] and [TBA86], except for Prop. 5.3 which is new. Similar convergence results can be proved for more general classes of algorithms which can be made arbitrarily slow (through a stepsize parameter), provided

that their synchronous version remains convergent under diagonal scaling of the update directions.

**7.6.** The algorithm and the results of this section are based on [TsB86]. The behavior of the algorithm in the presence of stochastic fluctuations was investigated in [Tsa86], where it was shown that the conclusions of the deterministic analysis are valid in the limit of a large number of small users in the network. This reference, as well as [TTB86], contains a discussion of alternative implementations of the adjustment of the path flows, for a given OD pair, to their desired values. A related asynchronous flow control algorithm is analyzed in [San88].

**7.7.** The model of this section and its properties are from [Tsi84] and [TBA86].

**7.8.** Synchronous stochastic gradient–like algorithms have been extensively studied (see, e.g., [RoM51], [PoT73], and [Lju77]) and have found applications in several areas [LjS83]. The asynchronous convergence result of this section is from [Tsi84] and [TBA86] and the proof is patterned after the proof in [PoT73] for the synchronous case. Some extensions and some applications to system identification problems can be found in [Tsi84]. Stochastic algorithms have also been studied using the so called ODE approach ([Lju77] and [Kus84]). In this approach, convergence is established by approximating the sequence generated by the algorithm by the solution of a deterministic differential equation. The ODE approach has been used to analyze asynchronous stochastic algorithms in [Tsi84] and in [KuY87a] and [KuY87b], the latter two references using the machinery of weak convergence theory.